Australian Government
Department of Defence
Defence Science and
Technology Organisation

An Analysis of SE and MBSE Concepts to Support Defence Capability Acquisition

Meredith Hue

Defence Systems Integration Technical Advisory
Joint and Operations Analysis Division
Defence Science and Technology Organisation

DSTO-TR-3039

ABSTRACT

System modelling has been an enduring method of enquiry supporting systems analysis and design synthesis in systems engineering for decades. New generation systems modelling tools provide sophisticated modelling capability, coined Model-based Systems Engineering (MBSE). The underpinning fundamentals of systems engineering and MBSE are scrutinised in the context of the current Defence capability development process and enterprise architecture initiatives. The capabilities, relevance, and utility of new generation MBSE tools and methodologies are then examined, contrasting Defence and industry perspectives to reveal potential implications for Defence. Potential benefits to Defence are highlighted together with potential issues of concern. Related aspects of software engineering, enterprise engineering enterprise architecting and operations research are also clarified to assist unravelling some of the complexities and interdependencies between the respective professional disciplines.

Published by

Defence Systems Integration Technical Advisory
Joint and Operations Analysis Division
DSTO Defence Science and Technology Organisation
PO Box 1500
Edinburgh, South Australia 5111 Australia

Telephone: 1300 333 362
Fax: (08) 7389 6537

AR- 016-126
September 2014

An Analysis of SE and MBSE Concepts to Support Defence Capability Acquisition

Executive Summary

Model-based systems engineering (MBSE) is proffered by modelling tool vendors to provide improved ability to cope with the more onerous demands of engineering the larger scale and more complex capability systems aspired to by Defence.

The underpinning fundamentals of systems engineering and MBSE are scrutinised in this report in the context of the current Defence capability development process and enterprise architecture initiatives. The capabilities, relevance, and utility of next generation system modelling tools and methodologies are examined, contrasting Defence and industry perspectives to reveal potential implications for Defence.

It is evident there are multiple overlapping MBSE perspectives, somewhat similar, but with different problem foci and different problem solving approaches. If no common agreement can be achieved, these differences in perspective can introduce considerable ambiguity within the Defence stakeholder community; this can potentially exacerbate rather than resolve the problems at hand.

MBSE tool vendors posit that MBSE methodologies can offer improved flexibility, consistency and traceability, and facilitate easier upgrade of the associated information set. However, adoption of an effective MBSE approach by Defence would entail a Defence-wide methodological change to the capability development and acquisition processes. This has the potential for significant and widespread impact, spanning corporate management processes, engineering technical processes, governance and the tool environment within Defence. This change would have an inevitable impact on resourcing, staffing levels, staff skill-sets, training and support requirements.

Analysis in this report has also revealed a major divide between Defence, as the customer, and industry as the supplier, in terms of mindsets, skill sets, scale of endeavour, process requirements, constraints, and responsibilities. The methods of enquiry and utility of MBSE tools for Defence and industry will therefore differ markedly between the two mindsets and the differing responsibilities.

The differing utility of systems engineering expertise as perceived by Defence and industry is also a major differentiator. Due to the distributed responsibilities within the overall Defence capability lifecycle, Defence does not have a unified systems engineering approach, which has the potential to decouple the capability development process from the traditional systems engineering approach. This in turn introduces additional challenges, and can negate other efforts towards achieving the desired decision outcomes.

Defence faces a number of challenges in developing and applying sufficient systems engineering knowledge and experience both at the high-end platform and the System of Systems engineering levels to effect any major improvement to capability acquisition

outcomes. The current approach to capability development does not explicitly define the role of the systems engineer, instead, relying on process description in the Defence Capability Development Handbook to drive the capability development and acquisition process. Process governance relies on extensive scrutiny by numerous stakeholders from many perspectives, however, there is no independent scrutiny from a systems engineering perspective to ensure the systems engineering precepts are preserved.

The need to undertake systems analysis is inherent but not explicitly acknowledged within Defence. Of particular import, the capability development and acquisition process is document-centric and governance-oriented. Early capability definition activities are centred on development of the documentation and satisfying governance requirements rather than following a traditional systems engineering process.

Finally, it is important to distinguish between the concept of a methodology that is facilitated by a tool environment and the analytical capability of a tool modelling environment. Established MBSE methodologies such as the Rational Unified Process for Systems Engineering (RUP SE) and the Object-Oriented Systems Engineering Methodology (OOSEM) are modelling language dependent and implementation focused, and thus may offer potential cost savings and efficiencies in industry. However, they do not address the problem space posed to Defence. These established methodologies are therefore not necessarily suited for adoption in the Defence context.

Notwithstanding, MBSE tools can provide a powerful analytic capability, particularly to investigate capability and project interdependencies and propagation of capability system properties. This is contingent on the system models being set up correctly, used by knowledgeable practitioners, and the results are used within the correct context.

From a Defence enterprise architecture perspective, the new generation MBSE tools provide a useful means to create Defence Architecture Framework (DAF) artefacts using templates. The MBSE tools are evolving to support future developments of the UK MODAF and US DoDAF towards a common Unified Architecture Framework, embracing data-centric system modelling concepts. The latest AUSDAF2 view-based orientation does not provide a pathway towards supporting MBSE data-oriented constructs, nor the Unified Architecture Framework.

A separate study is recommended to investigate these issues further, including:

- The implications to Australian Defence capability development and acquisition and Defence enterprise architecture initiatives of the Unified Architecture Framework proposed developments;
- The feasibility and selection criteria for different information elements for incorporation in an enterprise-wide repository, and associated knowledge management process support requirements; and
- Provision of formal methodology guidance to leverage the potential of new-generation MBSE tools to achieve improved Defence capability acquisition and integration outcomes.

Author

Meredith Hue
Defence Systems Integration Technical Advisory
Joint and Operations Analysis Division

Meredith is responsible for providing advice on defence systems integration principles and practices targeting systemic problems in Defence, and working with projects to address specific system integration issues. A former Chief Engineer, she has over 35 years experience in both industry and Defence as a Systems Engineering practitioner, in the areas of real-time systems, combat systems and military communications. Specific interests include Systems, Systems of Systems, Enterprise Architecting and Systems Architecting methodologies supporting capability development, including modelling and analysis of C4ISR architectures.

This page is intentionally blank

Contents

Abbreviations and Acronyms

ADF	Australian Defence Force
ADGE	Air Defence Ground Environment
ADL	Architecture Description Language
ADM	Architecture Development Method
AGA	Australian Government Architecture
AGIMO	Australian Government Information Management Office
ANSI	American National Standards Institute
API	Application Program Interface
ARM	Architecture Review Meeting
ATM	Automatic Teller Machine
AUSDAF	Australian Defence Architecture Framework (also known as DAF)
BOK	Body of Knowledge
BPMN	Business Process Modelling Notation
BRM	Business Reference Model
C4ISR AF	Command, Control, Communications, Computing, Intelligence, Surveillance and Reconnaissance Architecture Framework
CAD	Computer-aided Design
CADM	Core Architecture Data Model
CAE	Computer-aided Engineering
CASE	Computer-aided Software Engineering
CCA	Circuit Card Assembly
CDD	Capability Development Documentation
CDG	Capability Development Group
CDRL	Contract Data Requirements List
CI	Configuration Item
CIOG	Chief Information Officer Group
COE	Common Operating Environment
CORBA	Common Object Request Broker Architecture
COTS	Commercial-off-the-Shelf
DAF	Defence Architecture Framework (also known as AUSDAF)
DCDH	Defence Capability Development Handbook
DCG	Defence Capability Guide
DCP	Defence Capability Plan
DFD	Data Flow Diagram
DM2	DoDAF Meta-Model
DMO	Defence Materiel Organisation
DND	Department of National Defence, Canada
DNDAF	Department of National Defence Architecture Framework (Canada)
DoDAF	Department of Defense Architecture Framework (U.S.)
DRM	Data Reference Model
EA	Enterprise Architecture
EAF	Enterprise Architecture Framework
ECIA	Electronic Components Industry Association

e.g.	For example
EIA	Electronic Industries Alliance (known as Electronic Industries Association prior to 1997)
EMC	Electromagnetic Compatibility
EMI	Electromagnetic Interference
FEA	Federal Enterprise Architecture
FEAF	Federal Enterprise Architecture Framework (U.S.)
FIC	Fundamental Inputs to Capability
FPGA	Field Programmable Gate Array
FPS	Function and Performance Specification
FSR	Force Structure Review
GEA	Gartner Enterprise Architecture
GIG	Global Information Grid
HSI	Human Systems Integration
HW	Hardware
ICT	Information and Communications Technology
IDA	Integrated Defence Architecture
i.e.	That is
IEC	International Electro-technical Commission
IEEE	Institute of Electrical and Electronic Engineering
INCOSE	International Council on Systems Engineering
IPPD	Integrated Product and Project Development
IPT	Integrated Project Team
ISO	International Standards Organisation
ISR	Intelligence, Surveillance and Reconnaissance
IT	Information Technology
JCIDS	Joint Capabilities Integration Development System
M3	MODAF Meta-Model
MBSE	Model-Based Systems Engineering
MDA	Model-driven Architecture
MDD	Model-driven Design
MDE	Model-driven Engineering
MDSD	Model-driven System Design
MODAF	Ministry of Defence Architecture Framework (UK)
MOTS	Military Off-the-Shelf
NAF	NATO Architecture Framework
NATO	North Atlantic Treaty Organisation
NCOIC	Network Centric Operations Industry Consortium
NCOSE	National Council on Systems Engineering (in the US)
NCW	Network Centric Warfare
NCWIIS	Network Centric Warfare Integration and Implementation Strategy
NIF	NCOIC Interoperability Framework
OCD	Operational Concept Document
OMG	Object Management Group
OMT	Object Modeling Technique
O-O	Object-Oriented

OOA	Object-Oriented Analysis
OOAD	Object-Oriented Analysis and Design
OOD	Object-Oriented Design
OOSE	Object-Oriented Software Engineering
OOSEM	Object-Oriented Systems Engineering Methodology
OR	Operations Research
OT&E	Operational Test and Evaluation
OTS	Off-the-Shelf
PCB	Printed Circuit Board
PMSA	Program of Major Service Activities
PMTE	Process Methods Tools Environment
PRM	Performance Reference Model
RFT	Request for Tender
RUP	Rational Unified Process
SADT	Structured Analysis and Design Technique
SDL	System Description Language
SE	Systems Engineering
SESA	Systems Engineering Society of Australia
SETE	Systems Engineering Test & Evaluation Symposium
SI	Systems Integration
SIE	Single Information Environment
SOA	Service-oriented Architecture
SOAD	Service-oriented Analysis and Design
SOMA	Service-Oriented Modelling and Architecture
SOP	Standard Operating Procedure
SoS	System of Systems
SoSE	System of Systems Engineering
SoSI	System of Systems Integration
Specs.	Specifications
SRM	Service Reference Model
SSADM	Structured Systems Analysis and Design Method
STEP	Standard for the Exchange of Product model data
SW	Software
SysML	System Modelling Language
TCD	Test Concept Document
TOGAF	The Open Group Architecture Framework
TRM	Technical Reference Model
UAF	United Architecture Framework
UHF	Ultra High Frequency
UK	United Kingdom
UK MOD	United Kingdom Ministry of Defence
UML	Unified Modelling Language
UPDM	Unified Profile for DoDAF and MODAF
U.S.	United States of America
US DoD	United States Department of Defense
VACRM	Verification Assurance Cross-Reference Matrix
VOA	Variability-oriented Analysis

V&V	Verification and Validation
WBS	Work Breakdown Structure
XMI	XML Metadata Interchange
XML	Extensible Markup Language
ZF	Zachman Framework for Enterprise Architecture

1. Introduction

1.1 Report Impetus

Pioneered by the Defence and Aerospace sectors in the 1940s, Systems Engineering (SE) practice has evolved significantly in recent decades as systems engineers have both enthusiastically progressed the development of new computer-based technology, then embraced this new technology to enhance their own SE development environments.

Despite these changes, systems modelling has remained an enduring method of enquiry supporting systems analysis and design synthesis in SE. New generation system modelling tools seek to extend the SE development environment even further by providing a more sophisticated systems modelling capability, coined MBSE (model-based systems engineering) in its most recent form, suggesting that another paradigm shift in SE might be in the offing.

From a Defence perspective, in recent years, capability development and acquisition processes have strained to cope with the increasing scale, complexity, and interdependency of capability as Defence has embraced the onset of the information age and adopted new concepts of networked warfare. MBSE is proffered by modelling tool vendors to provide improved ability to cope with the more onerous demands of engineering the larger scale and more complex systems as aspired in Defence.

To improve system integration outcomes, it is important to understand what is so difficult about systems engineering and Defence capability development. An inherent challenge of developing integrated systems that involve multiple technical disciplines is that each professional may be an expert in their respective field, but they will not be expert in the other disciplines. However, dependencies are created between components across different technical disciplines as the components interact. In order to properly design for this interaction, some technical professionals, typically the responsibility of the systems engineers in particular, must have some knowledge of the other technical disciplines.

Escalating design complexity is reported as one of the top challenges of system design (Boucher & Kelly-Rand, 2011). Lack of cross-functional knowledge and significant system complexity means that it can become very difficult to predict system behaviour. This means identifying system level problems early in development can also be a problem. This is a significant source of risk. Problems arising from system complexity are exacerbated by lack of integration of tools and methods across the technical disciplines, and differences in cultures, work practices, and semantics.

The utility or otherwise of SE fundamentals and MBSE-specific concepts, methodologies and tools for Defence purposes will be shaped by a number of factors, including:

- the extent to which SE and MBSE concepts are relevant to Defence;
- compatibility with the Defence's capability development process[1];
- governance requirements;
- enterprise architecture (EA) initiatives; and

[1] The Defence capability development process is described in (DCDH 2012).

- other policy and regulatory directives, as well as required competencies of Defence personnel.

This report firstly examines the underpinning fundamentals of SE and MBSE, then compares them to current Defence practice and reports on their relevance. The heritage of key concepts is also clarified to highlight their significance in shaping current notions of SE and MBSE, from both Defence and industry perspectives, highlighting similarities and dissimilarities, and again reporting on their significance. The capabilities of new generation MBSE tools and methodologies are then examined, contrasting Defence and industry perspectives, to reveal potential implications for Defence. Potential benefits to Defence are highlighted, together with potential issues that might arise to constrain this potential.

Related aspects of software (SW) engineering, enterprise engineering[2], enterprise architecting and operations research are also clarified to assist unravelling some of the complexities and interdependencies between the respective professional disciplines.

1.2 Scope

This report places particular emphasis on providing a baseline for fundamental SE concepts and terminology, particularly focussing on modelling methodologies in relation to the SE development environment. The relevance and utility of these are then considered in the Defence context. While the report refers to a number of specific concepts, methodologies, and tools, it does not aspire to provide particular detailed explanation on each of these. Instead, further information can be found in the cited references.

While the report aims to introduce and clarify relationships between various methodologies and the supporting tool environment to illustrate particular principles, it does not intend to provide a comprehensive review of MBSE tools and/or tool vendors, nor details of individual tool capability. A more comprehensive review of MBSE methodologies is provided in (Estefan 2008). A sample list of tools and vendors is provided in Appendix A of this report. Detailed information on the respective tools and supporting White Papers can typically be found on the respective vendor's websites.

[2] The term "enterprise engineering" is sometimes used interchangeably with "enterprise architecture", depending on the enterprise architecture framework. The focus, typically, is on the business operations of the enterprise rather than a particular engineering process or system. In particular, the US Federal Government uses the term as applied to their Federal Enterprise Architecture Framework (FEAF) (FEAF 2001).

2. Concepts of Methodology and the SE Development Environment

2.1 Significance of Methodology

As Mar and Morais astutely noted some ten years ago, SE can be difficult to implement if the words and framework are not clearly understood by all parties involved (Mar & Morais, 2002). While many aspects of SE, including tools and methodologies, have evolved over the decades, the underlying fundamental concepts of SE have not changed, despite the passage of time (Mar 1997).

Adopting an MBSE approach in Defence is therefore not simply a matter of purchasing MBSE tool licenses and tool-user training courses by individuals at their own discretion. Leveraging the power of the new generation MBSE tools will require a much more considered approach in the Defence context, underpinned by a strong foundation of SE fundamentals.

When considering the utility of models and methods of model development to inform the SE process, it is essential to have a common understanding of the SE terminology and inferred meaning of the terms used, in the correct context. A useful starting point is to understand how models might relate to SE, and what is model-based systems engineering (MBSE).

In the same way that SE practice is undertaken, MBSE practice also manifests in a practical sense in terms of various methodologies that can be applied to different classes of engineering problems in particular environmental settings. An MBSE methodology can be considered as a collection of related processes, methods and tools used to support the discipline of SE in a "model-based" or "model-driven" context (Estefan 2008).

Here, the methodology is not just a collection of specific process steps but is an aggregation of several constituent parts as shown in Figure 1 where:

- A Process (P) is a logical sequence of tasks performed to achieve a particular objective, i.e. it defines "WHAT" is to be done without specifying "HOW" each task is to be performed.
- A Method (M) is a technique for performing a task to achieve a particular objective, i.e. it defines the "HOW" each task is to be performed. The term "method" is often used interchangeably with the terms "technique", practice" and "procedure".
- A Tool (T) is an instrument that, when applied to a particular method, can enhance the efficiency of the task, provided it is properly applied, and by someone with appropriate skills and training.
- The Environment (E) comprises the surroundings, the external objects, and conditions or factors that can influence the actions of an object, individual person, or group. These factors can be social, cultural, personal, physical, organisational or functional (Estefan 2008), (Sage 1992).

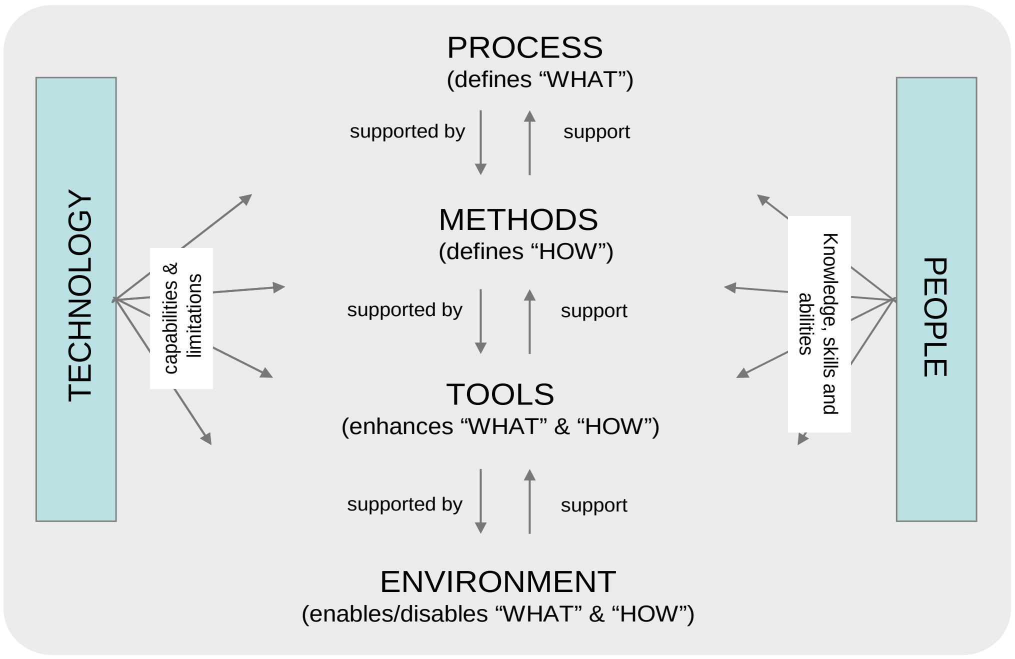


Figure 1. PMTE Elements and the Impact of Technology and People (Estefan 2008).

Notably, a process can be structured in a hierarchy which provides several levels of process aggregation - to support analysis and synthesis at different levels of abstraction to support different decision-making needs.

At any level of aggregation, process tasks are performed using methods. Since each method is also a process itself, comprising a sequence of process steps to be performed for that particular method, the "HOW" at one level of abstraction becomes the "WHAT" at the next lower level. This notion of recursiveness is widespread in SE, imbuing the ability to present either higher levels of abstraction with broader scope, or increased fidelity of representation with much narrower scope, without compromising the integrity of the description of individual piece-parts, the nature of their interactions, the relationships between the piece-parts, or the interactions with the external environment.

Importantly, the utility of the methodology is dependent on the people involved; the state of technology pertaining to both the SE development environment and the system undergoing engineering development; and the influence of the external environment, as well as the nature of the problem itself to be solved.

A Project Environment would thus be expected to integrate and support the use of the methods and tools used on a particular project to address a particular engineering problem, mindful of the skills and corporate resources available and suitable to bring to bear for the task, and the state of technology envisaged to drive the engineered system solution.

A Defence-wide methodological change to capability development and acquisition thus has the potential for significant and wide-spread impact, spanning corporate management processes, engineering technical processes, governance, and the tools environment, along with the inevitable impact on resourcing, staffing levels, staff skill-sets, training and support requirements. It is therefore crucial to understand what concepts are enduring, what change is necessitated to accommodate changing Defence needs and constraints, and what additional change may be precipitated if an MBSE approach was adopted in Defence.

2.2 Significance of Scale

The effectiveness or otherwise of a particular methodology to support acquisition of a new Defence capability system, or a major upgrade, will to a large extent, be determined by the scale and complexity of the problem to be addressed, and the nature of the engineering endeavour required.

For the most part, Defence capability systems are very large scale and very complex, costing many millions of dollars, and potentially affecting many thousands of Defence personnel, whether they are operators or maintainers. These systems can range in size from major warfighting platforms (e.g. submarines, tanks, aircraft), to fleets of specific equipment (e.g. UHF radios, sonobuoys, BDU-33 practice bombs), to individually licensed desktop computing applications such as Microsoft Office™.

The scale of engineering development can span a huge range of activity, from:

- designing an individual integrated circuit such as a custom processor or microcontroller;
- programming an integrated circuit such as a Field Programmable Gate Array (FPGA);
- designing a printed circuit board (PCB) containing devices such as microprocessors, microcontrollers and FPGAs;
- programming a microprocessor with embedded SW;
- designing equipment which includes multiple circuit card assemblies (CCAs) housed in an equipment enclosure or chassis, which may or may not be programmable;
- designing a dedicated purpose SW application which can be hosted on a general purpose desktop computer or operated in a distributed manner across a network;
- designing a dedicated SW application to be hosted on dedicated HW within a military system; to
- designing a large scale military platform.

A simple but useful categorisation in terms of scale of engineering endeavour is provided in Table 1 (adapted from Landherr 1997), where Defence capability systems are comprised of multiple assemblies of black boxes forming subsystems, systems, and systems of systems as described in Section 6.

Table 1. Scales of Engineering Development.

Activity	**Scope**			**Description**
	HW	*SW*	*Mechanical*	
Programming level	-	X	-	Ability to modify the source code but not to change the fundamental system design.
Circuit Card Assembly level	X	-	X	Ability to assemble different devices and/or sub-assemblies, and/or programmed or non-programmed electronic devices to modify the circuit card capability, but not to change the fundamental system design.
SW Linkage level	-	X	-	Ability to link selected SW modules together to produce different versions of the executable program.
Equipment Assembly level	X	-	X	Ability to assemble different HW modules (either programmed or non-programmed circuit card assemblies) together to change the system configuration.
Executable SW level	-	X	-	Ability to copy and load executable programs onto host computing platforms, and change the system configuration, but not to modify the program.
"White Box"	X	X	-	Ability to execute embedded programs and to perform diagnostic functions. No ability to change the system configuration.
"Black Box"	X	X	X	Ability to execute embedded programs without any visibility of the internal composition of the system. No ability to change the system configuration.
System Design level	X	X	X	Ability to fundamentally modify the system.
System of Systems level	-	-	-	No ability to fundamentally modify the hardware, software, or mechanical aspects of each system. Ability to apply SoS specific policy and provide guidance to influence system level design or purchase of each component system within the SoS.
Enterprise	-	-	-	No ability to fundamentally modify the hardware, software, or mechanical aspects of each system. Ability to apply pan-organisational policy and provide guidance to influence system level design or purchase of each component system within the policy remit.

The notion of the "Black Box" in Table 1 is significant in that it is a configuration item (CI) of equipment or an assembly of equipment which has a specific system identity and known configuration, and comes within specific SE life cycle management purview over the life-of-type of the Black Box. Subsequent assemblies of "Black Boxes" to form larger Defence systems, SoS, and Defence capability systems may or may not necessarily be identified as separate configuration items in their own right, with specific system identity, managed configuration, and with separate discernible system life cycle management.

Each level of engineering activity of Table 1 requires engaging different skill sets in different engineering development environments, with different process support, methodologies and tools. SE principles are applicable at all these scales of development, and across the range of engineering-oriented technologies, whether it is mechanical, electrical, electronic HW, SW or a combination thereof. However, the instantiation of the SE process in terms of specific methods and procedures, and degree of formality in process application (e.g. repeatability of process) will differ to suit the scope and nature of the development activity required, the extent of risk to be managed, and to meet any regulatory or governance requirements.

3. System Modelling Concepts

"All models are wrong but some are useful." George Box, 1979.

3.1 Modelling Concepts in Engineering

Before exploring the more elaborate notion of MBSE as it might apply to Defence, it is essential to establish a common understanding of some basic notions of systems modelling within the engineering disciplines.

In simple terms, a systems model in the engineering context is a semantically closed abstraction of a system, providing a simplified representation of reality (or potential reality). Abstraction is the suppression of irrelevant detail. Abstraction is an intrinsic response of the human mind as it relentlessly seeks to make sense of perceptions. These abstractions can form in various ways, including making generalisations, deleting detail, and forming distortions (i.e. different perspectives) (Dickerson & Mavris 2010).

Models are formed to enable a better understanding of a much more complex situation under consideration from a particular perspective. They are especially useful to represent particularly complex matters because the human mind cannot comprehend all the intricate and implicit interactions and interdependencies except for the simplest of problems (Rosenblueth & Norbet 1945), (Lieberman 2003a), (Friedenthal et al. 2008).

Models can be characterised in the way they are used, for example:

- they can be used prescriptively, to specify behaviour or a course of action;
- they can be used predictively, to predict possible future outcomes in light of different decisions or actions, or
- they can be used descriptively, to explain or describe a problem, phenomena or system to assist understanding (Pidd 2004).

Models have been used extensively for decades to investigate engineering-related problems, including:

- to help visualise a system or part of a system as it is, or how it is required to be;
- from an architectural perspective, to specify the structure or behaviour of a system, or describe how the parts might relate to each other, or behave dynamically with each other;
- to provide a template to guide the construction of a system, or to inform how to combine the parts together;
- to undertake explicit formal enquiry to identify alternate courses of action or possible outcomes to assist in decision making in relation to systems analysis and systems synthesis (i.e. systems design);
- to document design decisions that have been made during the development of a new system or parts of a system (Rechtin 1991), (Booch et al. 1999), (Maier & Rechtin 2002).

Models can take on many different forms: from a simple sketch to conjecture or communicate key ideas; to sets of equations to implement algorithms; to models which can be used to generate documentation and /or SW code; to models which can provide extensive engineering process support.

Computer-based SE and SW engineering tools have been used to support requirements management, systems analysis and systems synthesis, configuration management, automated document generation, and automated SW code generation since the 1980s. Graphical modelling tools have also evolved to replace text-based tools to help manage or reduce the complexity of expression to improve communication between system stakeholders.

A number of specific modelling techniques have been published over the years to address certain classes of problems, particularly relating to SW development. Formal methods and supporting computer-based modelling tools have evolved significantly from ad hoc beginnings to systematise systems analysis and systems synthesis in particular,[3] to cope with increasing complexity and ambiguity in the systems problems taken on, and the commensurate complexity inherent in the resultant system solutions.

Models are particularly useful in engineering to understand or predict properties or characteristics, behaviour, functional performance, and logical consistency to assist with system implementation. They can also be used to describe system processes, data, and data flows. Their utility stems from the ability to use precise modelling constructs and process descriptions to improve precision of expression and avoid the ambiguity that is often found in natural language descriptions (Hawryszkiewycz 1988), (Rechtin 1992).

However, the utility of a model is dependent on a number of considerations, including:

- the ability to acquire valid source information relating to the key characteristics and behaviours of interest;
- understanding the impact of simplifying approximations and assumptions; achieving the fidelity required; and
- establishing the validity of the model outputs.

Models can be manipulated to reduce the misfit between the model and the real world, but it requires real-world measurement to test the model prediction or explanation against measurements or observations to provide validation of the model. However, this may not necessarily be feasible, particularly for those circumstances where the input conditions cannot be adequately controlled, or the input and control conditions cannot be replicated.

3.2 Modelling vs. Simulation

It is also requisite to understand the difference between the concepts of modelling and simulation in an engineering context, where simulation is typically an imitation of a real-world process or system over time. The act of simulating something first requires that a model be developed; the model representing the key characteristics or behaviour of the selected physical or abstract system or process under scrutiny. It is then possible to show anticipated effects of alternate conditions and courses of action (Rechtin 1991), (SEF 2001).

[3] This is sometimes referred to as "systems analysis and design", as used in (Blanchard & Fabrycky 1998).

Models and the process of simulation can provide a more timely and resource-effective means of obtaining factual information (and therefore providing underpinning rigour) in lieu of building and testing prototype alternatives to allow a system design to converge to an acceptable system solution.

The primary purpose of simulation in SE is to explore the effects of alternative system characteristics on system performance without building and testing real-world alternatives. This might otherwise be a time consuming and costly process, and may not necessarily be feasible to undertake. Modelling and simulation can therefore offer an attractive alternative to expedite investigations in pursuit of the most cost-effective and swift implementation in a resource constrained SE development environment. Heuristically, cost escalates with schedule escalation. This can arise through reduced efficiency in implementation, and/or increased opportunity cost of lost product sales due to a potential reduction in the marketing window of opportunity.

The importance of systems modelling and simulation to SE is exemplified where typically, a major part of the design process relies on decisions made based on a model of the system (either current or proposed alternative) rather than decisions derived from a real-world system instantiation (Blanchard & Fabrycky 1998).

3.3 What is a Model?

3.3.1 Real World Models

What is meant by the term model in the engineering context? In its most basic form, a model is anything used to represent anything else for the purpose of informing and facilitating understanding about the subject matter they represent; whether the subject matter relates to the real world or whether it is conceptual. Models are used in many scientific disciplines, ranging from hard science to social, political, economics and management sciences for this same purpose. Models can thus take on many forms, from representing systems, processes, information, and operations, to representing organisations, depending on the nature of the enquiry. Many of these can be applied to systems problems (Blanchard & Fabrycky 1998), (Lieberman 2003b).

It is import to distinguish between what models are, and what the models are models of, when applied to systems problems. In the real world, for example, a physical model of a car can be constructed as a scale 1:10 representation. Its purpose may range variously:

- to convey alternate possible car design configurations for evaluation of aesthetics and functionality prior to selection of the preferred alternative;
- as a representation of a new car model about to be launched on the market for advertising purposes; or
- to simply to demonstrate the operation of a new door opening feature.

In a simple example of a model car, the detail included in the model will differ according to the purpose the model was constructed, whether for aesthetic evaluation, functional evaluation, or market evaluation; to decide on future possibilities, or to highlight current or past features.

In the same vein, a weather map can be regarded as a model of the weather patterns over a designated geographic region, and can include a wide variety of information. Model parameters can include, for example, measured or predicted pressure, wind speed and direction, temperature, precipitation, humidity, river flows, high and low tide levels and times, and sunrise and sunset times. A weather map published in a newspaper can report past or present measured weather conditions, or depict future predicted weather conditions, depending on the purpose the weather map was created.

However, it is important to note that the model representation in the newspaper only reports the parameter values determined by alternative means; the tool used to draw the weather map does not necessarily provide the ability to generate the parameter values displayed in the model. Similarly, the nature of any relationships between parameters cannot be inferred from the information displayed within the model unless a definition of the type of relationship is incorporated in the modelling technique - even though relationships may exist between different parameters. The weather map is an example of a non-architectural model.

In each of these instances, a common feature is the correspondence with the real world. The value of the model is proportional to how well it exemplifies a past or present actual real-world implementation, or, potential future real-world implementation, in terms of the information presented for the purpose intended, or understanding sought.

3.3.2 Conceptual Models

Conceptualisation from observation of physical existence and conceptual modelling are the necessary means people employ to think and solve problems. Concepts are used to convey semantics using natural language based communication. If the concepts in the mind of one person are very different to the concepts in the mind of the other then there is no shared model of the topic, and therefore no effective communication. Effective human communication entails:

- Translation of one person's ideas into the other's understanding;
- Embedding those ideas within the other's mental model;
- Maintaining those ideas with constant and consistent reinforcement; and
- Verifying the validity of the ideas and their translation for further action (Rechtin 1991).

The greater the number of people involved in the conceptual activity, the greater the challenge in arriving at, and maintaining sufficient shared understanding. Since a concept might map to multiple semantics by itself, an explicit formalisation is usually required for identifying and locating the intended semantic from several candidates to avoid misunderstandings and confusion in the conceptual models. This is critically important from an engineering perspective to ensure the right outcome is achieved for the right problem (Pidd 2004).

Conceptual models can be used to explore many different types of concepts, ranging from different views of stakeholders in an organisation to knowledge representation of subject matter experts, to explore different representations of "truths" or possible consequences from different perspectives.

When applied to systems problems, a model of a concept is quite different to a real-world model in that it does not need to have real-world correspondence to be a good model. Conceptual models are typically used by analysts who are not concerned with the truth or falsity of the concepts being modelled, but wish to clarify understanding by problem structuring or articulating different notions or perspectives (Gregory 1993).

In SE, conceptual modelling is used to promote effective human communication between client and system designer. A dialogue typically ensues between the client and the system designer to exchange ideas of what the system might do, and what it might look like. In the process, the conceptual model takes form and evolves to provide the basis from which subsequent design activity can be undertaken (Rechtin 1991).

A plethora of conceptual models can be drawn from numerous scientific disciplines to undertake systems analysis to inform system design; a snapshot of which is provided in Table 2 [4].

Table 2. Conceptual Model Types.

Conceptual Model Type	**Description**
Mental Models	A representation of something in the mind. Can also be a non-physical external model of the mind itself (Lieberman 2003a), (Jones et al. 2011).
Logical Models	A relational structure for which the interpretation of a logical sentence (in the predicate calculus) becomes valid. The relational structure is referred to as a model of the sentence. A relation is an assignment of a mathematical function of one or more arguments (or logical variables) whose range is the set of truth values {true, false} (Dickerson & Mavris 2010). A type of interpretation under which a particular statement (i.e. interpretation of a logical sentence) is true (Taha 2002). Two broad categories: • Those which only attempt to represent concepts (e.g. mathematical models) (Chang et al. 1990) • Those which attempt to represent physical objects and factual relationships (e.g. scientific models).
Mathematical Models	Can take many different forms using a variety of abstract structures (e.g. dynamical systems representations; statistical models; differential equations; game-theoretic models). Can also be a theoretical construct that represents processes by a set of variables and a set of logic and/or quantitative relationships between them. The model can have various parameters which can be changed to create various properties

[4] [online] URL http://en.wikipedia.org/wiki/Conceptual_model. (Rechtin 1991) and (Blanchard & Fabrycky 2010) also provide a useful discussion on the role of modelling in supporting systems analysis in system engineering. (Pidd 2004) provides a useful discussion from a complementary operations research perspective.

	(Taha 2002), (Pidd 2004), (Dickerson & Mavris 2010).
Scientific Models	Provide a simplified abstract view of the complex reality. Can represent empirical objects, phenomena, and physical processes in a logical way. Seeks to formalise principles of the empirical sciences using an interpretation to model reality.
Statistical Models	Provide a probability function for generating data (Taha 2002). These can take two forms: • parametric models - where the probability distribution function has variable parameters. • non-parametric models - provides a distribution function without parameters, and is only loosely confined by assumptions.
System Architecture Models	Describe mutually interdependent systems concepts of: • structure – what major elements are, how they are organised and decomposed, functionality, interfaces, and ties to system requirements • layout – physical arrangement, packaging and location of design aspects • behaviour – system dynamics response to events to providing a basis for reasoning about the system. Can represent multiple views of a system by using two different approaches: • non-architectural approach – a model is created for each view • architectural approach – single integrated model is created encompassing all required views. Can be used to model concepts or real world objects and events (Eeles 2006a), Maier & Rechtin (2002).
Data Models	Also known as data structure, in SW engineering, is an abstract model that describes how data is represented and accessed. Formally define data elements and relationships between data elements for a domain of interest. Provide various means of describing system data (in SW engineering and enterprise engineering): • Entity-Relationship Model – an abstract and conceptual representation of data to develop a conceptual schema or semantic data model of a system (e.g. relational database) (Chen 1976). • Domain Model – used to depict the structural elements and their constraints within a domain of interest (e.g. problem domain), including the

	various system entities, their attributes and relationships, with constraints governing the conceptual integrity of the structural model elements comprising the problem domain. Domain model can include a number of conceptual views where each view is relevant to a particular subject area of the domain or to a particular subset of the domain model that is of interest to a stakeholder of the domain model. Can be used to model concepts or real world objects and events (Cantor 2003a).

3.3.3 Decision Modelling

Insight can also be gained using systems analysis by formulating and manipulating decision models to determine how changes in those aspects of the decision under control of the decision maker affect the modelled system. This allows evaluation of a probable outcome of a decision without disturbing the current operational system itself (Taha 2002).

Models for operational decisions and design decisions are abstractions of the system under study. However, like all abstractions, models can make many assumptions – about the operating characteristics of the components; about the behaviour of people; and about the nature of the environment. The implications of these assumptions must be understood and evaluated when the models are used to aid decision making in design and operations.

Notably, a decision model cannot be classified as accurate or inaccurate in any absolute sense; to validate model manipulation would require reality manipulation. A decision model is therefore difficult to test except for an intuitive check for reasonableness (Blanchard & Fabrycky 1998).

3.3.4 Information Model

An information model is a different but important concept in systems and SW engineering. An information model is an abstract formal representation of concepts or real-world objects, and the relationships, constraints, rules, and operations to specify data semantics for a chosen domain. In SW engineering, it is typically used to provide a sharable, stable and organised structure of knowledge in the domain context.

An information modelling language is used to specify the notations representing the information in the information model. The ICAM[5] Definition (IDEF) Language IDEF1X graphical representation in particular, is widely used in systems and SW engineering.

IDEF1X is a data modelling language standard for the development of semantic data models. It used to produce a graphical diagram that represents the structure and semantics of information within a domain. The basic constructs of an IDEF1X model are:

1. Things about which data is kept, such as people, places, ideas, events, represented

[5] Published as Federal Information Processing Standards Publication 184 (IDEF1X 1993)

by a box;

2. Relationships between those things, represented by lines connecting the boxes; and
3. Characteristics of those things represented by attribute names within the box.

The basic constructs of an IDEF1X model are shown in Figure 2 (IDEF1X 1993).

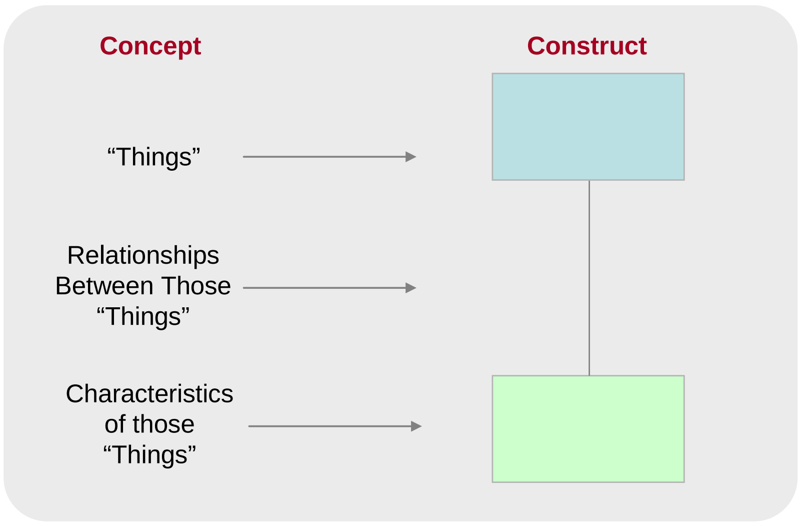

Concept
Construct
"Things"
Relationships Between Those "Things"
Characteristics of those "Things"

Figure 2. Basic IDEF1X Modelling Concepts

Importantly, the information model provides formalism to the description of a problem domain without constraining how that description is mapped to an actual implementation in software. There may be many mappings of the information model; these are called data models, regardless of whether they are object models, entity relationship models or XML schemas.

3.3.5 Meta-modelling

Meta-modelling is a related concept, often used in mathematics, computing science, SE and SW engineering, entailing the analysis, construction and development of the frames, rules, constraints, models and theories applicable and useful for modelling a predefined class of problems.

A meta-model is a higher-level abstraction of a model, highlighting the properties of the model within a certain domain. A model conforms to its meta-model in the way a computer program conforms to the grammar of the programming language in which it is written[6].

Meta-models can be viewed from three different perspectives:

- As a set of building blocks and rules to build models;
- As a model of a domain of interest; and

[6] [online] URL: http://en.wikipedia.org/wiki/Metamodeling.

- As an instance of another model[7].

Meta-models are closely related to ontology, which is also used to describe and analyse relationships between concepts, providing specific grammar, controlled vocabulary (non-redundant and unambiguous) and explicit semantics to express something meaningful within a particular domain. When a meta-model is used as a model for a domain of interest, the ontology is the meta-model together with the data set in the domain of interest.

An example of a meta-model is provided in Figure 3[8].

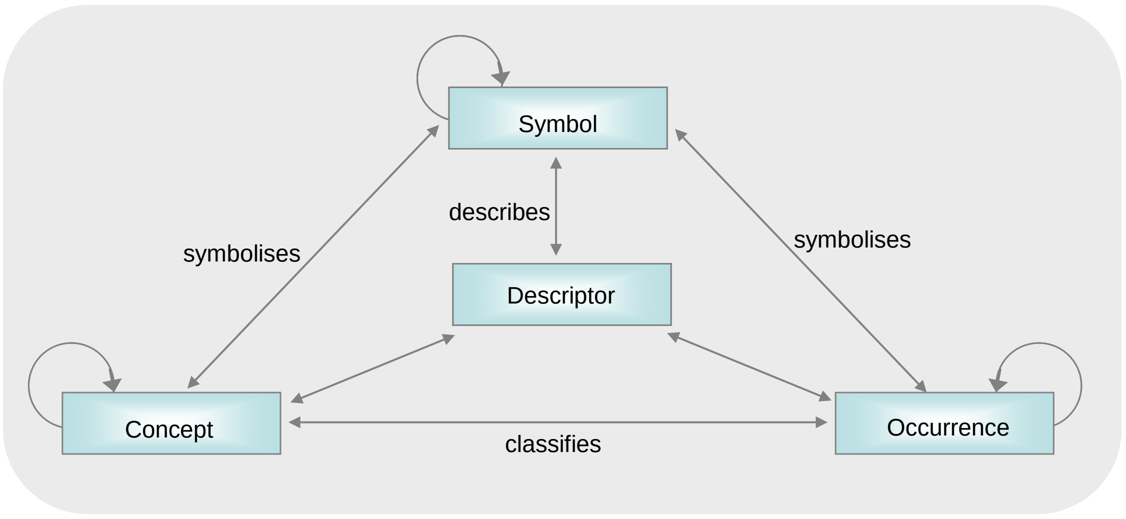


Figure 3. Information Meta-model with Four Different Meta-objects and their Relationships.

3.3.6 Architecture Modelling

The notion of architecture modelling, drawing initially from the principles of IEEE 1471, then subsequently superseded by ISO/IEC/IEEE 42010[9], has become more prominent over the last ten years to supplement classical systems analysis and design activity. The impetus stems from the need to simplify knowledge representations of very large-scale complex systems and systems-of-systems (SoS) whilst preserving the integrity of the underlying (and more complex) relationships (Maier & Rechtin 2002).

Architecture-based modelling enables a specific focus to be cast on a set of prescribed relationships across a domain of interest, which can contain SoS, systems or components of interest. A simple example considering a car as a system is illustrated as follows. Figure 4 reveals the static structure (i.e. system architecture) of the car (but only the major components of interest associated with the anti-lock braking system). Figure 5 shows the

[7] [online] URL: www.metamodel.com
[8] [online] URL: http://en.wikipedia.org/wiki/Metamodeling.
[9] ISO/IEC/IEEE 42010 superseded IEEE 1471 in 2007. However, because of the pivotal role of IEEE 1471 in shaping the concept of architecture descriptions, it is still widely referred to in the context of architectural modelling.

dynamic behaviour between some of these components making up the car anti-lock braking system to provide specific functionality (e.g. accelerating and braking capability) in the car.

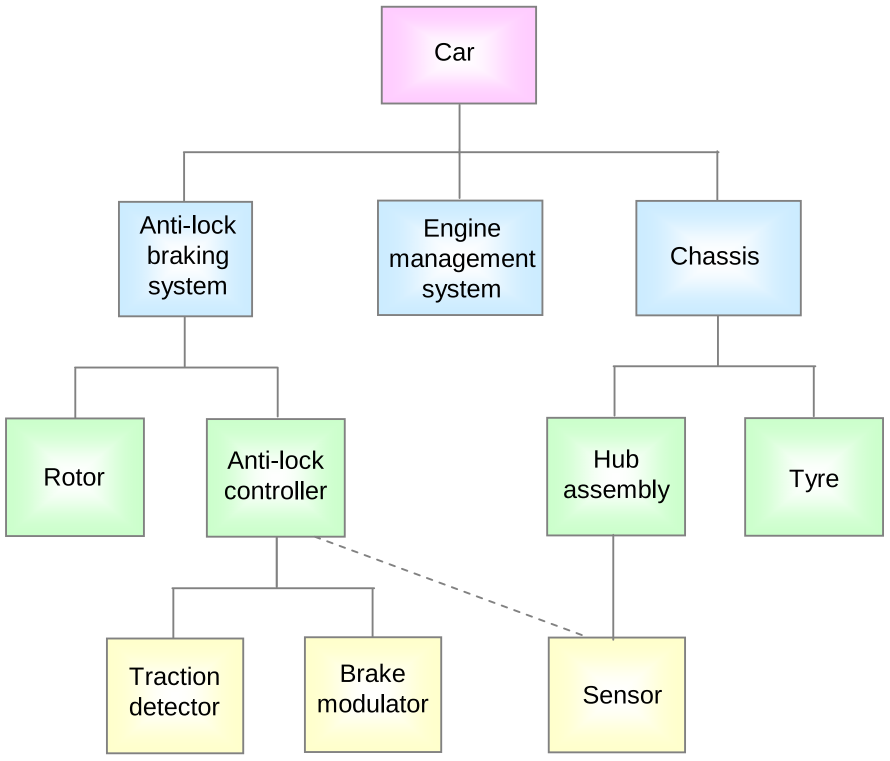


Figure 4. *Simple Architectural Model for a Car System (adapted from Shamieh 2011).*

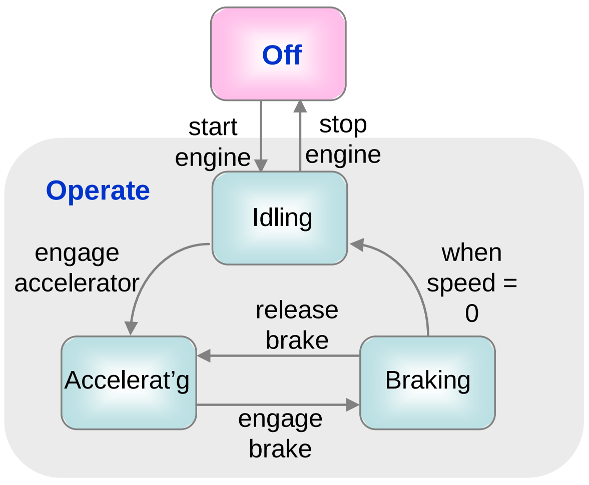


Figure 5. *Simple Behavioural Model for a Car System (Shamieh 2011).*

An example of an architecture modelling activity using a computer-based modelling tool and a defined graphical modelling language is shown in Figure 6[10],[11].

In recent years the concept has been applied more broadly to organisational entities within the fields of enterprise engineering and enterprise architecting as described in Section 9 Enterprise Architecture concepts and Section 10 Defence Enterprise Architecture context.

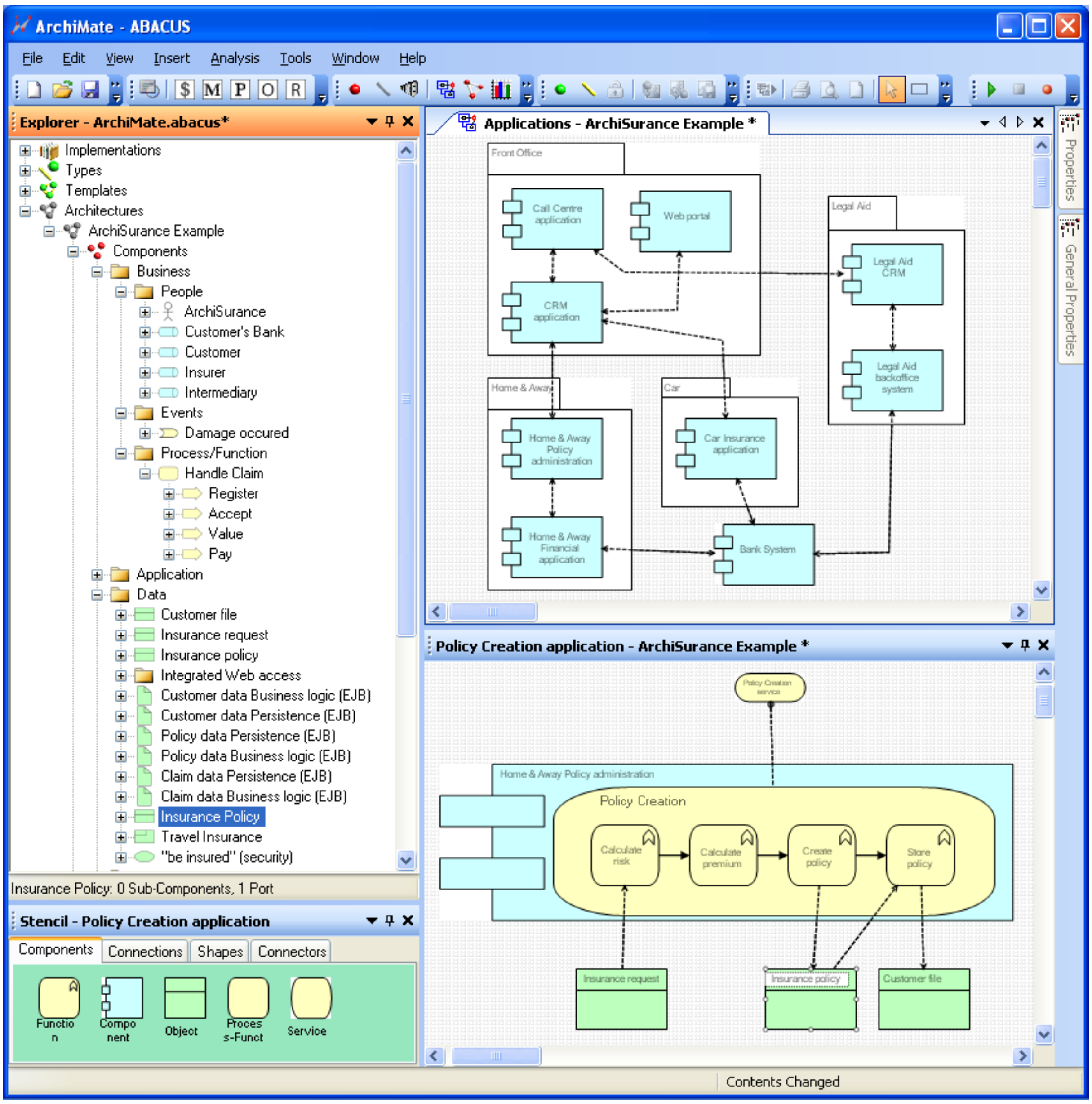


Figure 6. Example of Architecture Modelling Using the Archimate® Modelling Language and ABACUS modelling tool.

[10] [online] URL: http://www.avolution.com.au/releases/0809_archimate.html
[11] Vitech Corporation has devoted an entire book to explain how MBSE principles can be applied to architecture modelling using their MBSE tool, CORE (Long & Scott 2011).

3.3.7 SW Architecture Modelling

International SW engineering standards IEEE l471 and its ISO replacement ISO/IEC 42010 lay down basic terms, principles and guidelines for the consistent application of architectural precepts to systems throughout their life cycle. They also provide a framework for the collection and consideration of architectural attributes and related information for use in application of other IEEE standards. Most importantly, IEEE 1471 offers a widely accepted definition and a prescriptive meta-model to enable a description to be crafted of a SW architecture.

IEEE 1471-2000 offers a definition of ***software architecture*** as:

"the fundamental organisation of a system,
embodied in its components;
their relationships to each other and the environment;
and the principles governing its design and evolution".

The scope of the standard spans the creation, analysis and sustainment of architectures of SW-intensive systems (including IT systems or information systems), including recording the architecture in terms of architectural descriptions as described in Figures 4, 6 and 6.

A key tenet of the IEEE 1471 Conceptual Framework is the notion of multiple views of the data set comprising the system description. An architectural description is organised into one or more architectural views of the system; the particular views selected being dependent on the particular technique used.

IEEE 1471 was deliberately framed to be life cycle neutral, and independent of method, technique, notation, media, and format. The IEEE 1471 information model or meta-model shown in Figure 7 is agnostic to the process used to obtain the information to populate the model, and does not necessarily provide an ability to attribute meaning or context to the information contained in the model.

With regard to semantics, the standard makes an important distinction between the notions of architecture and architecture description. In the context of the standard, the architecture of the system is conceptual, and is a fundamental characteristic of the system. The architecture comprises the set of elements depicted in in the architectural model, and the links between the inter-related elements. The architecture description is a tangible artefact that records the details in the data set of elements and links that comprise the architecture (Hilliard 2000).

It is important to note that this interpretation differs markedly from that used in a number of view-centric enterprise architecture frameworks (EAFs) including TOGAF, AUSDAF[12], and the initial version of DoDAF, as described in Sections 9 and 10. These EAFs blur the distinction between the two terms and use them interchangeably, whilst inferring the meaning to be pertaining to the tangible artefacts. View-centric EAFs are therefore agnostic to the notion of an architecture described using an integrated architecture model, notwithstanding any EAF references to the use of the IEEE 1471 standard in regard to preparation of different artefacts to describe different views.

[12] Version 1 is also referred to as "the DAF".

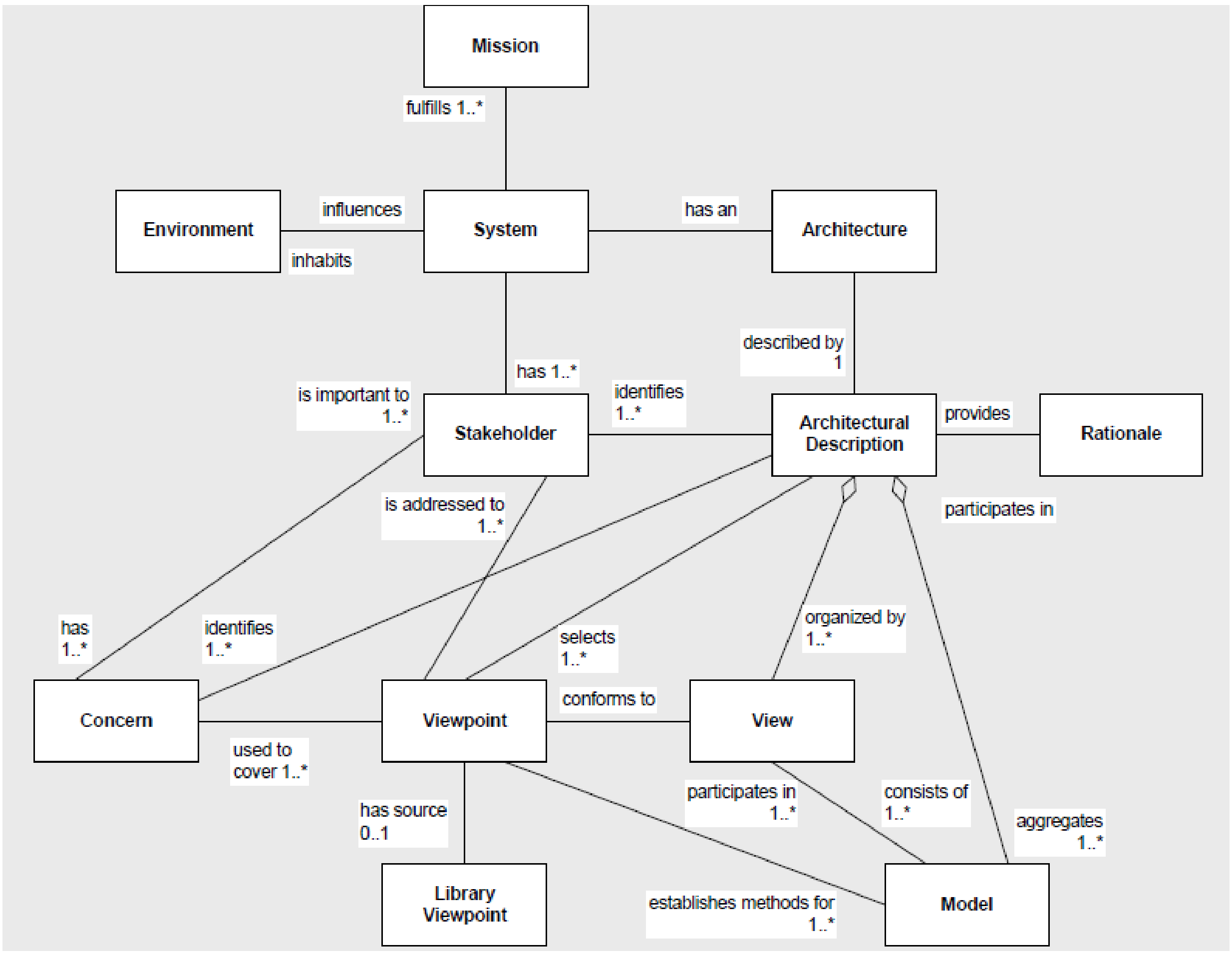


Figure 7. Knowledge Representation of Architectures – IEEE 1471 Conceptual Framework.

An example of an integrated architectural model is provided in Figure 8[13], which is an informational model supporting model driven SW-intensive system design methodology. In this example, the architecture data set comprises all the data populating the entities listed in the boxes, together with the relationships forming the connectors between the boxes.

A number of different architecture viewpoints can be expressed from the data set to draw out different perspectives of entities and relationships using different diagrammatic techniques. However the entire set of underlying entities and relationships is preserved, regardless of which subset of data is extracted for consideration in a particular viewpoint. Thus, a single integrated model or meta-model is created encompassing all required views.

[13] In Figure 7, the symbology '1..*' is used to indicate a 'many' relationship, i.e. the respective information items can be recursively decomposed.

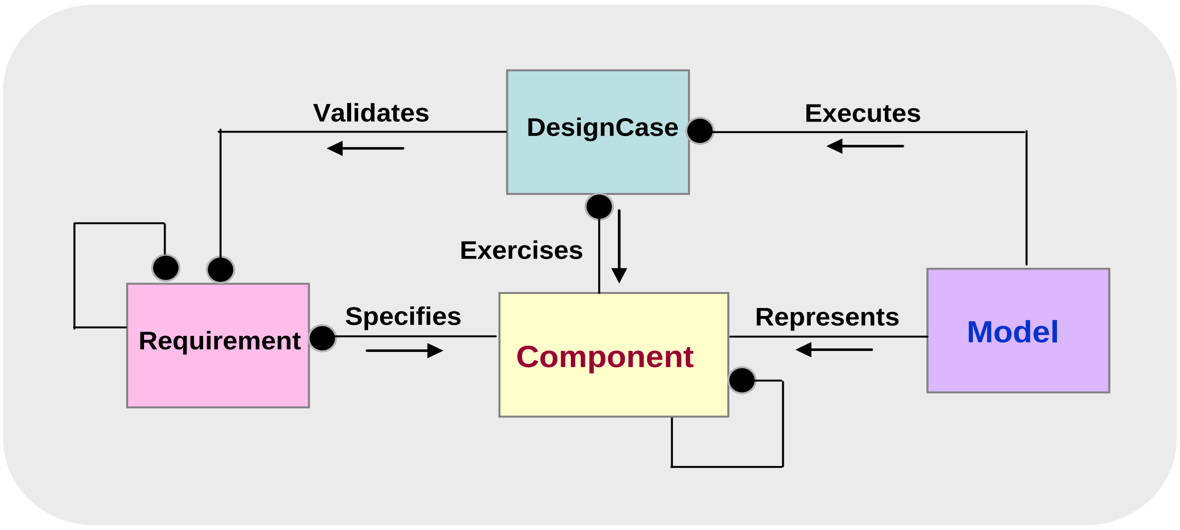


Figure 8. Informational Model for Model Driven System Design (MDSD) (Estefan 2008).

In Figure 8, the boxes show kinds of information; arrows show the direction of the relationship (not the direction of information flow); and the bullets show a 'many' relationship.

The diagram elements in Figure 8[14] can be unfolded as follows:

1. Requirements specify components (real world or conceptual).
2. Requirements are decomposed into other lower-tier requirements.
3. Components are decomposed into other lower-tier components.
4. Components represent design alternatives (e.g. design alternatives are proposed to potentially satisfy requirements).
5. Models are developed to represent components (i.e. design alternatives).
6. Models execute design alternatives using use cases to investigate the extent to which different design alternatives might satisfy the originating requirement as part of a trade study prior to selecting a particular alternative as the preferred design.
7. The results from exercising the selected component using use cases verify or not whether it actually satisfies the originating requirement.

The above example of the MDSD engineering methodology is significant in that it omits a number of key activities in SE; the most notable being the absence of an analysis or design activity, a purchase or build activity, and a test and evaluation activity. Why spend time and resources building a model then? In a SW context, this methodology might be useful, for example, if the modelling environment was capable of automatically generating the code forming the SW component solution. Thus building the model could be equated to building the SW. It can also be useful to analyse SW architectures to improve SW quality and correctness.

[14] In Figure 8, the boxes show kinds of information; lines represent relationships symbology '●' is used to indicate the presence of a one to many respective information items can be recursively decomposed.

Since the component hierarchy can be synthesised in the modelling environment, and the hierarchy will be preserved in the transition to a real-world SW instantiation, the modelling environment can also be used to evaluate the merits of alternative abstract SW structures more quickly and cheaply than developing detailed code implementations with alternative SW structures.

However, SW does not exist in isolation from its host environment in the real world, so an acquisition and integration activity is implied to result in a real-world solution that can be verified independently of the model. Although this method is not useful for synthesising hardware (HW) structures, it can be useful to record HW synthesis outcomes determined using more suitable alternate methods.

The methodology can also be useful for comparing commercial-off-the-shelf (COTS) product alternatives prior to selection and purchase, or evaluation of different system concepts prior to acquisition. This is the case in Defence where the above architecture-centric methodology is used widely to undertake analysis of architecture attributes supporting the capability development process.

However, in the Defence case, individual Defence acquisition projects have considerable freedom to create their own project specific architecture modelling approaches. The guidance provided does not include the notion of a formal information model prescribing the data elements, architectural attributes, or process. Thus, there can be considerable variation in semantics and vocabulary as well as methodology from project to project, and from one tool-user to another. This approach, while possibly useful from a project-specific point of view, does not provide extensibility beyond the boundaries of the project and the particular project system of interest to the broader Defence systems environment.

Since Government policy obliges a solution independent approach to the acquisition of military capability, the acquisition process relies considerably on the generation of project-specific specifications to provide sufficient guidance on function and performance to procure acceptable system solutions, abstracted from technology considerations and solution-space constraints. Without a notion of the solution implementation (i.e. the components to be acquired), it is not possible to provide feedback to verify the information model as represented in Figure 8. Thus, there can be no verification activity in an architectural context of the implemented components or architectural perspectives. The absence of verification and validation activity of the synthesised components and of the resultant system assembly is a major shortfall in the methodology illustrated from a systems engineering perspective. These considerations are revisited in this report in the context of Defence capability acquisition, and Defence enterprise architecture practice.

3.3.8 Reference Models

Another concept sometimes used in the systems modelling environment is the notion of a reference model[15]. In the SE, SW engineering and enterprise engineering disciplines, a reference model is an abstract framework or domain-specific ontology consisting of an interlinked set of clearly defined concepts produced by an authoritative source within a defined stakeholder community. A reference model can represent the piece-parts of any consistent idea, from business functions to system components, as long as it represents a complete set. This frame of reference can then be used to communicate ideas clearly among

[15] [online] URL: http://en.wikipedia.org/wiki/Reference_model

members of the same stakeholder community from the different perspectives provided in the reference model (Eeles & Cripps 2009).

The reference model is distinct from, but can also include related taxonomies of concepts, entities and relationships to reveal hierarchies of significance to inform stakeholders, including the system hierarchy or system architecture. Thus, as the system architecture is developed and documented during the system design process, it can provide guidance on preferred terminology, and standardised functions and components to use if relevant to the system problem under consideration.

In enterprise engineering, the boundary between engineering and business is quite blurred, with greater emphasis being placed on facets of enterprise-wide business concerns. In enterprise engineering, a business reference model (BRM) is typically included as part of an EAF, where the EAF is used to define a series of reference models to inform stakeholders how to organise the structure and views associated with an Enterprise Architecture (EA). Notions of enterprise also transcend notions of individual systems in the engineering sense; their boundaries can be quite indistinct, often determined by organising principles or other abstract criteria rather than necessarily having physically realised interface boundaries.

An example of a reference model associated with the commercial Zachman Framework for Enterprise Architecture (ZF) is provided in Figure 9 (Sowa et al. 1992). Notably in the ZF, engineering terminology is used in many instances using similar knowledge representation tools and techniques drawn from the engineering discipline, but tailored to provide an enterprise-specific business focus. For example, in the context of EA, a BRM is an important concept, where it provides a means to describe the business operations of an organisation independent of the organisational structure that performs them. It can also depict the relationships between business processes, business functions, and business areas to provide a foundation for analysis of service components, technology, data, and performance. Many of these have engineering underpinnings as evident in the various cell contents of the reference model in Figure 7[16].

A more detailed description of the ZF is provided in Appendix D. These considerations are also revisited in this report in the context of Defence enterprise architecture practice.

[16] [online] URL: http://www.zachman.com

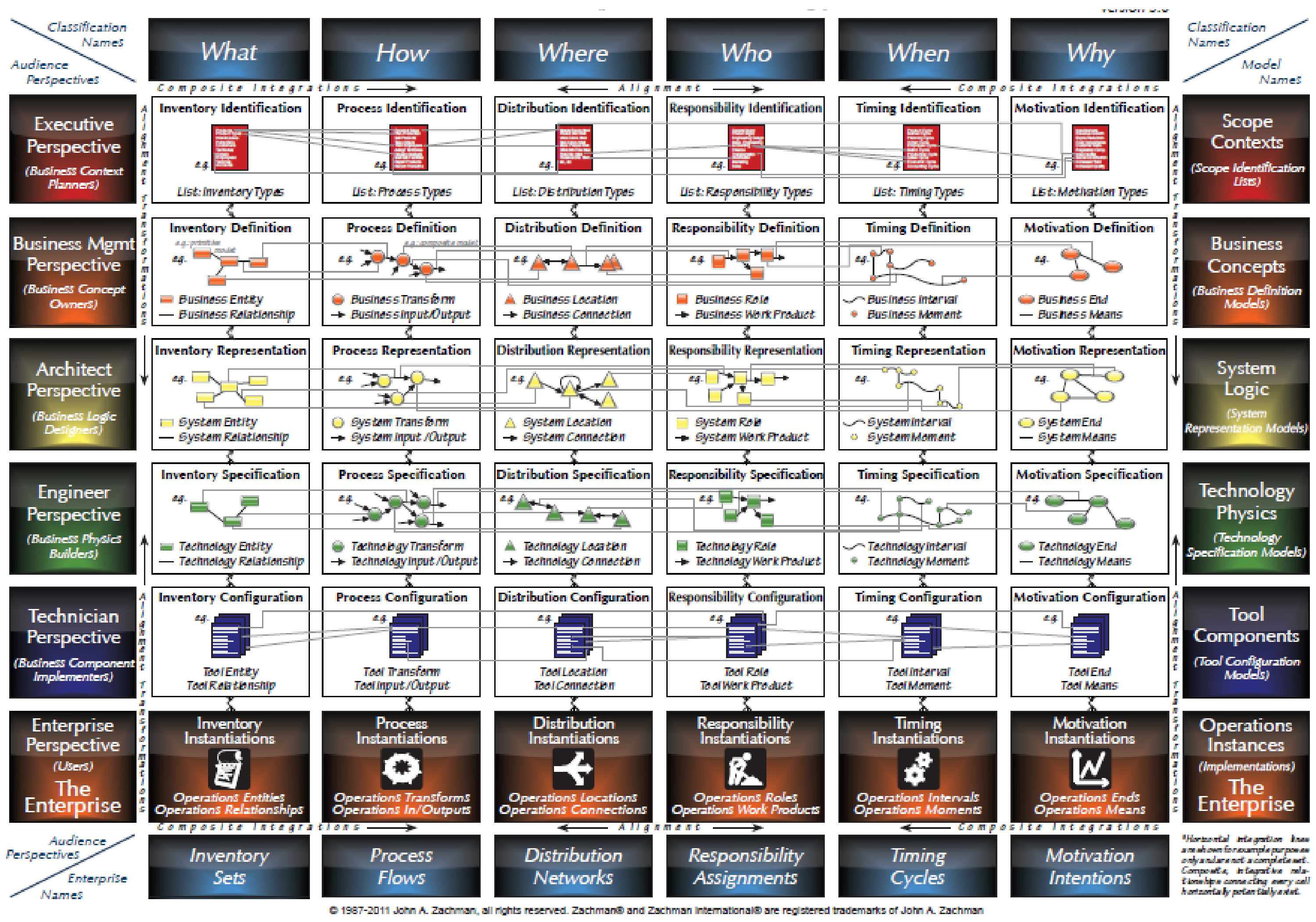


Figure 9. Example Reference Model Used in Zachman Framework for Enterprise Architecture v 3

3.3.9 Reference Architectures

Similarly, the term reference architecture is used both in a SW engineering and an enterprise engineering sense, and is typically a template solution for an architecture for a particular domain[17]. It is used to provide a common vocabulary with which to discuss prospective implementations, with a particular focus on commonality.

The reference architecture typically comprises:

- a list of functions;
- some indication of their application program interfaces (i.e. APIs);
- a description of the interactions between listed functions within the reference architecture; and
- a description of interactions with functions external to the reference architecture.

All of these can be captured, presented, analysed, and modified in a systems modelling environment.

A reference SW architecture typically provides a template to document those significant SW structures and respective elements and relationships for a particular project, domain or family of SW systems. This is often based on a generalisation of a set of solutions. These solutions may have been generalized and structured for the depiction of one or more SW architectures based on the harvesting of a set of patterns that have been observed in a number of successful implementations, together with guidance on how to to compose elements together to form a system solution (Eeles & Cripps 2009).

3.3.10 Design Patterns

3.3.10.1 Software Design Patterns

Another useful concept drawn from SW engineering which can be utilised in systems modelling is the notion of a design pattern[18] (Gamma et al. 1994), (Eeles & Cripps 2009). A design pattern is a recurring structure within a design domain.

A pattern typically expresses a specific problem or functional objective for a system along with a solution. The set of patterns sufficient to span the entire design within a domain is known as a pattern language. Using the Alexandrian method, patterns can be composed to synthesise solutions to diverse problems; the patterns that evoke the elements desired in the system become the building blocks for synthesising the solution. The patterns either suggest instructions for a solution structure (i.e. solution architecture) or contain solution fragments. The fragments and instructions are merged to form the system design (Maier & Rechtin 2002). A somewhat similar notion to a reference architecture, a design pattern aims to provide a generalised template solution for solving a specific type of problem that is reusable in different circumstances.

---

[17] [online] URL: http://en.wikipedia.org/wiki/Reference_architecture

[18] Christopher Alexander first conceptualised an approach to synthesis using formalised patterns in architecture in the field of civil architecture and urban design. The notion of design patterns was subsequently embraced by the field of SW engineering (Maier & Rechtin (2002).

However, design patterns typically manifest in terms of prescribed SW modules at the code level and interconnections internal to a SW element. This is usually on a micro-scale, rather than between SW elements or components within larger systems or between SW systems relating to system architectures.

Object-oriented[19] design patterns typically show relationships and interactions between classes or objects, without specifying the actual application specific classes or objects in the finalised design. A design pattern is therefore not a completed design that can be directly coded into a SW implementation, but seeks to articulate best practice distilled from previous successful implementations to guide new SW implementations. A design pattern must therefore be reprogrammed for each application, which differentiates itself from the concept of SW reuse for a new application, or using specific library modules or SW elements when building a new SW application.

A published design pattern is typically ascribed a specific identity or name, and includes prescribed information relating to:

- the identity, including intent for use (i.e. name);
- motivation or problem context in which the pattern can be used (i.e. the problem statement);
- applicability;
- pattern structure in terms of class diagrams and interaction diagrams;
- participants, comprising classes and objects used in the pattern, and their roles in the design;
- collaborations in terms of how the classes and objects used in the pattern interact with each other; consequences, providing a description of the results, side effects, and trade-offs caused by using the pattern; and finally,
- a description of the implementation of the pattern.

Sample code can also be included, as well as real-world examples where the pattern has been successfully used and codified. Design patterns can take different forms, including creational patterns, structural patterns, and behavioural patterns. Examples of SW design patterns are provided in Table 3 (Gamma et al. 1994).

Table 3. Examples of SW Design Patterns.

Name	**Description**
Creational Patterns	
Abstract Factory	Provides an interface for creating families of related or dependent objects without specifying their concrete classes.
Builder	Separates the construction of a complex object from its representation allowing the same construction process to create various representations.

[19] Object-oriented concepts and the object-oriented design paradigm are discussed in detail in Section 4 – Systems Approach to Problem Solving.

Singleton	Ensures that a class has only one instance, and provides a global point of access.
Structural Patterns	
Adapter (wrapper, translator)	Converts the interface of one class into another interface expected by clients, allowing classes to work together that would not be able to otherwise.
Bridge	Decouples an abstraction from its implementation allowing the two to vary independently.
Facade	Provides a unified interface to a set of interfaces in a subsystem
Behavioural Patterns	
Iterator	Provides a way to access the elements of an aggregate object sequentially without exposing the underlying representation.
Mediator	Defines an object that encapsulates how a set of objects interact. This promotes loose coupling by keeping objects from referring to each other explicitly, and allows the interaction to be varied independently.
Observer (publish/subscribe)	Defines a one-to-many dependency between objects where a state change in one object results in all of its dependents being consequently notified and updated.

Patterns also allow SW developers to communicate using well-known names for SW interactions. As common design patterns are evolved and improved over time, they can become more robust when applied under different circumstances compared with ad-hoc bespoke designs. By providing tested and proven SW development paradigms, design patterns can reduce both the development effort required and the development risk, reducing the likelihood of occurrence of subtle problems that might otherwise cause major problems[20].

3.3.10.2 SoS Design Patterns

Over the last decade, the notion of design patterns has been adapted for application in an entirely different context - for use in shaping net-readiness[21] in large scale SoS and socio-technical system solutions. The organisation NCOIC, for example, has coined the term "net-centric patterns", where they have applied the notion of patterns to assist in solving shared interoperability problems by soliciting government and industry-wide consensus on the approach [22] (Bowler 2010).

NCOIC is a consortium comprising government and industry representatives from several

[20] [online] URL: http://en.wikipedia.org/wiki/Design_Patterns; Gamma et al., 1994.

[21] Net-readiness is described by NCOIC in terms of a system's ability to connect to a common communication network together with other net-ready systems to form a SoS.

[22] [online] URL: https://www.ncoic.org/technology/deliverables/patterns/

nations including the US, UK, and Australia[23], to facilitate private and public sectors working together between cross-domains towards achieving interoperability[24] goals. Here the solution may not necessarily be SW-based, but some of the same principles associated with design patterns have been emulated to promote interoperability and interface compatibility on a much larger scale.

The NCOIC Interoperability Framework (NIF) was developed to provide a vehicle to distil information that is considered relevant to net-centricity, and to recommend particular standards for international adoption to support improved net-centricity, together with flexible guidance to promote multiple use[25] (NCOIC 2008). The impetus stemmed from the difficulty encountered in trying to achieve harmonisation of technical standards and processes between interconnected systems and SoS, and across multiple organisations, each of which is evolving independently at different rates, with diverse needs, drivers and constraints.

Net-centric design patterns have been developed over three domains as follows:

- Operational – comprising standard practices and their interoperability requirements needed to conduct activities (military operations or business objectives) in a given mission context;
- Capability – comprising standard methodologies and functions needed to support required activities in a given mission context; and
- Technical – comprising technical standards, technologies and interoperability techniques needed to support required capabilities in a functional context specified in the associated capability patterns (NCOIC 2008)[26].

3.3.11 Model Reuse

In a similar vein to design patterns, another useful SW engineering concept used in systems modelling is the notion of model reuse. Model reuse is simply copying the implementation of some parts of a model or all of the model and reusing it in a different model implementation[27].

Model reusability is the ability to reuse segments of the model to add new functionality with minimal modification; the impetus being to reduce redundant effort, and hence time and cost to develop, verify, and validate new models. The ability to reuse segments of the model relies on the ability to identify commonalities between different segments such that larger models can be built by combining the smaller segments (Frakes et al. 2005).

[23] Member organisations include Object Management Group (OMG), The Open Group, Thales, Australian Department of Defence, Federal Aviation Authority (FAA), IBM, Boeing Ltd., Raytheon, Lockheed Martin CISCO, Saab, MITRE, and EADS.
[24] NCOIC has published an interoperability reference model whose scope spans people, process, applications, information services and network transport considerations.
[25] [online] URL: https://www.ncoic.org/technology/deliverables/nif/

[26] This is akin to the notion of open architectures, although the mechanism to select the standards and the motivations of the participating organisations can differ, but both approaches seek similar outcomes.
[27] [online] URL: http://en.wikipedia.org/wiki/Code_reuse

Reusability implies explicit management of numerous aspects during model development, including:

- modelling language and application compatibility,
- documentation,
- separate verification and validation (V&V),
- packaging,
- distribution,
- installation,
- configuration,
- maintenance, and
- upgrade.

These issues might not otherwise have been given attention if reusability was not considered. Since the life cycle of the portion being reused, or the life cycle of a library implementation may differ from that of the model being developed, the ability to maintain reused code is an important consideration when weighing the perceived benefits of reusing code against the total life cycle cost of supporting the entire model.

Models, or model segments, can be reused by a modeller on an a-hoc basis at a later date to leverage previous effort. An obvious example of model reuse is the refinement of the implementation from one version to the next:

- to fix implementation problems within the model;
- to provide enhanced model features; or
- to provide a known starting point for development of a different application.

A more deliberate approach to model reusability may also be taken where internal abstractions are used to create specific model segments or modules (or objects in the case of object-oriented implementation) for later reuse, and are explicitly copied for separate storage in a library. These library implementations are particularly suited for performing common operations on data that may be used repeatedly in many different models. Library implementations therefore need a defined interface, and documented features, attributes and testing so that newly developed code can readily access the required functionality within the library module with a known degree of confidence.

Model segments can be imbued with certain characteristics and attributed to particular libraries to facilitate easier sharing and reuse. Characteristics that facilitate model segment reuse include:

- modularity,
- loose coupling,
- high cohesion,
- information hiding, and
- separation of concerns.

Similarly, common data can be located in libraries for use by independent models. Library implementations can also be sourced from third parties for use in new model development.

However, the cost-benefit of using tested library implementations must be weighed against inherent limitations including an inability to tune the library implementation to optimise features and attributes, including interface details and performance, and any additional time or cost incurred to acquire, learn, configure and support the library to suit the required application.

3.4 Programming Language Concepts

3.4.1 Imperative Programming Language Paradigm

Specific to SW modelling, two fundamentally different paradigms predominate, offering either an algorithmic perspective or an object-oriented perspective of the system.

The algorithmic perspective originated in the mid-1950s. It was quickly embraced by the engineering community where algorithms could be described using one of a family of imperative computer programming languages; the main building block of SW being the procedure or function. In computing science, imperative programing is a programming paradigm that describes computation in terms of statements that change a program's state[28].

This particular approach supports the notion of structured programming and the decomposition of larger algorithms into smaller ones; particularly suited for numeric computation, mathematical modelling, and quantitative analysis.

These techniques are useful to analyse and solve scientific problems typically found in engineering. The resultant SW architecture is therefore a reflection of the SW partitioning of the algorithm into its smaller parts. Examples of imperative (procedural) programming languages include Fortran, ALGOL, COBOL, PASCAL, and the C programming language.

3.4.2 Object-Oriented Programming Language Paradigm

The basis of object-oriented development methods is the "object". An object is a fundamental concept in the object-oriented SW modelling paradigm, where it is a "thing", generally drawn from the vocabulary of a problem space or solution space. A class is also a fundamental concept, which is a description of a set of objects that share the same attributes, operations, relationships and semantics. Using object-oriented vernacular, the object is known as an "instance" of the class.

Of particular significance, each object has:

- a specific identity (i.e. it can be distinguished from other objects);
- a state (i.e. it usually has data associated with it); and
- behaviour (i.e. the object can interact with other objects, and it can interact with external influences).

[28] [online] URL: http://en.wikipedia.org/wiki/Imperative_programming

Other important concepts defined in the UML object-oriented paradigm include:

- an element, which is defined as an atomic constituent of a model;
- a component, which is defined as a physical and realisable part of a system that conforms to and provides the realisation of a set of interfaces;
- an interface is a collection of operations which affect behaviour that are used to specify a service of a class or a component[29], and
- a node is a physical element that exists at run-time and that represents a computational resource, generally having at least some memory, and, often times, processing capability.

In UML, "use cases" are used to describe the behaviour of the system as seen by its end users, analysts and testers. A "use case" is comprised of a number of discrete scenarios, each of which provides a specific sequence of actions, including variants, that yields an observable result that illustrates system behaviour to the actor. Here the actor is a coherent set of roles that users of use cases play when interacting with the use cases. A set of use cases is used to verify and validate the system's architecture.

The resultant SW architecture is revealed in terms of:

- the set of significant decisions about the organisation of the SW system,
- the selection of the structural elements and their interfaces from which the system is composed, together with their behaviour as specified in the collaborations among those elements,
- the composition of the structural and behavioural elements into progressively large subsystems, and
- the architectural style that guides the organisation of the elements and subsystems.

Object-oriented development thus provides the conceptual foundation for assembling systems out of SW components that are standardised technology building blocks (Booch et al. 1999). This approach is therefore particularly suited for SW-intensive systems largely comprised of COTS components. The object-orientation maps well into the physical realm making it particularly suited for representing and analysing physical architectures and their interfaces, and providing a robust audit trail for the recursive functional to physical allocation synthesis activity. It is also much easier to model large numbers of asynchronous interactions between many interacting entities.

UML has continued to evolve over the ensuing years, leading to development of notions of EA modelling and MBSE[30] - ***it is thus prerequisite to be familiar with the basic ideas underpinning object-oriented SW and systems modelling and the accompanying vernacular and standards, to understand notions of EA modelling and MBSE***.

[29] A useful definition of interface from a SW engineering perspective is provided in (Sparx Systems 2007a). The UML2 Tutorial using the tool enterprise Architect defines an interface as a specification of behaviour that implementers agree to meet. It is therefore a contractual obligation. By realising an interface, classes are required to guarantee they support a required behaviour, which allows the system to treat non-related elements in the same way, through the common interface.

[30] UML version 2.4.1 was formally published in April 2012 in two parts, as ISO/IEC standards ISO/IEC 19501-1:2012(E) and ISO/IEC 19501-2:2012(E).

3.5 Modelling Language Concepts

"Computers do not solve problems, they execute solutions" – Laurent Gasser, 1995.

3.5.1 Language Concepts

Another central concept in systems modelling is the notion of a modelling language. A modelling language is any artificial language that can be used to express information in a structured manner that is defined by a consistent set of rules; the rules providing the basis for interpreting the meaning of the information in the structure[31].

From Table 2, it is evident there are many approaches to conceptual modelling of systems. Aside from the specifics of each modelling process to construct a model, each approach can be differentiated according to the form of representation of the information that comprises the model, i.e. the modelling language used.

Various modelling languages are used in many different disciplines, including computer science, operations research, business and operations management, SW engineering, SE, and enterprise engineering. These modelling languages provide the ability to specify or describe system requirements, structures and behaviours with the required fidelity in such a way that stakeholders (e.g. customers, operators, analysts, designers) can better understand the system being modelled. These can vary in quality of knowledge representations from simple informal pictorial representations using commodity drawing tools such as PowerPoint™ or Visio™ to produce diagrammatic knowledge representations, to precise, executable languages using specialised SW tools which can support automated system V&V, simulation, and code generation from the same representations.

These modelling languages can be either graphical or textual. A textual or imperative modelling language typically uses standardised keywords accompanied by parameters to construct computer-interpretable expressions. A simple example of some object-oriented Java code for the class MessageParser is provided in Figure 10 (Booch et al. 1999, p 338). The concept of object-orientation is described in more detail in Section 3.4.2 below.

Graphical modelling languages use diagramming techniques with named symbols that represents concepts, together with lines that connect the symbols representing relationships, and other graphical notations to indicate constraints and other relevant notions as shown in Figure 11.

[31] [online] URL: http://en.wikipedia.org/wiki/Modeling_language

```
class  MessageParser  {
public  Boolean  put  (char c)  {
      switch  (state)  {
                  case  Waiting::
                        if  (c ==   '<' ) {
                                         state  =  GettingToken;
                                         token  =  new StringBuffer ( ) ;
                                         body  =  new StringBuffer ( ) ;
                        }
                        break;
                  case  GettingToken :
                        if  (c == '>' )
                                         state  =  GettingBody;
                        else
                                         token.append (c) ;
                        break ;
                  case GettingBody :
                        if (c == ';')  {
                                         state = Waiting;
                                         return true;  }
                        else
                                         body.append (c) ;
                  }
                  return false;
      }
public stringbuffer getToken ( )  {
                  return  token;
public stringbuffer  getBody ( )  {
                  return  body;
      }
private  final  static  int  Waiting = 0;
private  final  static  int  GettingToken = 1;
private  final  static  int  GettingBody = 2;
private  int  state = Waiting;
private  StringBuffer token, body;
}
```

Figure 10. Textual Representation of State Machine Providing Specific Software Functionality in Java Code Created Using Tool Automated Code Generation Capability (Booch et al. 1999).

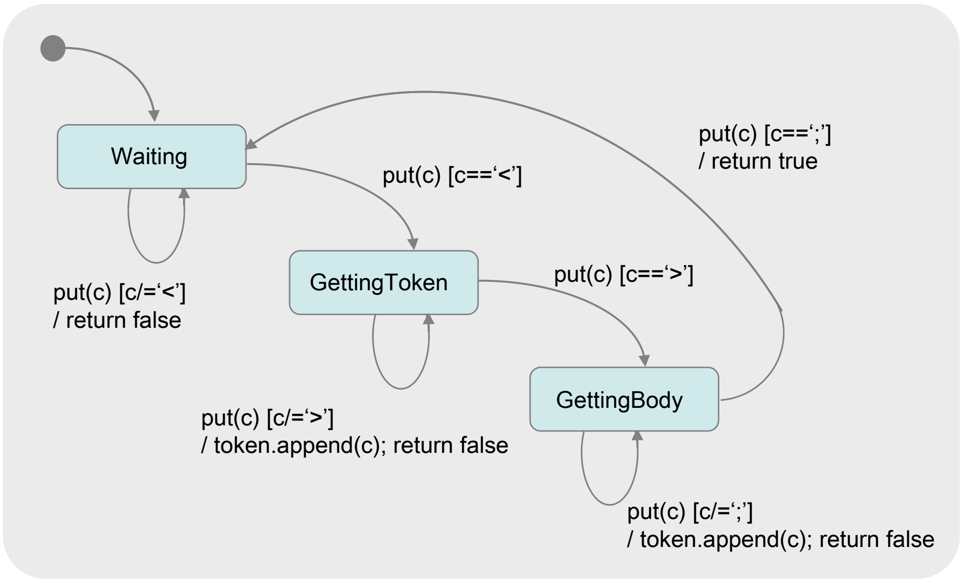


Figure 11. Tool Generated Graphical Representation of a State Machine Providing the Specific Software Functionality of Figure 8 (Booch et al. 1999).

Graphical modelling languages are based on the notion that knowledge can be described in terms of entities, relationships, interpretations and structure. Entities, relationships and interpretations are formally defined in the mathematical sciences in predicate calculus. Structure is formalised in logic. Graphs provide a visual means of describing entities and relationships (Dickerson & Mavris 2010).

Examples of graphical modelling languages include:

- behaviour trees,
- business process modelling notation (BPMN™),
- flowcharts,
- IDEFx™ family of diagrams, and
- Architecture Description Languages (ADL)[32].

In the example shown in Figure 10, a machine is implemented using Java code which parses different messages when certain conditions are met. Code is generated automatically from a simple state diagram using a tool where the graphical representation of the state machine is shown in Figure 11 (Booch et al. 1999, p 338).

Figures 10 and 11 show an example of modelling a reactive (i.e. event driven) object, useful particularly for instances of classes, use cases, and modelling the system as a whole.

When modelling the behaviour of a reactive object, it is necessary to specify three things:

- the stable states in which the object may live;
- the events that trigger a transition from state to state; and
- the actions that occur on each state change.

It also involves modelling the lifetime of the reactive object, starting at the time of the object's creation, and continuing until the object's destruction, highlighting the stable states in between in which the object may be found.

In graphical terms, an interaction diagram models the behaviour of a society of objects working together, whereas the statechart diagram models the behaviour of a single object over its lifetime. The activity diagram models the flow of control from activity to activity, whereas the statechart diagram models the flow of control from event to event.

Textual and graphical modelling languages are used widely in both the commercial and the Defence sectors to assist in developing engineering system solutions. The choice of visual presentation aesthetics and techniques in diagramming is particularly important in terms of determining how to create the most effective graphical knowledge representations of the system at hand.

Unlike computer drawn abstract pictorial representations, for example, using Microsoft PowerPoint™ drawing tool, where an artist can enjoy considerable discretion in creative presentation, model diagraming of the ilk of BPMN, UML and SysML[33] is heavily rules-

[32] Architecture Description Languages are also used for enterprise architecture modelling, and may be vendor tool specific such as for the Vitech CORE SE Tool, or be an industry standard such as Archimate®, managed under the auspices of Object Management Group (OMG).

[33] BPMN, UML and SysML are graphical modelling languages managed under the auspices of OMG (BPMN 2011), (UML 2011a,) (UML 2011b), (SysML 2006).

oriented. This allows the modeller to maximise the clarity of the information contained in the model so the modeller can accurately project the desired understanding to the intended audience.

Concerns such as line and contour scale and proportion, colour and thickness, and composition and layout, and number and complexity of diagram elements, all become important considerations in system, software, and process modelling. This may assist or detract from understanding. This is especially important when specific meaning is attributed to particular variations in visual presentation (Lieberman 2004).

Two examples of graphical knowledge representation, using BPMN notion, and UML 2.0, are provided in Figures 12 and 13 respectively (White 2004). The same information is provided in both diagrams in each figure, however, the way it is presented, and the method of interpretation, are dependent on the knowledge representation technique and associated rules set.

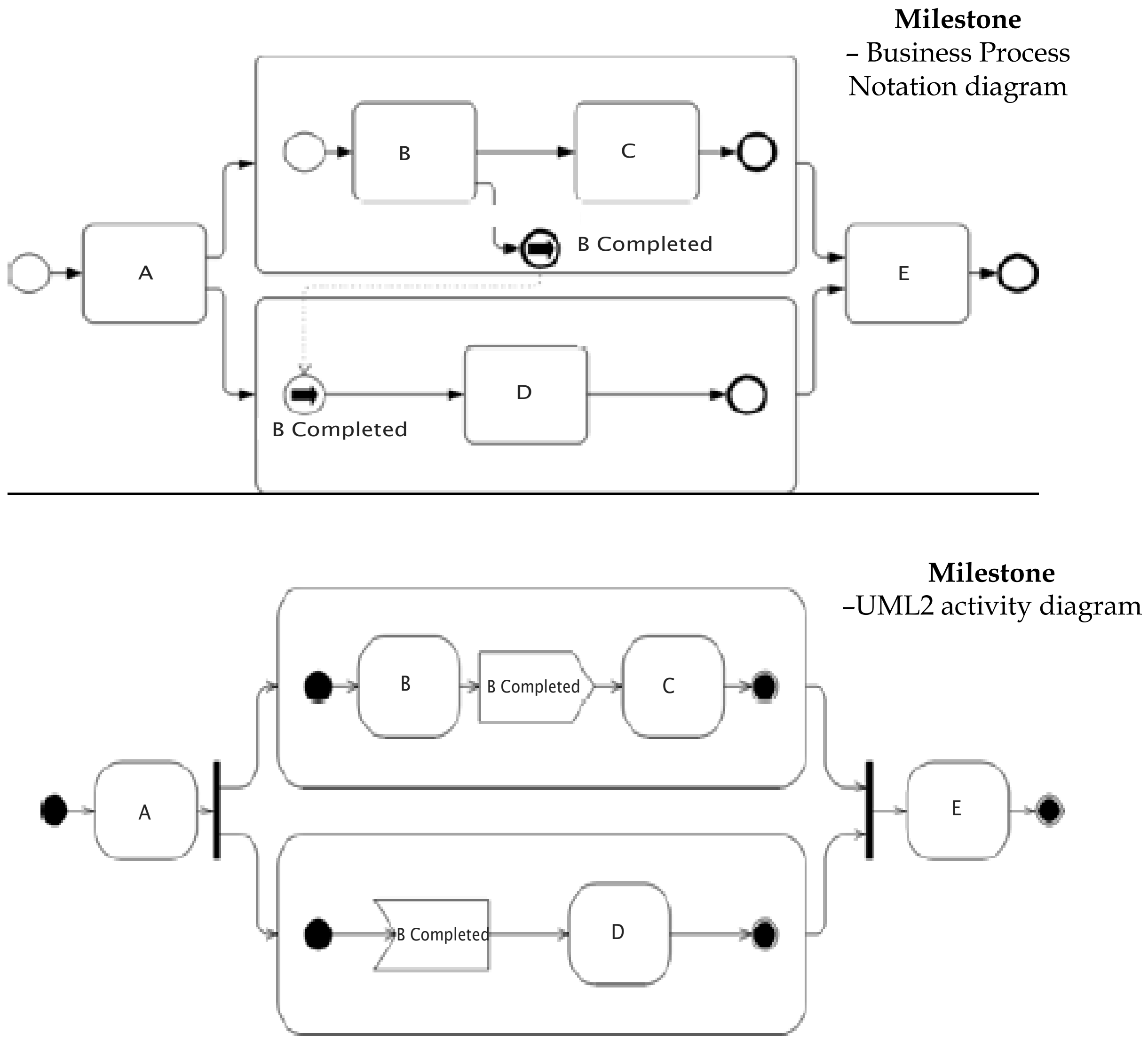
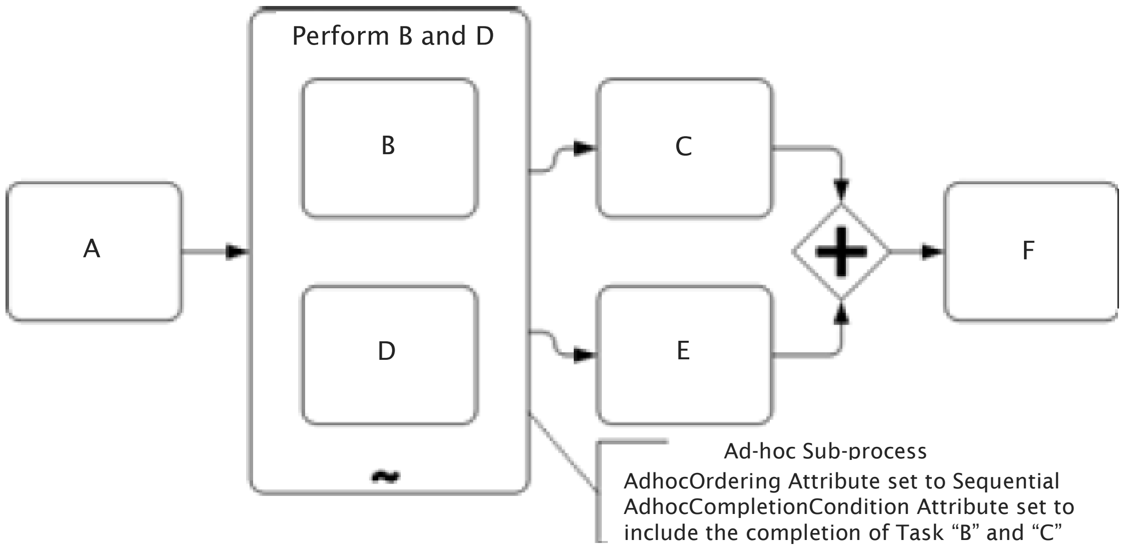


Figure 12. Example Milestone Graphical Knowledge Representation using BPMN & UML2 (White, 2005).

Interleaved Parallel Routing

– Business Process Notation diagram


Perform B and D
B
D
~
A
C
E
F
Ad-hoc Sub-process
AdhocOrdering Attribute set to Sequential
AdhocCompletionCondition Attribute set to
include the completion of Task "B" and "C"

Interleaved Parallel Routing

– UML2 Activity diagram

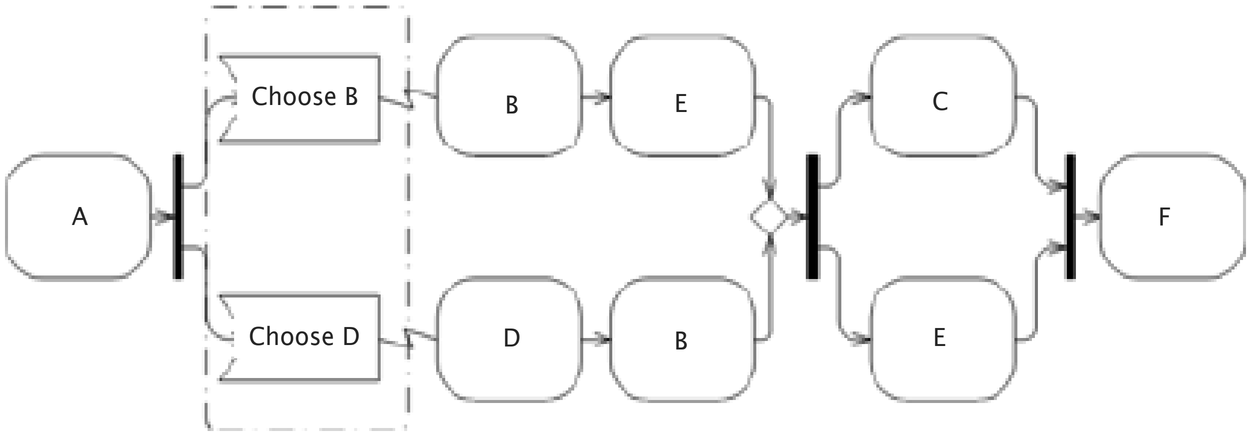


Figure 13. Example Workflow Graphical Knowledge Representation using BPMN and UML2 (White, 2004).

3.5.2 Architecture Description Languages

An international engineering standard, ISO/IEC/IEEE 42010:2011 Systems and Software Engineering – Architecture Description, defines an ADL as any form of expression for use in architecture descriptions. The standard also specifies the minimum requirements to create an ADL. The use of an ADL is inherent in the notion of SW and SE-related architecture modelling so as to be able to articulate their respective architectures in a prescribed manner.

In the SW engineering discipline, an ADL is a computer language used to describe and represent SW architectures in an integrated form, i.e. the structure and behaviour of a SW system and the non-SW entities that the system interfaces to. Thus, the SW system is represented as a set of SW components, their connections, and their significant behavioural interactions.

In SE, an ADL can be a language and/or a conceptual model used to represent the system architecture in an integrated form, in terms of its structure, layout, behaviour and other system-specific views associated with systems analysis and synthesis.

For enterprise engineering, various approaches may be taken, depending on the particular EAF. Enterprises can be modelled, however for the most part, they use commonly available textual and graphical tools using recognised standards rather than having an EAF specific ADL to promulgate EAF based information.

EAF utilise various modelling techniques including Business Process Modelling Notation (BPMN®), Archimate and UML. They can also use office and graphic drawing tools such as Microsoft Office PowerPoint™ or Visio™, or even use SW or SE ADL based tool sets to draw pictorial representations of enterprise-related information.

For example, the EAF published by The Open Group, TOGAF, focuses on a particular architecture development method, ADM, depicted in Figure 14[34].

[34] [online] URL: http://www.opengroup.org/subjectareas/enterprise/togaf

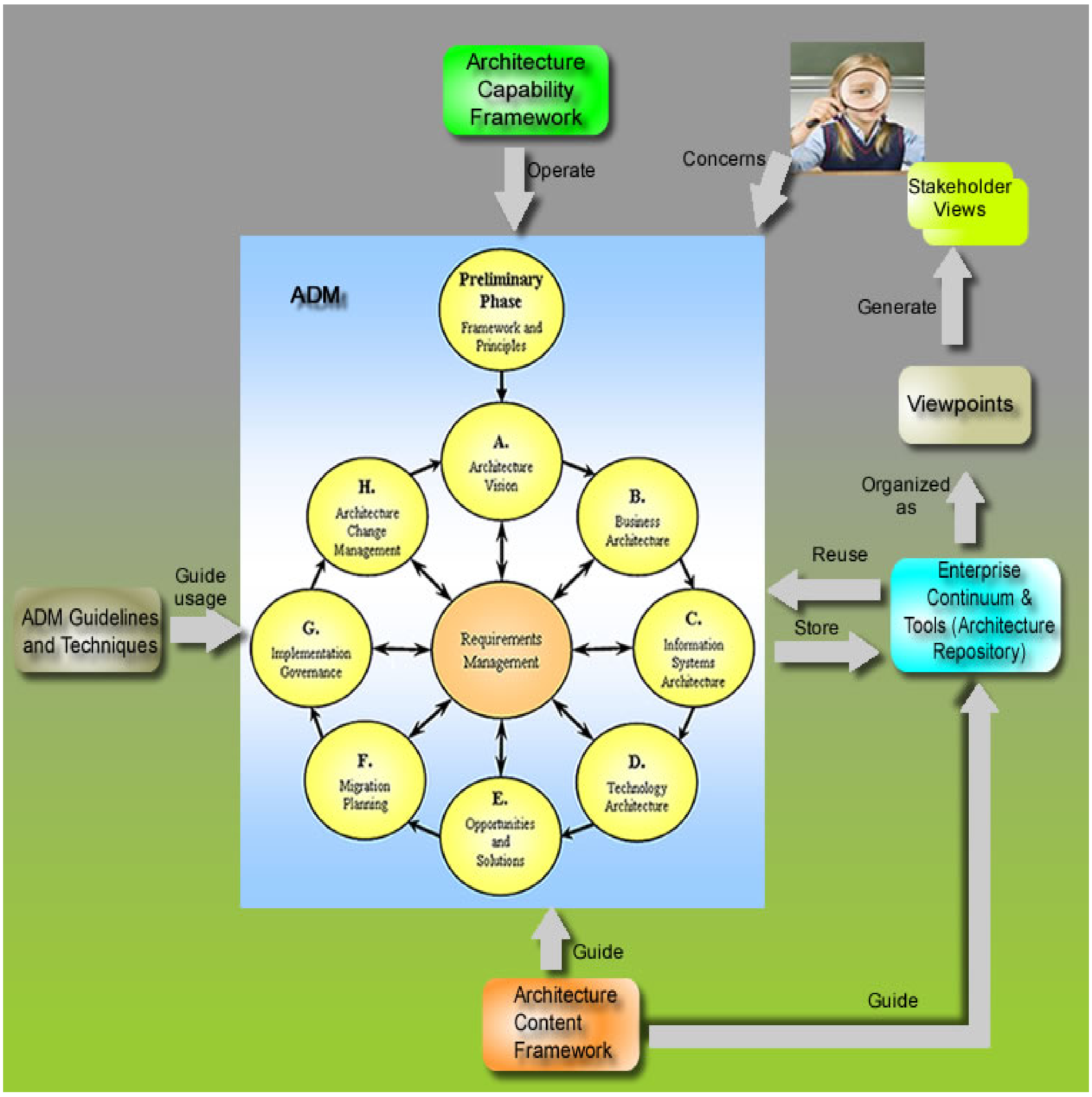

Architecture Capability Framework
Operate
Concerns
Stakeholder Views
Generate
Viewpoints
Organized as
Reuse
Store
Enterprise Continuum & Tools (Architecture Repository)
ADM
Preliminary Phase
Framework and Principles
A. Architecture Vision
B. Business Architecture
C. Information Systems Architecture
D. Technology Architecture
E. Opportunities and Solutions
F. Migration Planning
G. Implementation Governance
H. Architecture Change Management
Requirements Management
ADM Guidelines and Techniques
Guide usage
Guide
Architecture Content Framework
Guide

Figure 14. TOGAF Architecture Development Method Process Overview.

The TOGAF does not prescribe any particular suite of products to build, nor represent an EA model (i.e. architecture descriptions or views), nor direct information content. Instead, TOGAF provides two reference models, the TOGAF Technical Reference Model, and the Integrated Information Infrastructure Model as depicted in Figure 15 (Josey 2009).

The Open Group suggest that enterprise-related information can be depicted by populating templates replicated from military EA frameworks such as the MODAF developed by MOD in the UK and DoDAF developed by DoD in the US[35] (Dandashi et al. 2006).

[35] The MODAF and DoDAF are described in more detail in Section 9 Enterprise Architecture Concepts.

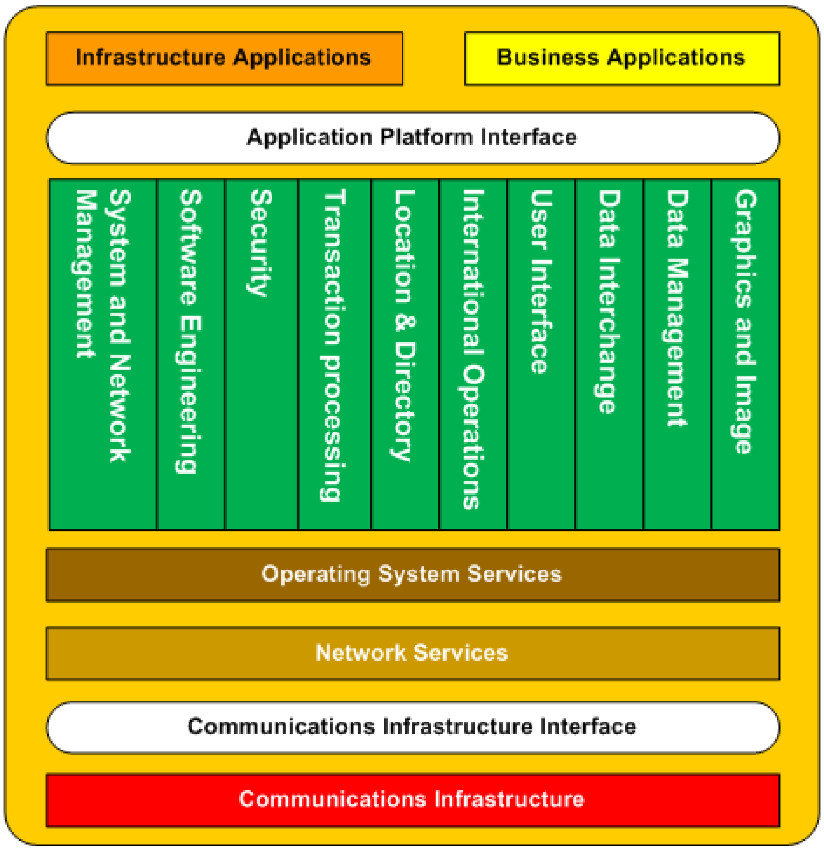


Figure 15a. TOGAF Technical Reference Model.

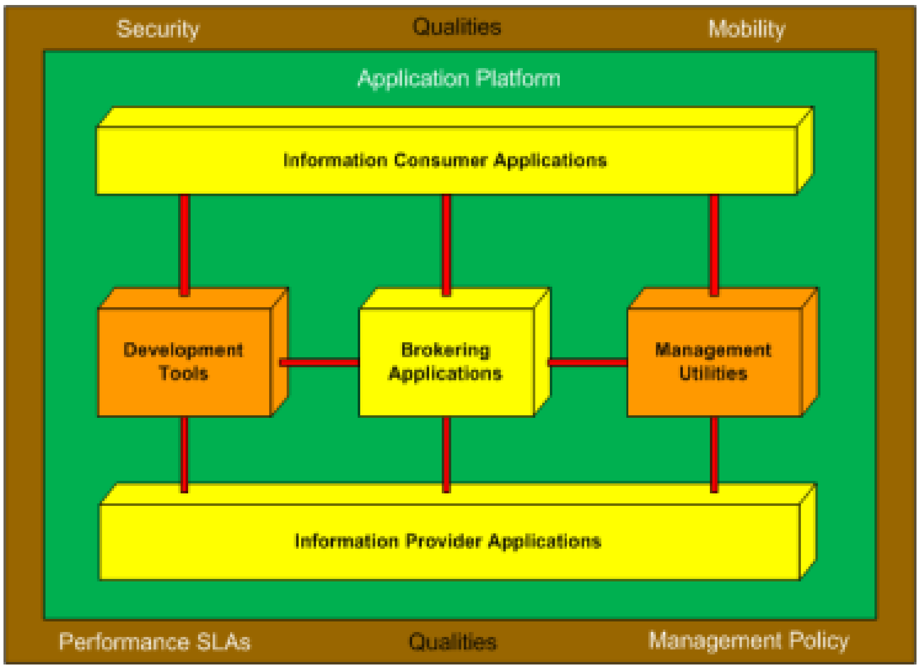


Figure 15b. TOGAF Integrated Information Infrastructure Model.

The EA model is then the aggregation of the populated templates that describe the enterprise from various perspectives as prescribed by the reference model. This information is stored in an architecture repository, typically in artefact form, for later retrieval.

Since the TOGAF is agnostic to the SE and SW engineering disciplines, definitions can differ, despite frequent use of well-known terms from these disciplines. Some effort has been made to align the TOGAF approach to produce DoDAF products, however, this is strictly view-centric, and is agnostic to any underlying concurrent systems engineering, quality management, project management or other supporting processes in train to produce the information for inclusion in the respective views.

The military EAFs, MODAF and DoDAF, in particular, have developed their own ADLs, each of which can be implemented as profiles of internationally recognised modelling language standards. They each define numerous architecture views (similar but different), that can be generated using commercial EA and MBSE tools supporting internationally recognised standard ADLs such as UML[36].

Importantly, the DoDAF and MODAF are also underpinned by information meta-models. These are integrated data models, which define the set of underlying architectural information (entities and relationships), and are stored in the tool or repository.

The different view templates can thus be populated from data stored according to the single integrated data model representing the associated systems or SoS architectures associated with the respective problem domains. The MODAF Meta-Model (M3) is the information model for MODAF (MODAF 2010). This defines the structure of the underlying architectural information that is presented in the MODAF views. Similarly, the DoDAF Meta-Model (DM2) is the information model for the DoDAF v2, which defines the structure of the underlying architectural information that is presented in the DoDAF views (DoDAF 2009).

An example is shown in Figure 16 of a simplified high-level integrated data model showing operational and system level data elements and relationships associated with the first military EAF, the C4ISR Architecture Framework (C4ISR AF) (C4ISRAF 1997). The underlying architectural information is stored in a database in the tool or in a managed data repository.

The C4ISR AF v 2.0, was a precursor to the DoDAF v 1.0; its underlying core architecture data model (CADM) specifying its ADL was several hundred pages in length. While this representation could be implemented using a number of different approaches, it is particularly well-matched to the object-oriented language constructs in UML. A corresponding ADL spanning the CADM would thus be implemented as profile of UML in a UML-based tool.

An architecture process overview to generate the respective C4ISR AF artefacts[37] using a computer-aided SE tool is provided in Figures 17 and 18, where the CADM (i.e the

[36] The Vitech CORE tool is one exception where the DoDAF and MODAF profiles are overlaid on a propriety ADL. This tool used structured analysis principles rather the object-oriented concepts inherent in UML (Long 2010), (Long & Scott 2011).

[37] The C4ISR AF, and its successor the DoDAF, define numerous document artefacts including common or all views (e.g. AV1 operational views (e.g. OV1 to OV6), systems views (e.g. SV1 to SV-11) and technical views (e.g. TV1-TV3). Later versions of the DoDAF have increased the number and types of views supported. See DoDAF Desk book Volumes 1, 2, and 3 for further information on the DM2 (formerly CADM) and the DoDAF specific artefacts (DoDAF 2009).

integrated data model or meta-model) is overlaid on the tool native ADL (Levis 2000).

However, there are some crucial differences between the commercial and military approaches. Each commercial EA Framework is typically based around a particular reference model and/or reference architecture. They are agnostic to the notions of systems or SoS architecture that can be represented in an integrated form, underpinned by an ADL and information meta-model as described above for MODAF and DoDAF.

EAFs are also agnostic to engineering notions of analysis and design, whereas MODAF and DoDAF are integral process overlays on the engineering and acquisition processes in the UK MOD and the US DoD respectively (Ryder & Flannigan 2005).

In commercial EAFs, information is commonly expressed in natural language form. This information is typically organised into categories according to the reference model and displayed in non-architectural form to reflect a particular viewpoint or stakeholder perspective.

This is akin to the weather map example described above, where prescribed information is derived separate to the model, and displayed a prescribed manner; the information populating a standardised set of templates, possibly drawn from a designated solution architecture or prescribed design patterns. Enterprise-related information may be inter-related within a particular category, for example, when describing a particular business process, but many facets of the EA cannot be represented using an ADL or produced as an analytical outcome.

The particular military EA frameworks mentioned above, i.e. MODAF and DoDAF support the notion of different viewpoints or perspectives based on a single integrated system architecture model, thus preserving the integrity of the systems architectures. The MODAF and DoDAF employ specific but distinct tailored interpretations of the internationally recognised standard modelling languages UML and SysML managed under the auspices of OMG, and thus inherently presume an integrated architectural approach.

Developed in collaboration with Defence industry, UK MOD, US DoD, OMG has recently released a new ADL, Unified Profile for MODAF and DoDAF, known as UPDM[38] (UPDM 2012). UPDM incorporates modelling features of both these military EAFs to improve interoperability between commercial tools and data sets, and to expand the types of analyses and presentation formats supported (Hause, 2010), (Hause et al. 2012), (IBM UPDM 2012).

Further elaboration is provided in Section 9 Enterprise Architecture Concepts and Section 10 Defence Enterprise Architecture Concepts of this report in the context of enterprise architecture practice in Defence.

[38] [online] URL: http://www.omg.org/spec/UPDM/2.0/

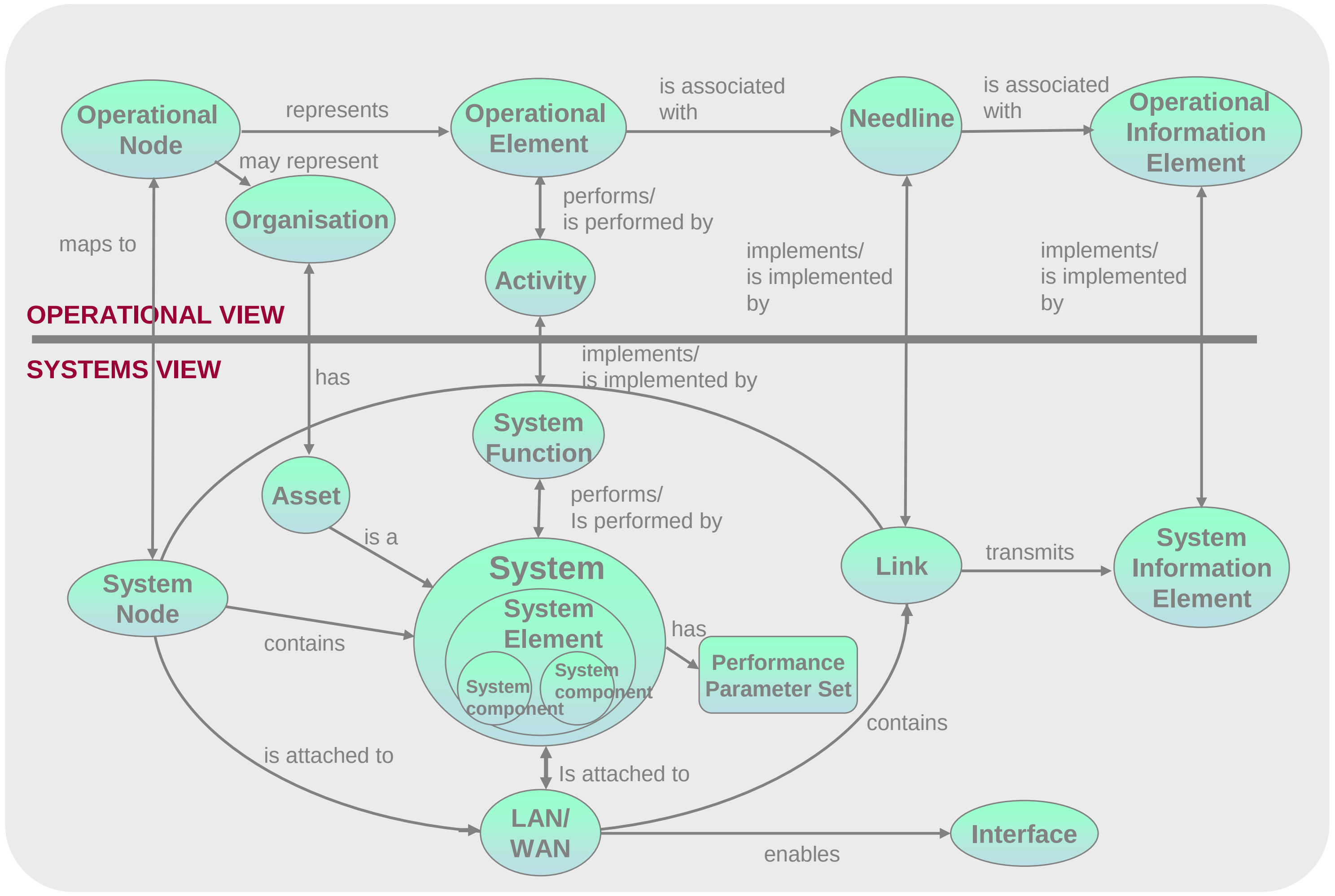
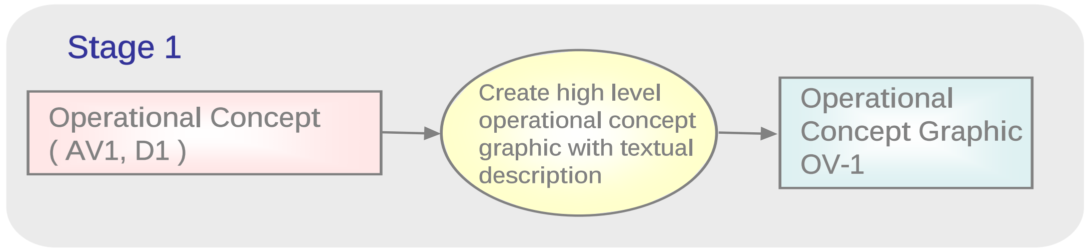
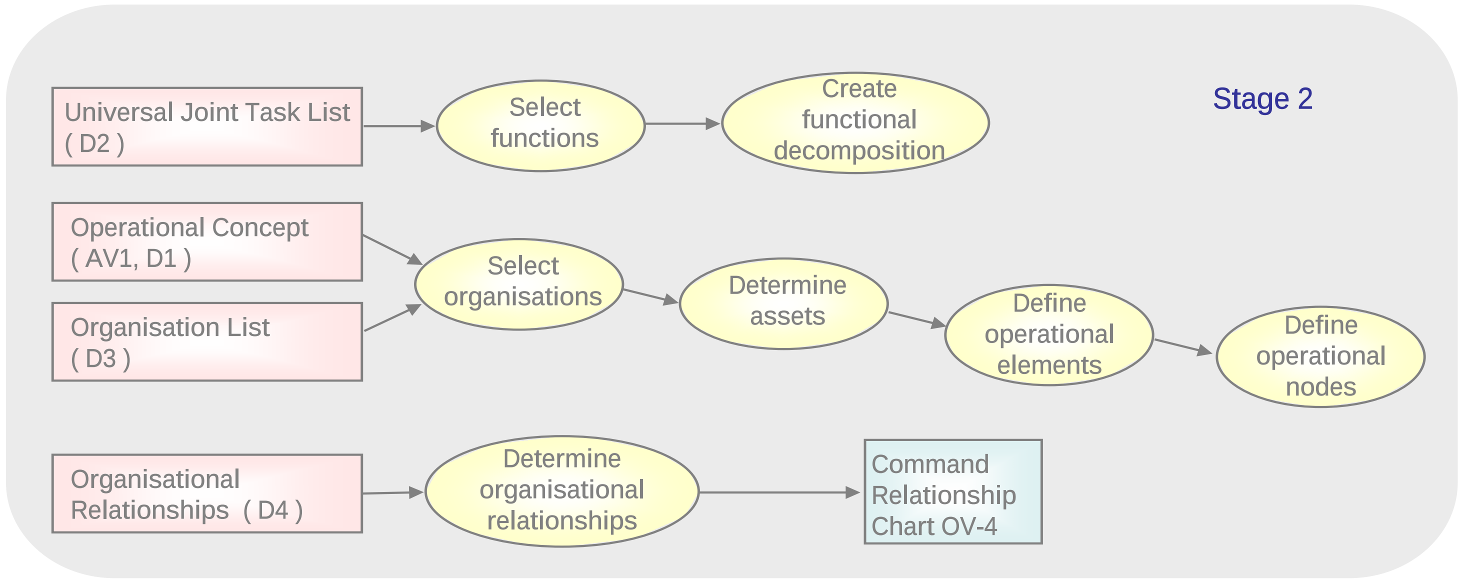


Figure 16. Integrated Data Model Representation of C4ISR Architecture Framework V2.0 – Key Entities Example (Levis 2000).


Stage 1
Operational Concept (AV1, D1)
Create high level operational concept graphic with textual description
Operational Concept Graphic OV-1

(a) Stage 1


Stage 2
Universal Joint Task List (D2)
Select functions
Create functional decomposition
Operational Concept (AV1, D1)
Organisation List (D3)
Select organisations
Determine assets
Define operational elements
Define operational nodes
Organisational Relationships (D4)
Determine organisational relationships
Command Relationship Chart OV-4

(b) Stage 2

Figure 17. Example Process Steps to Generate C4ISR AF V2.0 Artefacts (Levis 2000).

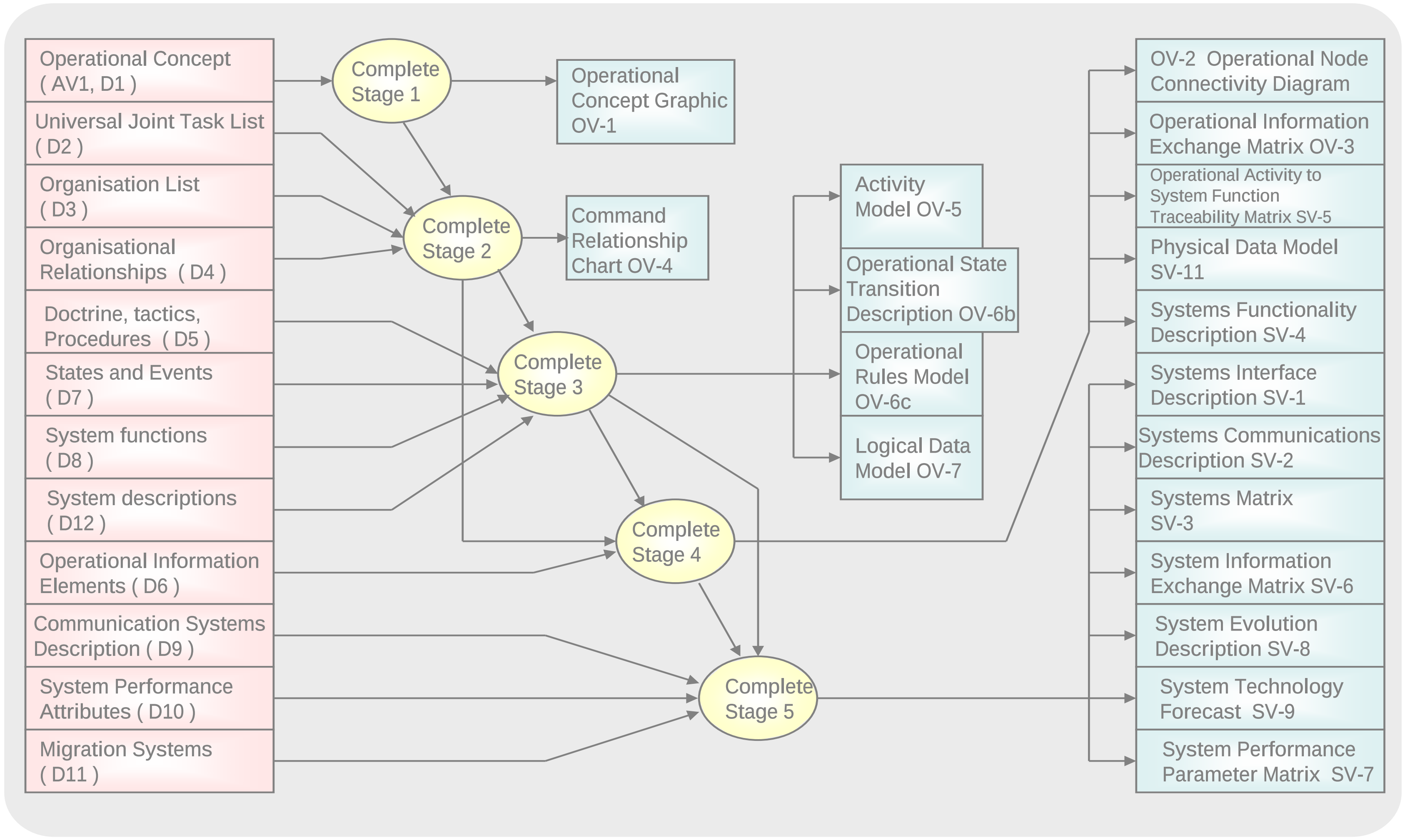

Operational Concept (AV1, D1)
Universal Joint Task List (D2)
Organisation List (D3)
Organisational Relationships (D4)
Doctrine, tactics, Procedures (D5)
States and Events (D7)
System functions (D8)
System descriptions (D12)
Operational Information Elements (D6)
Communication Systems Description (D9)
System Performance Attributes (D10)
Migration Systems (D11)
Complete Stage 1
Complete Stage 2
Complete Stage 3
Complete Stage 4
Complete Stage 5
Operational Concept Graphic OV-1
Command Relationship Chart OV-4
Activity Model OV-5
Operational State Transition Description OV-6b
Operational Rules Model OV-6c
Logical Data Model OV-7
OV-2 Operational Node Connectivity Diagram
Operational Information Exchange Matrix OV-3
Operational Activity to System Function Traceability Matrix SV-5
Physical Data Model SV-11
Systems Functionality Description SV-4
Systems Interface Description SV-1
Systems Communications Description SV-2
Systems Matrix SV-3
System Information Exchange Matrix SV-6
System Evolution Description SV-8
System Technology Forecast SV-9
System Performance Parameter Matrix SV-7

Figure 18. Example 5 Stage Process Summary to Generate C4ISR AF V2.0 Artefacts (Levis 2000).

4. Systems Approach to Problem Solving

4.1 Systems Analysis and Design Concept

The importance of purpose, method and context to systems modelling is readily apparent from the initial discussion thus far. Akin to system modelling, it is similarly essential to establish a common understanding of problem structuring in the engineering context. The SE environment encapsulates methods, tools, and people to support staged engineering activity. This in essence, starts with defining a problem which can be resolved by engineering a technical system solution, then progressively undertaking a sequence of activities from analysis to synthesis (i.e. system design) through to construction and V&V as shown in Figure 19 (adapted from Hawryszkiewycz 1988).

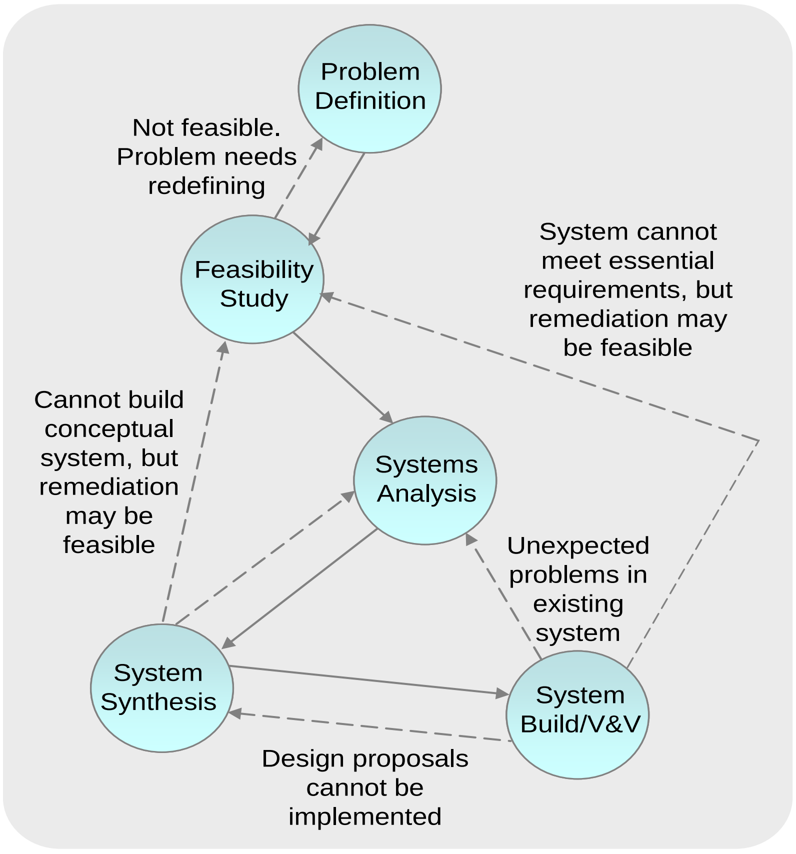

Problem Definition
Not feasible. Problem needs redefining
Feasibility Study
System cannot meet essential requirements, but remediation may be feasible
Cannot build conceptual system, but remediation may be feasible
Systems Analysis
Unexpected problems in existing system
System Synthesis
System Build/V&V
Design proposals cannot be implemented

Figure 19. Staged Problem Solving through Systems Analysis and Design.

The terms analysis and synthesis are Greek in origin, where they mean respectively "to take apart" and "to put together". Analysis can be described as the way the human mind breaks down an intellectual or substantial whole into parts. In contrast, synthesis can be described as the way the human mind combines separate elements or components to form a coherent whole.

Systems analysis is described as "the study of sets of interacting entities to identify alternative courses of action to aid a decision maker identify a better course of action and make a better decision than otherwise might have been made" (Ritchey 1991). In the SE context, it is essentially a directed enquiry to investigate the system operation in the context of the system problem and the system stakeholders against the backdrop of the system environment and the inherent constraints therein.

Analysis is a critical precursor to gain an understanding of the system problem being addressed, in order to be able to synthesise a description of a proposed or revised system, and to determine what is required of it. If there is no existing system to be modified, then the analysis will only provide a set of requirements. These requirements then form the basis for synthesising a proposed system solution.

If an existing system is to be modified, then the analysis will yield a set of requirements to guide evolution of the system from its current state to a future desired state. These requirements form the basis for synthesis of possible modifications to the extant system to achieve the desired result.

While SE effort can be directed towards creating an entirely new system using combinations of analysis and synthesis, often the effort is directed towards modifying, expanding or documenting existing systems.

The scope of a system's requirements typically includes consideration of:

- input/output requirements (including functional transformations and input/output interface definitions);
- technology;
- performance;
- cost-benefits;
- trade-offs;
- constraints; and
- system test requirements, considered over a trajectory of time (Wymore 1993).

The system design activity which follows analysis proposes a new system (or a number of alternatives) that meets these requirements. System design is the aggregation of the process activities of defining the architecture, components, interfaces, data, and the data flows for a proposed system implementation to satisfy the system requirements as summarised in Table 4. If a system already exists, then it can be modified or replaced with a new system. Once the systems design is finalised, it can be built (Hawryszkiewycz 1988).

Table 4 Functions of the System Design Process (adapted from Buede 2000)

Design Function	**Major Inputs**	**Major Outputs**
1. Define system level design problem	Stakeholder's inputs	Originating requirements Operational concept
2. Develop system functional architecture	Originating requirements Operational concept	Functional architecture
3. Develop system physical architecture	Originating requirements	Physical architecture
4. Develop system operational architecture	Originating requirements Functional architecture Physical architecture	Operational architecture
5. Develop interface architecture	Operational architecture	Interface architecture
6. Define the qualification system for the system.	Originating requirements System requirements	Qualification System Design Documentation

Decision evaluation is also an important part of systems analysis and design. Evaluation criteria or metrics are needed, and evaluation activity is needed, to provide a basis for choice among proposed solution alternatives (i.e. perform trade-off activities) that arise from the systems analysis and design activities (Blanchard & Fabrycky 1998).

Three radically different systems modelling paradigms have been spawned to support systems analysis and design activities. Each has developed their own definitions, concepts, methodologies, tools, modelling languages, and diagrammatic forms to apply to problems amenable to systems analysis and design. Unfortunately, many terms used across the different paradigms are common, but have different definitions. It is therefore crucial to understand the context of the modelling in order to understand the semantics associated with the modelling activity and the modelling outcomes.

4.2 Structured Analysis and Design Paradigm

4.2.1 General Principles

Structured analysis and design techniques are fundamental tools of systems analysis, developed from classical systems analysis during the 1960's and 1970's. They were typically applied to system problems in SW engineering, where the system solution entailed significant development of SW components (de Marco 1979).

These techniques were characterised by their use of diagrams to aid communication between users and developers. Data flow diagrams (DFDs) were typically used to document the structured analysis, and structure charts (e.g. flow charts) were used to document the structured design (i.e. the SW architecture was documented as the SW code was implemented).

During the 1980's, computer-based tools emerged which automated the drawings and kept track of the information included in the diagrams in a data dictionary. The use of these tools was coined Computer-aided Software Engineering (CASE).

The essential characteristic of structured analysis is the initial separation of the problem description from the solution (Dickerson & Mavris 2010). Structured analysis views a system from the perspective of the data flowing through it. The function of the system is described by the processes that transform the data. It is reductionist in nature, typified by creating a hierarchy employing a simple abstraction mechanism (Maier & Rechtin 2002).

The method is process driven, and starts with a purpose and a viewpoint. It takes advantage of information hiding through successive decomposition analysis, allowing attention to be focussed on pertinent details at the same level of abstraction for analysis and design, thus avoiding confusion from looking at other details that are not relevant for the particular abstraction under scrutiny. As the level of detail increases, the breadth of information for viewing purposes is reduced, but the integrity of the underlying inter-relationships is preserved.

The result is a set of related graphical diagrams, process descriptions and data definitions that describe the transformations that need to take place, and describe the data required to meet those aspects of a system's requirements that are being implemented as SW component (Peters 1987).

Structured design is the creation or synthesis of SW modules (i.e. SW components) in a module hierarchy based on cohesion and coupling considerations. Cohesion is concerned with the grouping of functionally related processes into a particular SW module. Coupling refers to the flow of information or parameters passing through the modules.

The structure chart documents the module hierarchy or calling sequence of the modules. Best practice in structured analysis and design therefore seeks to minimise the complexity of the SW implementation of the modules, including the interfaces through optimising module cohesion and coupling (Yourdon & Constantine 1978). Different structured analysis and design approaches and supporting computer-based tools have been developed to support SW and SE activity. Early methods of note included the Structured Analysis and Design Technique (SADT), and the Structured Systems Analysis and Design Method (SSADM).

4.2.2 Structured Analysis and Design Technique

SADT is a SW engineering method that performs functional analysis of a given process using successive layers of decomposition, resulting in a description of an information system in terms of a hierarchy of functions and its associated data and control relationships.

It uses diagrammatic notation in the form of activity models and data models to communicate the analysis and design outcomes to assist stakeholders to understand the functions and relationships of the information system under consideration. Since it provides a functional view, it can also be used to represent manufacturing and other business processes and functions in an organisation and their relationships (Marca et al. 1987).

Because SADT is focussed on functions and data and control relationships, the technique can be useful in informing systems analysis and synthesis activities of systems which contain elements other than just SW or IT. However, the lack of representation of non-functional concepts means that the approach cannot be used for system synthesis in isolation from the broader set of system concerns. It also lacks consideration of verification and validation activity. This is essential to confirm that an acceptable solution has been implemented within the feasible solution envelope that solves the original problem.

4.2.3 Structured Systems Analysis and Design Method

SSADM was developed by a UK Government office concerned with the use of IT in government (Eva, 1994). SSADM is a registered trademark of the UK Office of Commerce. It prescribes another systems approach to the analysis and design of information systems based on various stages of activity including:

- carrying out feasibility studies - addresses technical, financial, organisational and ethical concerns;
- investigating the current environment – assumes underlying data will be relatively unchanged even though a new system may be radically different from the old system;
- developing business system options – where a set of new business options is developed offering different ways in which the new system can be produced.
- preparing requirements specifications;
- considering technical system options; and,
- performing logical and physical design.

The method provides explicit guidance on the nature of enquiry to be undertaken, with a particularly strong business emphasis. Its end products are intended to inform engineers how to build the system in terms of specific details on the HW and SW, and informs of the appropriate standards. However, but the construction and verification aspects of the system are not included in the method. The method also lacks consideration of non-functional requirements in the same vein as SADT.

In terms of structured analysis and design, the three most important techniques used in SSADM are:

- Logical Data Modelling – the process of identifying, modelling and documenting the data requirements of the system being designed. The data are separated into entities (things about which a business needs to record information) and relationships (the associations between the entities);
- Data Flow Modelling - the process of identifying, modelling and documenting how data moves around an information system. Data Flow Modelling examines processes (activities that transform data from one form to another), data stores (the holding areas for data), external entities (what sends data into a system or receives data from a system), and data flows (routes by which data can flow); and
- Entity Behaviour Modelling - the process of identifying, modelling and

documenting the events that affect each entity and the sequence in which these events occur[39] (Yourdon 1989).

SSADM implies that the information system will be developed for in-house use, and that a starting concept for the system has already been developed based on current organisational practices. For example, during the feasibility study, the main topics for consideration include project affordability, compatibility with current organisational practices, and whether the new system concept will be socially acceptable within the culture of the organisation.

4.3 Object-Oriented Analysis and Design Paradigm

Object-oriented analysis and design (OOAD) is a radically different SW engineering approach that models a system as a group of interacting objects. Each object represents some entity of interest in the system being modelled, and is characterised by its class, its state (data elements), and its behaviour (Booch et al. 2007) (Maier & Rechtin 2002). Various models can be created to show the static structure, dynamic behaviour, and run-time deployment of these collaborating objects. There are a number of different notations for representing these models, including UML as previously described in Section 3.

Object-oriented analysis (OOA) applies object-modelling techniques to analyse the functional requirements for a system. Object-oriented design (OOD) elaborates the analysis models to produce implementation specifications. That is, OOA focuses on what the system does; OOD on how the system does it.

OOA is the process of analysing a task (also known as a problem domain), to develop a conceptual model that can then be used to complete the task. The conceptual model that results from OOA will typically consist of a set of UML use cases, one or more UML class diagrams, and a number of UML interaction diagrams. It may also include some kind of user interface mock-up.

During OOD, a developer applies implementation constraints to the conceptual model produced in object-oriented analysis. Such constraints can include not only constraints imposed by the chosen architecture but also the non-functional aspects. These consider transaction throughput, response time, the run-time platform, and the development environment, as well as those constraints inherent in the nominated programming language. Concepts in the analysis model are mapped onto implementation classes and interfaces resulting in construction of a model of the solution domain, which is a detailed description of how the system is to be built.

Since the design paradigm is SW focussed, it offers few formalisms to consider non-functional aspects relating to the physical implementation (e.g. technological and environmental considerations). Additional insight on modelling in an object-oriented design paradigm is provided in Appendix B.

[39] [online] URL: http://en.wikipedia.org/wiki/Structured_analysis ; http://en.wikipedia.org/wiki/SSADM; http://sharpertutorials.com/design-methodology-and-system-lifecycle/ ; Office of the Government Chief Information Officer: SSADM v4.2 Structural standards.

4.4 Service-Oriented Analysis and Design Paradigm

Service-oriented analysis and design (SOAD) is another radically different paradigm, with its genesis in distributed computing. In SW engineering, a service-oriented architecture (SOA) is a set of principles and methodologies for designing and developing SW in the form of interoperable services (Stojanovic, 2005). Rather than defining an API, SOA defines an interface in terms of protocols and functionality comprising a "service". SOAD is a SW engineering methodology focused on the development of SW systems by composition of reusable services (service-orientation), often provided by other service providers.

SOA therefore provides a uniform means to organise and integrate widely disparate SW applications, hosted on multiple implementation platforms, and under the control of different ownership domains, typically in a web-based environment.

SOA provides a way for consumers of services, such as web-based applications, to be aware of available SOA-based services as shown in Figure 20 (Peraire 2007).

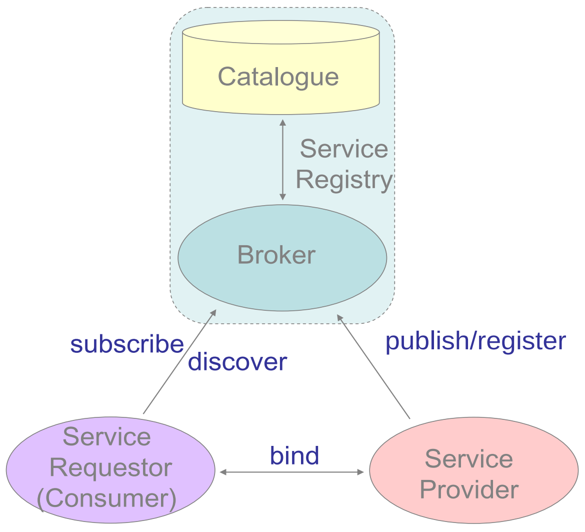


Figure 20. SOA Publish-Subscribe Model (adapted from Peraire 2007).

A service in this context is described as an entity that has a description, and that is made available for use through a published interface that allows it to be invoked by a service consumer. It is generally implemented as a coarse-grained, discoverable software entity that exists as a single instance, and interacts with applications and other services through a loosely coupled, message based communications model. (Densmore & Bohn, 2007).

An event broker features a catalogue repository that contains meta-data describing events or services exposed by the various event producers or service providers.

SOA separates functions into distinct units or services, which developers can make available over a network in order to allow users to combine and reuse them in the production of SW applications. These services and their corresponding consumers communicate with each other by passing data in a well-defined, shared format, or by coordinating an activity between two or more services.

Service-orientation requires loose coupling of services with the operating systems and other technologies underlying the SW applications. Since it involves composition, it shares many characteristics of component-based SW engineering, including the composition of SW systems from reusable components. It differs in one important way, where it adds the ability to dynamically locate necessary services at run-time. These services may be provided as web services, but the essential element is the dynamic nature of the connection between the service users and the service providers.

Since the design paradigm is SW focussed, it offers few formalisms to consider non-functional aspects relating to the physical implementation (e.g. technological, environmental).

4.5 Service-Oriented Modelling and Architecture (SOMA)

Service-oriented modelling is the application of modelling for the specification and design of service-oriented business and SW systems using a variety of architectural styles; these include enterprise architecture, application architecture, service-oriented architecture, and cloud computing.

Any service-oriented modelling methodology typically includes a modelling language that can be employed by both the "problem domain organisation" (e.g. the Business), and "solution domain organisation" (e.g. the IT Department), whose unique perspectives typically influence the "service" development life cycle strategy and the projects implemented using that strategy.

Service-oriented modelling typically strives to create models that provide a comprehensive view of the analysis, design, and architecture of all "SW entities" in an organisation that can be understood by individuals with diverse levels of business and technical understanding. Service-oriented modelling typically encourages viewing SW entities as "assets" (i.e. service-oriented assets), and refers to these assets collectively as "services" (Bell 2008).

The vendor IBM[40] published the Service-Oriented Modelling and Architecture (SOMA) methodology in 2004 as the first publicly announced Service-oriented Architecture-related methodology (Asanjani 2004). SOMA refers to the more general domain of service modelling necessary to design and create SOA. SOMA covers a broader scope and implements SOAD through the identification, specification and realisation of services; components that realise those services (i.e. service components); and flows that can be used to compose services.

SOMA incorporates an analysis and design method that extends traditional object-oriented and component-based analysis and design methods to include concerns relevant to and supporting SOA. It consists of three major phases of identification, specification and realisation of the three main elements of SOA, namely, the services, the components that realise those services, and the flows that are used to compose the services (Endrei et al. 2004).

SOMA is an end-to-end SOA methodology for the identification, specification, realisation and implementation of services (including information services), components, and flows

[40] IBM is a tool and host-platform developer and vendor; a software developer, and a consultancy service provider spanning diverse domains including software engineering, enterprise architecture and business management. [online] URL: http://www.research.ibm.com/.

(processes/composition). It builds on current techniques in areas such as domain analysis, functional areas grouping, variability-oriented analysis (VOA) process modelling, component-based development, object-oriented analysis and design and use case modelling. SOMA introduces new techniques such as goal-service modelling, service model creation and a service litmus test to help determine the granularity of a service[41] (Arsanjani 2004).

SOMA identifies services, component boundaries, flows, compositions, and information through complementary techniques that include domain decomposition, goal-service modelling and existing asset analysis.

Again, since the design paradigm is SW focussed, it offers few formalisms to consider non-functional aspects relating to the physical implementation.

[41] [online] URL: http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/.

5. Systems Engineering Concepts

5.1 What is Systems Thinking?

Thus far, the focus of discussion has centred on relating modelling concepts to problem structuring, and describing the resulting emergence of different systems analysis and synthesis paradigms within the engineering disciplines. It is equally important to understand what is a system in the same regard, and what is systems thinking, before examining the application of the notions in the Defence context.

Systems thinking is a style of thinking and reasoning scientifically about certain classes of problems, phenomena, events and situations based around the concept of a system. A plethora of scientific disciplines employ systems thinking to study different types of problems, spanning both the hard and the soft sciences, ranging from engineering to physics, mathematics, biology, economics, management science, operations research, and the social and cognitive sciences. However, the notion of a system can vary markedly from one scientific discipline to another.

Systems theory is described both as the science of complex systems and the science of wholes; from every perspective, including the science of how wholes form, how they stabilise, how they behave, how they function, how they adapt, how they decay, how they reconfigure, how they become moribund and so on (Hitchens 2007).

General systems theory views the world as a complex system of interconnected parts. In systems thinking, the components or parts of the system are considered in the context of the relationships with each other and other systems, rather than in isolation. However, there is no assertion on the characteristics of the components or parts, other than recognising that they interact, and that their interaction results in formation of a functional unit with specific outcomes.

A system (or system-of-interest) is determined by defining a boundary, and deciding which entities are inside the system and which are outside, and hence part of the external environment (Blanchard & Fabrycky 1998). Systems can manifest in many forms: natural or man-made; physical or abstract; in open or closed form.

In each case, these systems share common defining characteristics including:

- A system has structure – it contains parts that are directly or indirectly related to each other;
- A system has behaviour – it contains processes that transform inputs into outputs (e.g. energy, matter, data);
- A system has interconnectivity – the parts and processes are connected by structural and/or behavioural relationships;
- A system's structure and behaviour may be decomposed via subsystems and sub-processes to elementary parts and process steps;
- A set of rules may also be applied which determine the structure and/or behaviour of the system and its parts (Blanchard & Fabrycky 1998).

In the SE context, systems thinking is described in the INCOSE SE Handbook as a way of thinking, where the primacy of the whole is acknowledged. It manifests through discovery, learning, diagnosis, and dialog that leads to a better understanding of how the (engineered) systems fit into the larger context of day-to-day life, how they behave, and how to manage them (INCOSE 2012).

By viewing a problem as an individual part in relation to a larger whole, it can assist to understand why the problem occurs. The mindset is thus particularly useful for studying systems and system behaviour, and for problem solving in an engineering context.

5.2 What is a System?

While the terms "system" and "component[42]" are used widely in the general community, with many and varied interpretations, these terms have specific significance in the SE context.

The contemporary engineering-based understanding of a system is a "an assemblage of inter-related components working together to form a unitary whole towards some common purpose" (Blanchard & Fabrycky 1998). This is sometimes described as "the whole is more than the sum of its parts" (Mar 1997). A system can also be a grouping of parts (i.e. inter-related components) that operate together for a common purpose (Forrestor 1968).

An engineered system has three essential elements:

1. Components: i.e. the operating parts of the system consisting of input, process and output, where each component can assume a variety of values to describe a system state as set by some control action and one or more constraints;
2. Attributes: i.e. the properties of the components that characterise the system; and
3. Relationships: i.e. the links between components and attributes.

The components also possess the following properties from the perspective of the particular system, where:

1. The properties and behaviour of each component in the system has an effect on the properties and behaviour of the system as a whole; and
2. The properties and behaviour of each component within the system have an effect on at least one other component in the system (Blanchard & Fabrycky 1998).

A component of a system in this context, is a subset of the physical realisation (and hence the physical architecture) of the system to which a subset of the system's functions have been allocated to. As with requirements and functions, there is often a hierarchical structure to the components that comprise the system (Buede 2000).

The INCOSE SE Handbook provides a similar definition of a system as a combination of interacting elements organised to achieve one or more stated purposes, where it is an integrated set of elements, subsystems or assemblies that accomplish a defined objective. These elements can include products (HW, SW, firmware), processes, people, information, techniques, facilities, services, and/or other support elements (INCOSE 2012). Here the term system element is equivalent to the term component used in contemporary SE texts

[42] The term "component" has been superseded by the term "system element" in the 2014 draft of international standard ISO/IEC 15288 and ISO/IEE 12207 (both awaiting ratification). This change acknowledges that components can be independent systems or SoS in different system contexts.

such as (Blanchard & Fabrycky 1998) and (Sage & Rouse 2009).

Bounding the system and identifying its external interfaces is critical to establishing the system identity. The system is bounded by the definition and characterisation of all of the inputs and controls that enter the system, as well as the outputs that the system must produce. The external interface is a connection resource for hooking the external systems to the system. Internal interfaces are the connection resources provided within the system for hooking one component to another (Buede 2000).

Notably, the notion of a system is much broader in the SE context compared to that in a SW engineering context pertaining to SW-intensive systems.

5.3 What is Systems Engineering?

5.3.1 Systems Engineering Origins and Purpose

In abstract terms, (Sage 1992) suggests that SE is a purposeful, managed human activity, where technology is the result of, and represents the totality of the organisation, application, and delivery of scientific knowledge for the intended enhancement of society. He frames the purposeful management of this activity against the backdrop of an organisation interacting with its external environment. Technology management in this context therefore involves the interaction of science, an organisation, and its environment, shown conceptually in Figure 21.

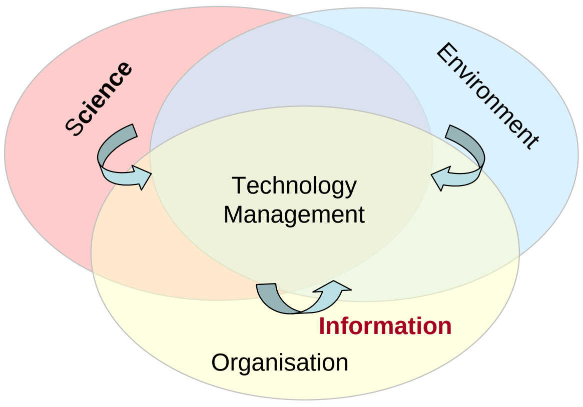


Figure 21. Systems Engineering as a Technology Management Concept (adapted from Sage 1992)

This is underpinned by the exchange of information, where different knowledge perspectives are brought together; knowledge principles are applied to formal problem solving approaches (particularly in new situations); and knowledge practices are employed which encapsulate accumulated wisdom and experience into standard operating policies, directed towards the creation of deliberate technology outcome, i.e. an innovative product or service as shown in Figure 22 (Sage 1992).

In contemporary terms, a point sometimes overlooked – systems engineering (SE) is about the engineering of systems.

Engineering as described by (Buede 2000), has morphed into a discipline for transforming scientific concepts into cost-effective products through the use of analysis and judgement and deliberate application of resources.

Engineering of systems is described as an engineering discipline that develops, matches, and trades-off functions and alternate system resources to achieve a cost-effective, life-cycle balanced product based on the needs of the stakeholders.

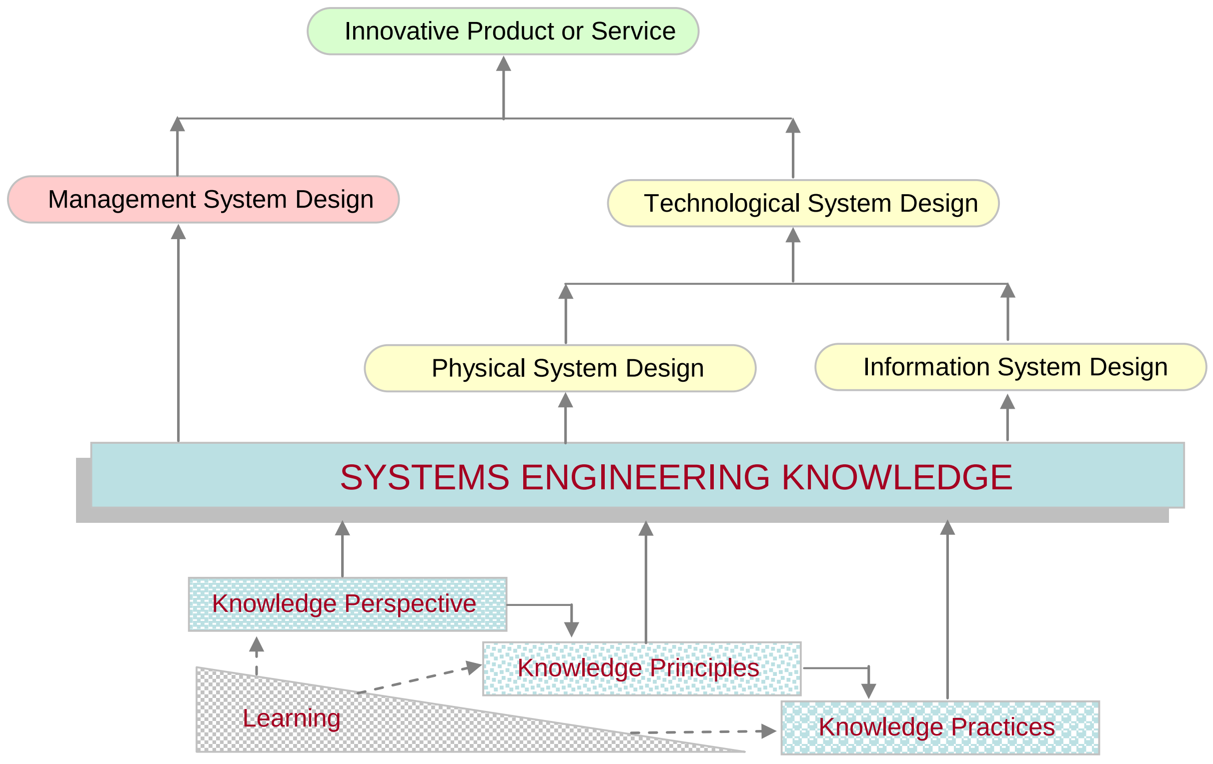


Figure 22. Systems Engineering in the Production of Innovative Products and Services (Buede 2000).

As a problem solving approach, Hitchens suggests SE has several objectives:

- To scope the problem space;
- To explore the problem space;
- To characterise the whole problem;
- To conceive remedies;
- To formulate and manifest the best solution achievable in the situation, constraints and circumstances; and hence
- To solve, resolve or dissolve the whole problem (Hitchens 2007).

The term "systems engineering" first came to the fore in Bell Laboratories in the 1940's[43].

[43] A brief history of the origins of systems engineering is provided on the INCOSE website, accessible at URL: http://www.incose.org/mediarelations/briefhistory.aspx, and in (Buede 2000 p 6.).

This came with the realisation that in many instances, it was no longer possible to rely on design evolution as the primary means to improve upon a system's capability. Likewise, the tools were no longer sufficient to meet the growing demands and increased complexity of the time, necessitating a new multi-disciplinary approach.

Contemporary SE has evolved as an interdisciplinary field of engineering that specifically focuses on how to manage engineering activity to design, deliver and support bespoke complex systems over the duration of their life cycles[44]. The purpose of this engineering activity is to transform customer needs, requirements and constraints[45] into a realised system solution, then to maintain the system over its life cycle. To accomplish the difficult tasks of engineering a complex system, personnel from many disciplines need to be involved in a team effort, including the system stakeholders as shown in Figure 23.

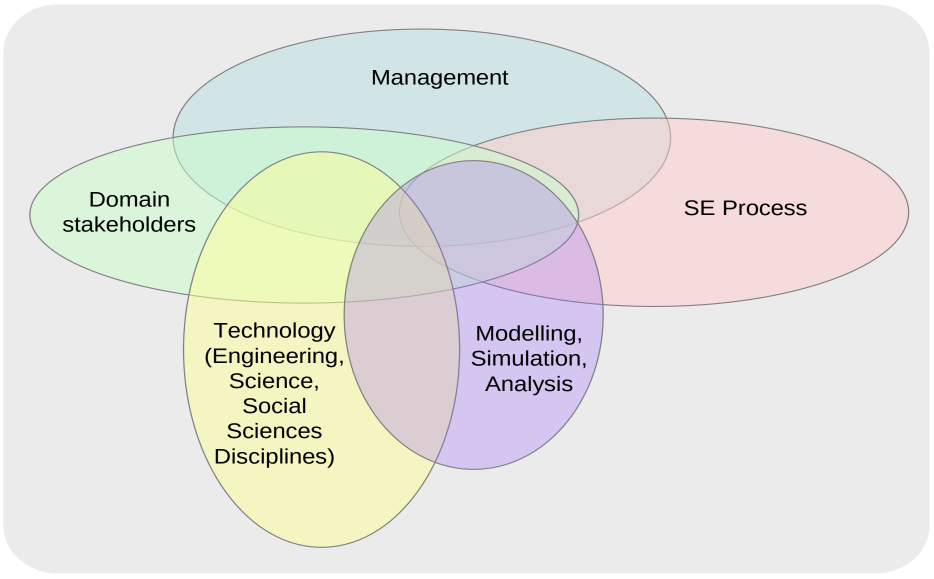

Management
Domain stakeholders
SE Process
Technology (Engineering, Science, Social Sciences Disciplines)
Modelling, Simulation, Analysis

Figure 23. SE Team Expertise Required for Engineering a System (Buede 2000)

Discipline engineers with knowledge of the technologies associated with the system concept provide the expertise needed to make design and integration decisions throughout the development process. Discipline expertise is not only required from traditional fields of engineering such as electrical, mechanical and civil, but also from the social sciences to address psychological, informational, physical and cultural issues of personnel involved in the deployment, operation and maintenance of the system[46].

Additional expertise is required for costing, scheduling, project management, risk management, manufacturing and support purposes. Ignoring any aspect of the system over its life cycle while engineering the system, can result in negative consequences, up to and including total system failure (Buede 2000).

[44] The notion of the system life cycle is described in Section 5.5.
[45] A requirement is one of many statements that constrain or guide the design of the system in such a way that the system will be useful to one or more of its stakeholders. A specification is a collection of requirements that completely defines the constraints and performance requirements for a specific physical entity that is part of the system (Buede 2000).
[46] Also known as "human engineering".

The failure to identify all functions and constraints early in the development process is a source of many cost overruns, poor product performance, and schedule delays (Mar 1992). The earlier the problems are identified, the cheaper they are to address. Early on in the system design, less of the design is fixed, so there are more options to address the problem and fewer components may be impacted (Boucher & Kelly-Rand 2011).

The cost and difficulty to remedy defects is known heuristically to increase exponentially with the stage of life cycle detection, as shown in Figure 24 (Shamieh 2011). It is therefore far more cost effective to prevent, or identify and remedy design defects as early as practicable in the design process.

SE therefore focuses on the deliberate design and management of the engineering work processes, and employment of tools and methods as appropriate, to manage the associated risk of the activity to achieve the technical, cost and schedule outcomes sought over the life cycle of the engineered system.

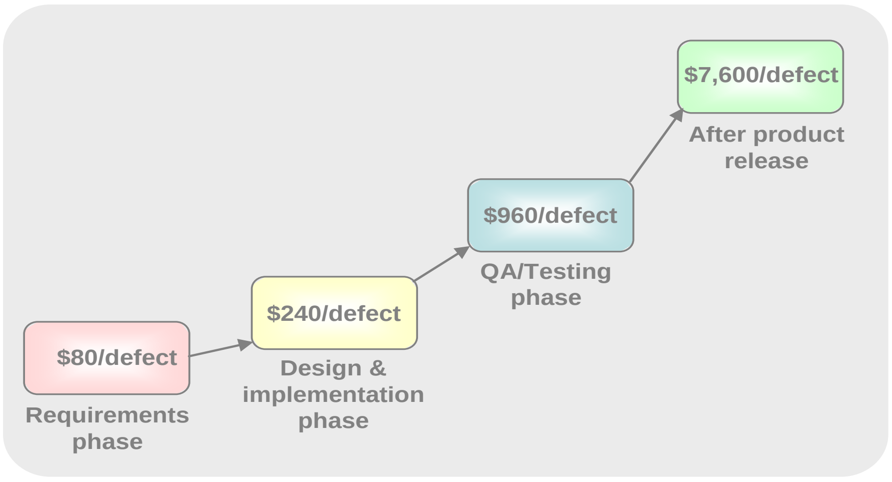


Figure 24. Cost/Defect Comparison with Stage of Life Cycle Detection.

Figure 30 also highlights the criticality of good communication and good team work throughout the design process to minimise the number of hidden defects. This is to ensure the multitudes of perspectives are given adequate consideration in a timely manner to ensure the design progresses smoothly, and as planned towards successful completion.

5.3.2 Basic Notions of SE Process

SE incorporates both technical processes to produce deliberate technical outcomes, as well as management processes to organise the technical effort, whilst controlling cost, schedule and risk.

In its simplest form, systems engineering provides a consistent and logical approach to engineering new system solutions, characterised by:

- A structured and disciplined process that defines problems before seeking solutions;
- A systematic search for solutions that examines trade-offs between alternative solution sets;

- A traceable and disciplined integration process that verifies the product system meets the original requirements and performs the needed functions; and
- An effective information management system, that provides each team member and the customer with information concerning the system being generated (Mar 1992).

This structured and iterative process is exemplified by explicit stages of engineering design activity with typical discernible milestones of achievement such as shown in Figure 25.

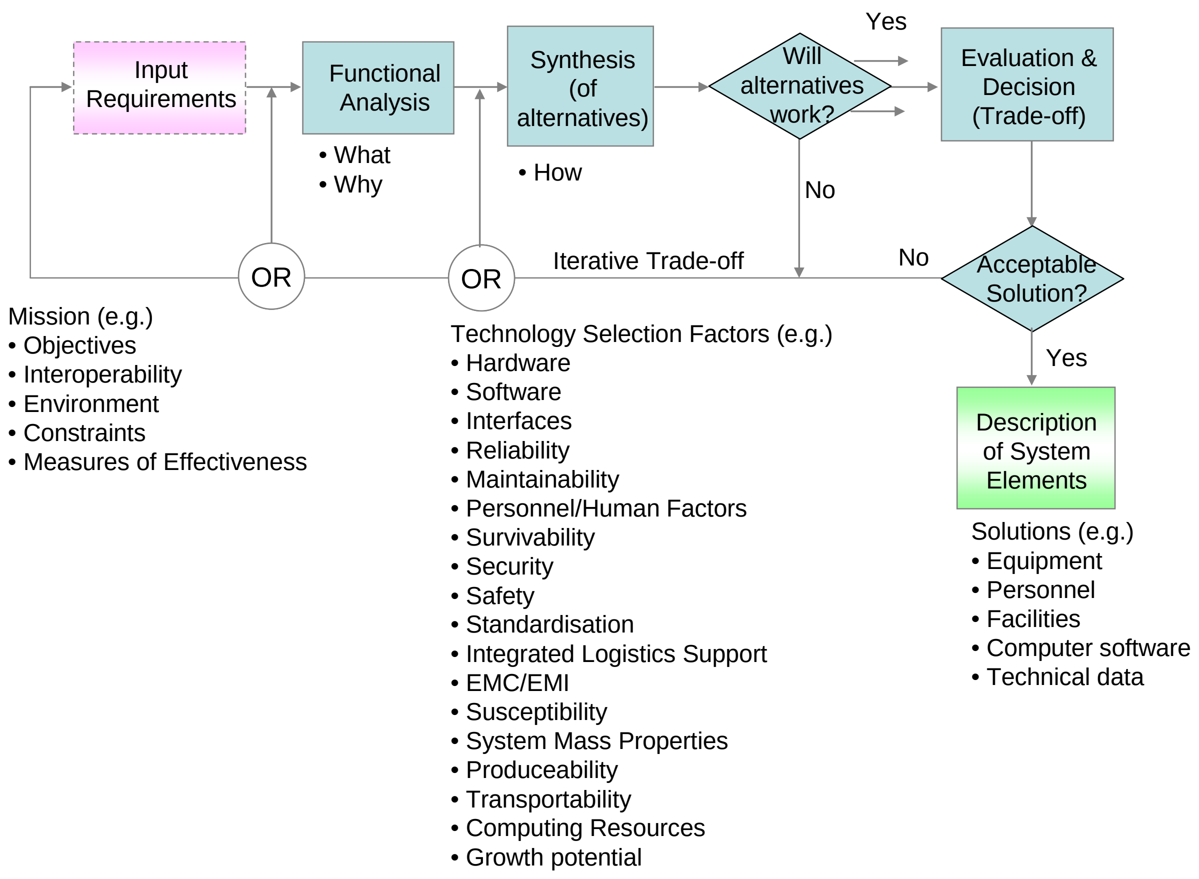


Figure 25. Iterative Stages of Systems Design with Trade-offs (adapted from Hoban & Lawbaugh 1993c)

Work is typically managed within the auspices of an engineering project[47], broken up into structured work packages (i.e. the work breakdown structure (WBS)), and managed using project management principles to achieve the desired outcomes. These outcomes are typically are framed in terms of key decision milestones, percentage work complete, completion of designated project documentation and progressive delivery of system piece-parts towards achieving project completion.

[47] The notion of a project is as described in the Project Management Body of Knowledge publication (PMBOK 2009).

The project documentation requirements are guided both by internal process standard operating procedures (SOPs), and by customer documentation specified to be delivered under the auspices of the contract, commonly in the form of a Contract Data Requirements List (CDRL).

The project typically has an organisational structure closely aligned to the WBS, usually a tree structure, with clusters of personnel assigned to undertake the work outlined in the different work packages. Project documentation is the product of work outcomes achieved within each work package (Hoban & Lawbaugh 1993a).

5.3.3 Engineering Management Planning and Control Basics

Technical planning and controls during system development and production can be extensive, particularly for project-based large complex system development. A list of typical considerations is provided in Table 5 (adapted from Hoban & Lawbaugh 1993b).

Table 5 Typical SE Management Process and Control Considerations

Role and Process Contribution	**Control Processes and Other Influences**
• The role of the project office (if the SE development is project based)	• The role and contribution of Logistics Support Engineering
• The identity and role of the user	• Applicable standards
• The identity and role of the Stakeholders	• Applicable procedures and training
• The role of the Contracting Office Technical Representative (if appointed)	• Configuration Baseline control process
• The role and contribution of Systems Engineering	• Change control process
• The role and contribution of Design Engineering	• Interface control process
• The role and contribution of Speciality Engineering	• Control of contracted or subcontracted engineering
• The role and contribution of Manufacturing Engineering	• Project Data control and information management process (i.e. documentation and other project pertinent)
• The role and contribution of Test Engineering	• Make-or-buy control process
• The role and contribution of Logistics Support Engineering	• Parts, materiel and process control
• System definition process	• Manufacturing control process
• System analysis and design process	• Life cycle cost management control
• System decomposition process	• Risk management process
• Trade study process	• Quality control
• Use of mathematical models and simulation	• Safety control

• System qualification process	• Security control
• System acceptance process	• Contamination control
• EMI/EMC	• Reliability and supportability planning and control
• Survivability and vulnerability	• Integrated Logistics support planning and control
• Technical performance measurement	• Control gates (review milestones, decision governance)
• Reporting process	• Integration planning and control
• Internal technical reviews	• Verification planning and control
• Tools and resources to be used	• Acceptance testing planning and control
	• Validation planning and control

By taking a holistic view of the development effort, SE provides a formalised structured engineering development process to combine the various technical contributions into a unified team effort. This unified effort spans the entire life of the system, from initial concept articulation through analysis and design activity, to production, operation and support of the engineered system over its life cycle.

5.3.4 Basic Activities and Responsibilities

From the earliest notions of SE, the process has been depicted as spanning typical project activities as shown in Figure 26 (Hoban & Lawbaugh 1993a).

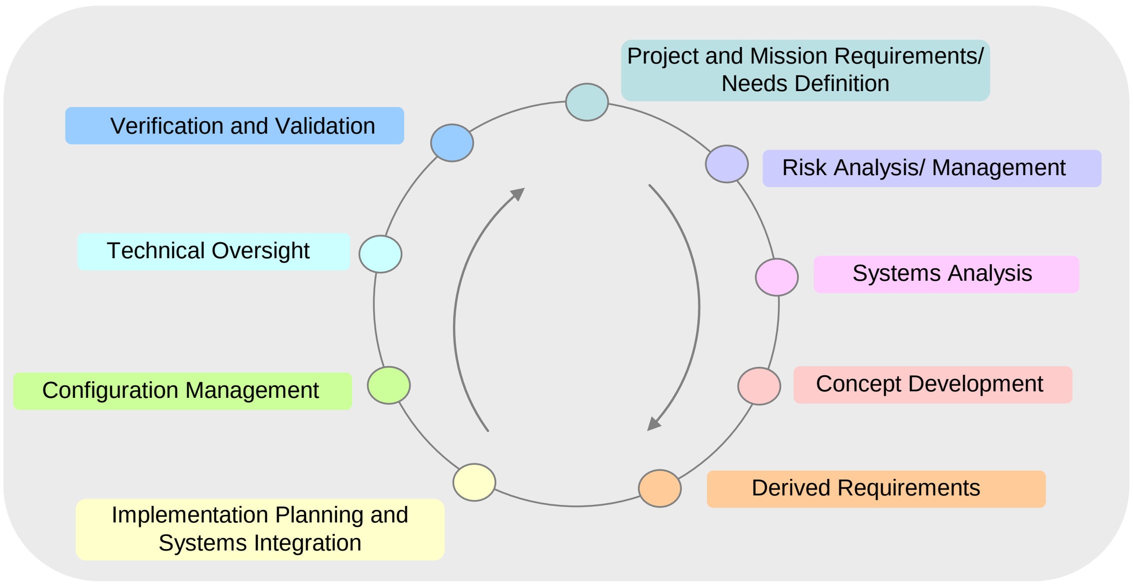

Project and Mission Requirements/ Needs Definition
Verification and Validation
Risk Analysis/ Management
Technical Oversight
Systems Analysis
Configuration Management
Concept Development
Derived Requirements
Implementation Planning and Systems Integration

Figure 26. Systems Engineering Cycle (Hoban & Lawbaugh 1993a).

These include (Hoban & Lawbaugh 1993a):

1. Project and Mission Requirements/Needs Definition

   The process by which the needs of the customer (including parliamentary and budgetary authorities are determined. This allows the Systems Engineer to define the requirements for a system that will meet the needs of the customer. The requirements provide a hierarchical description of the customer's desired product system as seen by the Systems Engineer.

   The interaction between the customer and the Systems Engineer to develop these requirements is one way of ensuring the customer perspective is captured, while ensuring the customer is informed about the value proposition, and the customer can make a judgement that they are willing to pay for a product system that meets the specified requirements (Hoban & Lawbaugh 1993b).

2. Risk Analysis/Management

   Risk management is an ongoing process to identify and assess the risks involved with the development and operation of the system, including technical, schedule, cost and organisational risk. The Systems Engineer is responsible for developing an implementation plan to control and if possible, reduce the risks incurred by the project.

3. Systems Analysis

   Systems analysis involves understanding how the key mission and system functional elements interact. The mission analysis translates the user's needs into functional and performance requirements and design constraints. A functional analysis takes these requirements and breaks them down to specific tasks.

4. Concept Development

   Concept development is a process of making informed trade-offs among various options to select the one that best meets the requirements and design constraints. This activity produces a preliminary design and implementation architecture.

5. Derived Requirements Definition

   Requirements Definition entails translating mission and functional analysis results, system operational concepts, and the selected system architecture into a set of system performance and interface requirements (without presenting an actual design solution).

6. Implementation Planning and Systems Integration

   Implementation planning and systems integration are complex activities to produce a coherent, integrated set of implementation tasks and responsibilities for the detail design (i.e. of the implementation or solution), development, fabrication, verification, operation and maintenance of the required system. It requires negotiation between the system requirements definition personnel and the system implementation personnel whilst considering the project constraints of schedule and budget, and avoiding unnecessary risk.

7. Configuration Management

   Configuration management is an activity that ensures that controlled definition of all engineering documentation is maintained and the information is distributed to the appropriated parties in a timely manner. Importantly, this activity is the mechanism by which the system development process is documented (i.e. the design knowledge is captured).

8. Technical Oversight

Technical oversight ensures all subsystems work together as intended, and implements mechanisms to ensure to guarantee the developed and documented architected concept is not inadvertently changed during the development process. This allows the system developer to certify that the system (as tested) will meet the customer's requirements. This entails a number of reviews and audits that gather consensus from all parties involved that the effort at any given time is correct and adequately planned for the continuance of the work.

The Systems Engineer is responsible for communicating the customer's requirements to the design organisation as to what to design and build or code. As the requirements are allocated, they inevitably become linked to the system architecture and product breakdown, which consists of the hierarchy of project, and its subservient systems, segments, subsystems, components and elements (Hoban & Lawbaugh 1993b).

The Systems Engineer is responsible for assuring the systems requirements are understood and correctly implemented by the design organisation, and therefore needs to work closely with the design organisation over the duration of the project. Importantly, the Systems Engineer must recognise the initial set of systems requirements may not be "perfect", where during the design evolution, or because of the inability of a subsystem to meet its intended functional and performance requirements, changes in the systems requirements will be necessitated. These changes are an essential and normal part of the design process.

9. Verification and Validation

Here, the characteristics and performance of the implemented system are compared to the requirements and specifications. Tests, analyses and demonstrations are performed to verify that the hardware and software satisfactorily meet the function and performance requirements of the system specifications.

The engineering design of complex systems thus involves making many decisions during the development process. To be successful, these need to made using a rational, explicit, and traceable process, i.e. the engineering organisation's instantiation of a SE process. This is typically expressed within the organisation's standard operating procedures (SOPs). A sample of system design decisions supporting SE process activity is provided in Table 6. Typical project documentation produced during development is described in Table 7.

Table 6 Sample of Decisions Made during System Design (adapted from Buede 2000)

Development Phase	Example Decisions in Systems Engineering
Conceptual Design	• Should a conceptual design effort be undertaken? • Which system concept (or mix of technologies)should be

	the basis of design? • Which technology for a given subsystem should be chosen? • What existing hardware and software can be used? • Is the envisaged concept technically feasible based on cost, performance, and schedule requirements? • Should additional research be undertaken before a decision is made?
Preliminary Design	• Should a preliminary design effort be undertaken • Which specific physical architecture should be chosen (from several alternatives)? • Which physical resource should a function be allocated to? • Should a prototype be built? If so, to what degree of reality? • How should verification, validation and acceptance testing be structured?
Full-scale Design	• Should a full-scale design effort be undertaken? • Which configuration items should be bought rather than manufactured? • Which detailed design should be chosen for a particular component, given that one or more performance requirements are critical?
Integration & Qualification	• What is the most cost-effective schedule for implementation activities? • What issues should be tested? • What people, equipment, facilities should be used to test each issue? • What models of the system should be developed or adapted to enhance the effectiveness of integration? • How much testing should be devoted to each issue? • What adaptive testing (Fall-back testing in case of failure) should be planned for each issue?
Product Refinement	• Should product improvement be introduced at this time? • Which technologies should be the basis for the product improvement? • What redesign is best to meet some clearly defined deficiency in the system? • How should the refinement of existing systems be implemented, given schedule, cost, performance and risk

	criteria? • Are there any external interdependencies affected? If so, has the impact been accounted for in the schedule, cost, performance and risk criteria?

Table 7 Sample of Typical Requirements Documentation (Buede 2000).

Document Type	**Document Contents**
Problem Situation or Mission Element Need Statement	• Definition of stakeholders and their relationships • Stakeholder's description of the problem and its context • Description of the current system
Systems Engineering Management Plan (SEMP)	• Definition of mission requirements • Definition of the systems engineering development system (requirements, architectures, interfaces)
Operational Need or Operational Requirements Document or Originating Requirements Document (ORD)	• Definition of the problem needing solution by the system, including the context and external systems with which the system must interact • Definition of the operational concept upon which the system will be based • Creation of the structure for defining requirements • Description of the Stakeholder's requirements in the Stakeholder's language with considerable breadth but little depth • Trace of every requirement to a recorded statement or opinion of the stakeholder. • Description of trade-offs between performance requirements, including cost, schedule and operational effectiveness
System Requirements Document or Mission Requirements Document (SRD)	• Restatement of the operational concept on which the system will be based • Definition of the external system interfaces and interactions in engineering terms • Restatement of the operational requirement in engineering terms • Trace of every requirement to the previous document • Justification of the engineering version of the requirements in terms of analyses, expert opinions, and stakeholders meetings • Description of a test plan for each requirement

System Requirements Verification or Systems Acceptance Document	• Documents analyses to show that the requirements in the systems requirement documentation are consistent, complete, and correct, to the degree practicable. • Demonstrates that there is at least one feasible solution to the design problem as defined in the system requirements documentation, and that it has been achieved.

5.3.5 Documentation in Systems Engineering

As evident from Table 7, formalised capture and management of specific types of information created and used to support the SE management and technical sub-processes is a key feature of SE practice. Traditionally, a document-based approach has been used to convey system requirements, design and test information. This is characterised by the generation of textual specifications, design documents, test documents and drawings, in hard copy or electronic file format.

This documentation[48] is exchanged between the respective stakeholders, including customers, users, developers and testers, to solicit input, review and response, and record a myriad of significant decisions towards forging sufficient common and agreed understanding. SE practice places particular emphasis on controlling the documentation and ensuring the document and drawing contents are valid, complete and consistent, and that the system solution implemented complies with the documentation, including the originating specifications (Friedenthal et al. 2008).

5.3.6 Formalisation of Systems Engineering as a Discipline

A significant milestone in the formalisation of SE as a discipline came with the publication by US DoD of the military standard MIL-STD-499 in 1969, and the mandate of its use in industry for design and development of US Defense major military capability. This had the impact of re-aligning entire company organisational structures and standard operating procedures, as well as their skill bases, to facilitate ease of compliance with the directives of the military standard.

The discipline of SE has continued to evolve, both tools and methods, with various interpretations and adaptations based on the same theme published in SE Handbooks written by large science and engineering organisations including NASA, Jet Propulsion Laboratories, and more recently, by INCOSE (Hoban & Laughbaugh 1993a), (NASA, 2007), (Jansma 2006), (INCOSE 2012) [49].

Notably, it was not until as recently as 1990 that a professional society was founded for the discipline: a group of representatives from a number of US corporations coming together to form the National Council on Systems Engineering (NCOSE). Increasing prominence of SE outside of the US as a discipline of significance led to the repositioning of the organisation

[48] The terms ‘documentation’ and “drawing’ are both ‘information items’.

[49] (Buede 2000) describes a number of definitions of systems engineering drawn from pre-eminent sources including MIL-STD-499A, Sage (1992), Wymore (1993), Forsberg and Mooz (1992) and INCOSE in the 1999 edition of the SE Handbook.

as the foremost international representative of the global SE community, and renaming of the organisation in 1994 as the International Council on Systems Engineering[50].

Coinciding with the demise of use of military standards by the US DoD during the 1990's, responsibility for formalising and evolving SE process methods and other related engineering technical and process standards has since been devolved to a number of international standards organisations and commercial corporate bodies. These include the International Standards Organisation (ISO)[51], the Electronics Industries Alliance (EIA)[52][53], and the Institute of Electrical and Electronic Engineers (IEEE) Standards Association[54].

The commensurate rise in use of COTS products and use of open standards, replacing bespoke engineering development using specialised military standards, has led to numerous proprietary industry-based approaches being adopted as de facto standards. One of the most prominent of these being that associated with the desktop computing environment, typified by the use of Microsoft Office™ product suite running on the Windows Operating System™ to provide basic word processing and spreadsheet functionality.

Crucially, while SE practice is inherently multi-disciplinary, and draws extensive contributions from other mainstream engineering disciplines (e.g. electrical engineering, electronics engineering, computer systems engineering, SW engineering, mechanical engineering) to achieve the desired technical outcomes, SE has maintained a separate identity from these other specialised engineering disciplines.

Over the last fifteen years in particular, considerable effort has been devoted to harmonising the key SE standards (Croll 2002) and compiling and publishing internationally recognised bodies of knowledge (BOK). The progressive emergence of discrete bodies of knowledge as separate formal disciplines, from earlier military and commercial standards to contemporary notions of engineering and enterprise architecture, are shown in Figure 27.

Key development milestones relating to SE process standards and capability models, together with the SW engineering counterparts are shown in Figure 28[55] (Martin 1998), ISO/IEC JTC1/SC7/WG7 2002), (Doran 2008). For completeness, the timeline for different EA frameworks, discussed in Section 9, is also provided in Figure 29 to highlight the correlation between the respective stages of maturation of SE practice with evolving maturity of SW engineering practice and EA architecture practice (Hause 2010), (Friedenthal et al. 2008)[56].

[50] [online] URL: http://en.wikipedia.org/wiki/International_Council_on_Systems_Engineering ; http://www.incose.org/

[51] [online] URL: http://en.wikipedia.org/wiki/International_Standards_Organization ; http://www.iso.org/iso/home.html

[52] [online] URL: http://en.wikipedia.org/wiki/Electronic_Industries_Alliance#EIA_standards ; http://www.eciaonline.org/eiastandards/

[53] The EIA was renamed from Electronic Industries Association to Electronic Industries Alliance in 1997. The EIA ceased operations in February 2011, and designated ECIA to continue to develop standards for interconnect, passive and electro-mechanical electronic components under the ANSI designation of ECIA standards.

[54] [online] URL: http://standards.ieee.org/ ; http://en.wikipedia.org/wiki/IEEE_Standards_Association ;

[55] (Sheard & Lake 1998) provides a useful overview of the various SE standards and models of the time, and discusses similarities and differences in definition, scope and applicability within the respective standards.

[56] [online] URL: http://www.bespokesystems.net/ea/timeline/; http://www.opengroup.org/openca/cert/methods.tpl.

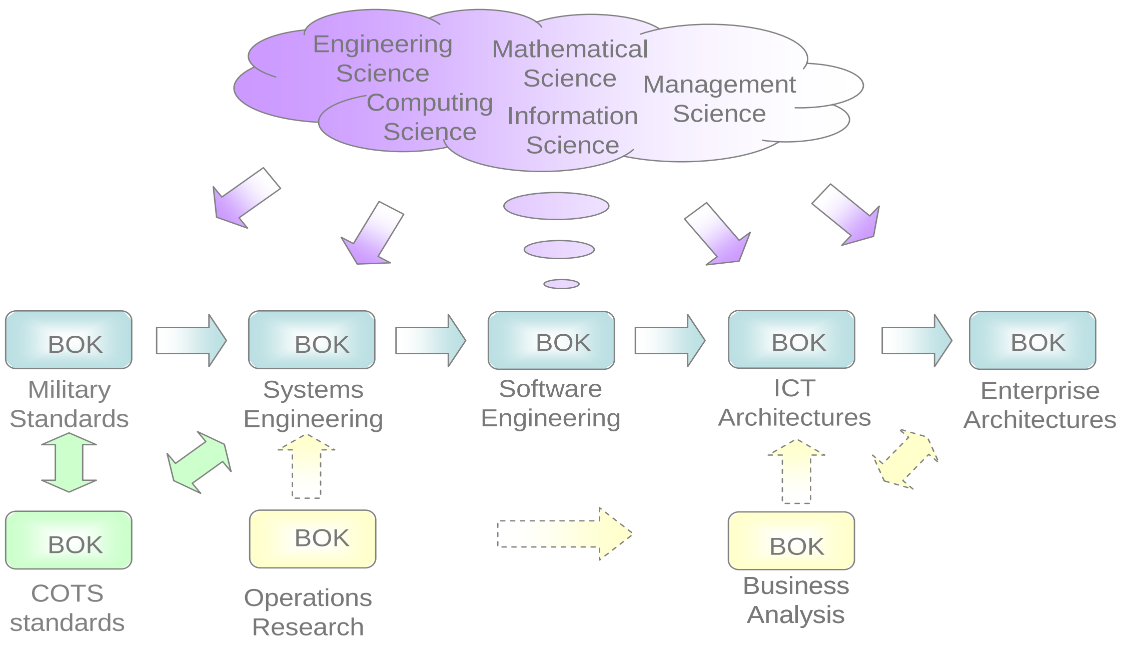


Figure 27. Progressive Emergence of Systems Engineering, Software Engineering and Enterprise Architecture Practice Formal Disciplines.

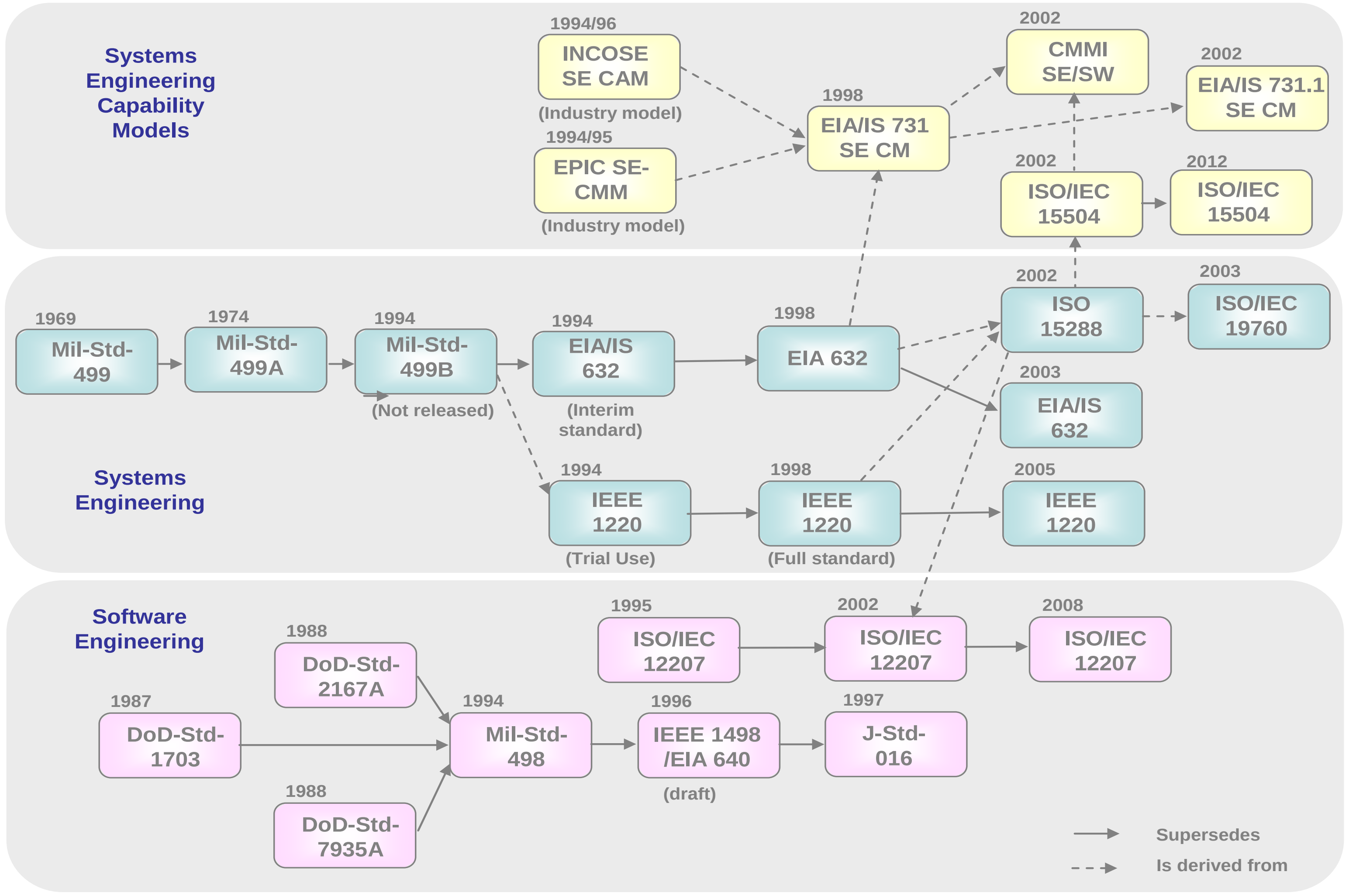


Figure 28. Key Development Milestones - SE Standards, System Capability Models, and SW Engineering Standards.

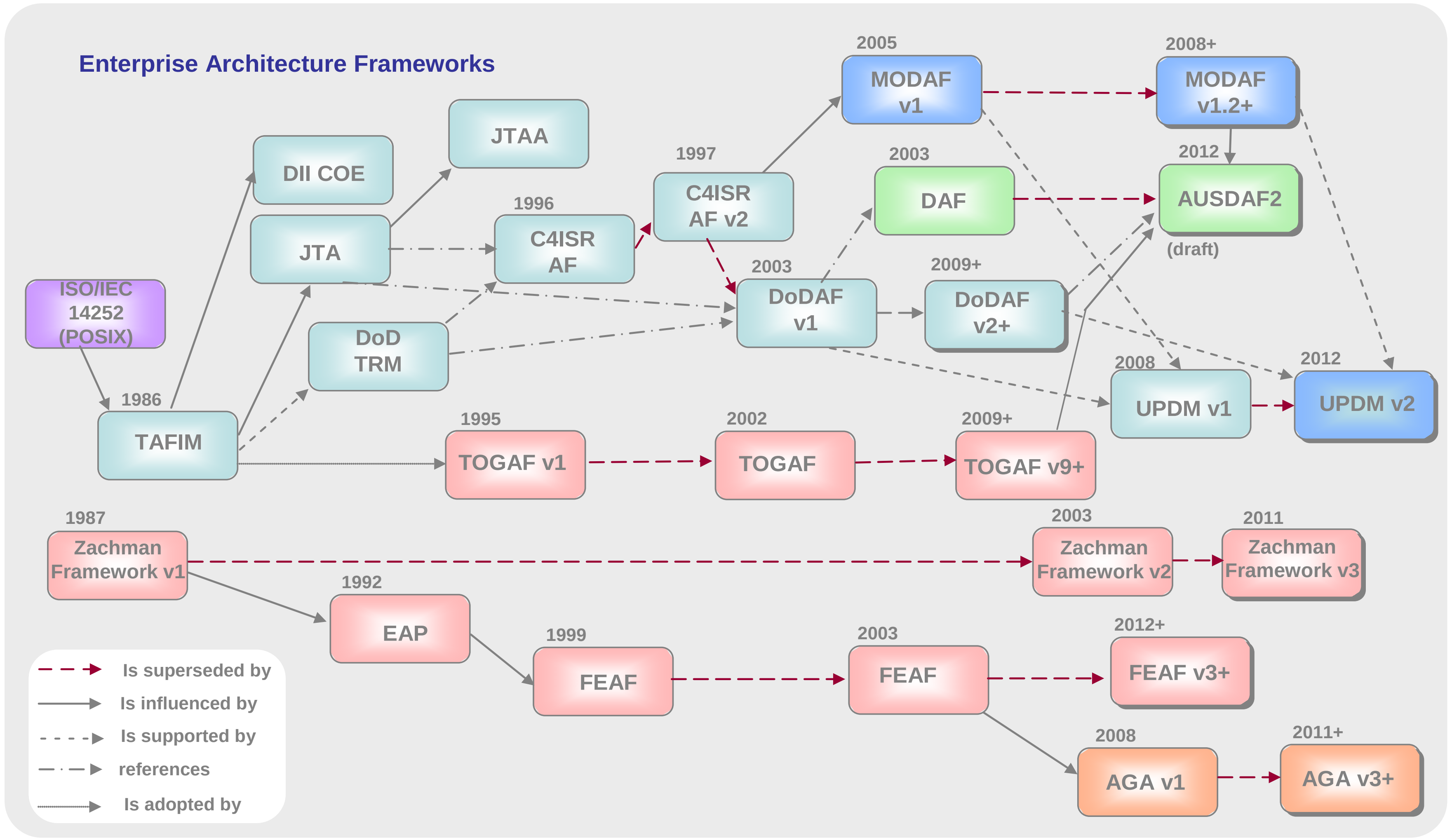


Figure 29. Key Development Milestones - Enterprise Architecture Frameworks.

In recent times, the SE discipline has drawn heavily from the SW engineering discipline. However, as shown in Figure 33, the two disciplines maintain separate identities, with separate bodies of knowledge, as revealed in knowledge repositories such as "The SE Body of Knowledge" (SEBoK) published by INCOSE (SEBoK 2012); and "The Software Engineering Body of Knowledge" (SWEBOK) published by IEEE (SWEBOK 2004).

5.3.7 Contemporary Systems Engineering

"Systems Engineering becomes the bridge between the system problems generated by society and solutions provided by technology" A. Wayne Wymore (Wymore 1993).

Contemporary SE incorporates the concepts and processes utilised widely in industry and encapsulated in international standards such as ISO/IEC 15288:2008, ANSI/EIA-632:2009, and IEEE-1220-2005 as shown in Figure 30 (Estefan 2008). The typical context for applying SE process standards with respect to an organisation or enterprise is shown in Figure 31 as represented in EIA-632.

INCOSE, the international professional body representing the profession of SE has elected to support standard EIA 15288:2008 for the practical application of SE concepts and processes within an organisation or enterprise.

The IEEE 15288 standard describes the enduring concept of a cradle-to-grave life cycle for a product or system, from initial conception to final disposal, mapped to specific technical and management process activities spanning different domain areas across the organisation. Each domain area has its own technical and management process steps, with required outputs, outcomes and completion criteria as described in Figure 32 (ISO/IEC JTC1/SC7/WG7 2002).

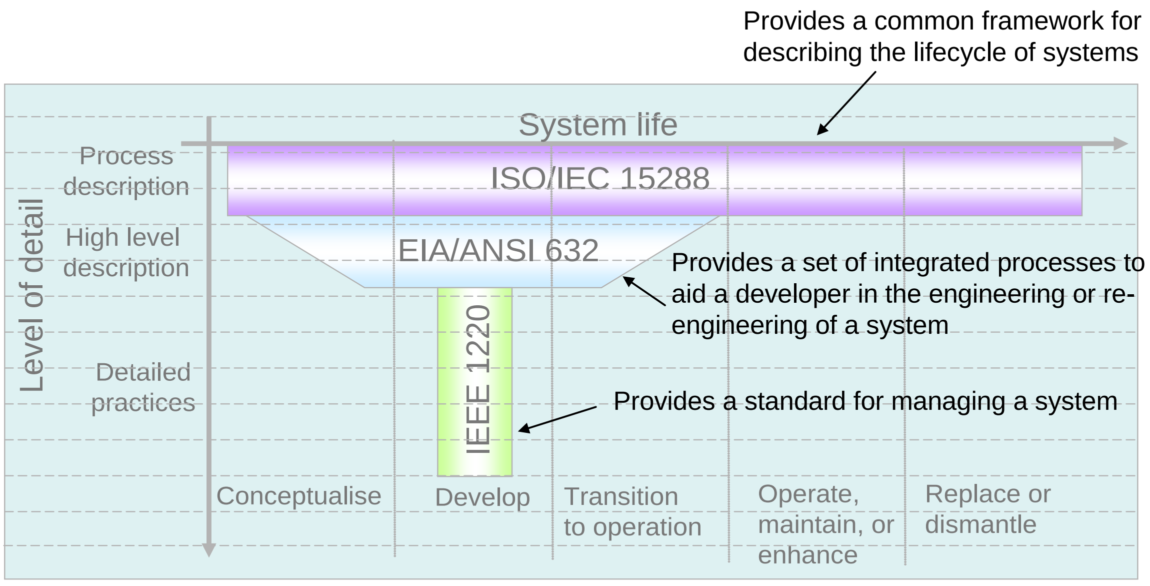

Provides a common framework for describing the lifecycle of systems
System life
Process description
ISO/IEC 15288
Level of detail
High level description
EIA/ANSI 632
Provides a set of integrated processes to aid a developer in the engineering or re-engineering of a system
IEEE 1220
Detailed practices
Provides a standard for managing a system
Conceptualise
Develop
Transition to operation
Operate, maintain, or enhance
Replace or dismantle

Figure 30. Systems Engineering Process Standards and Capability Model Comparison.

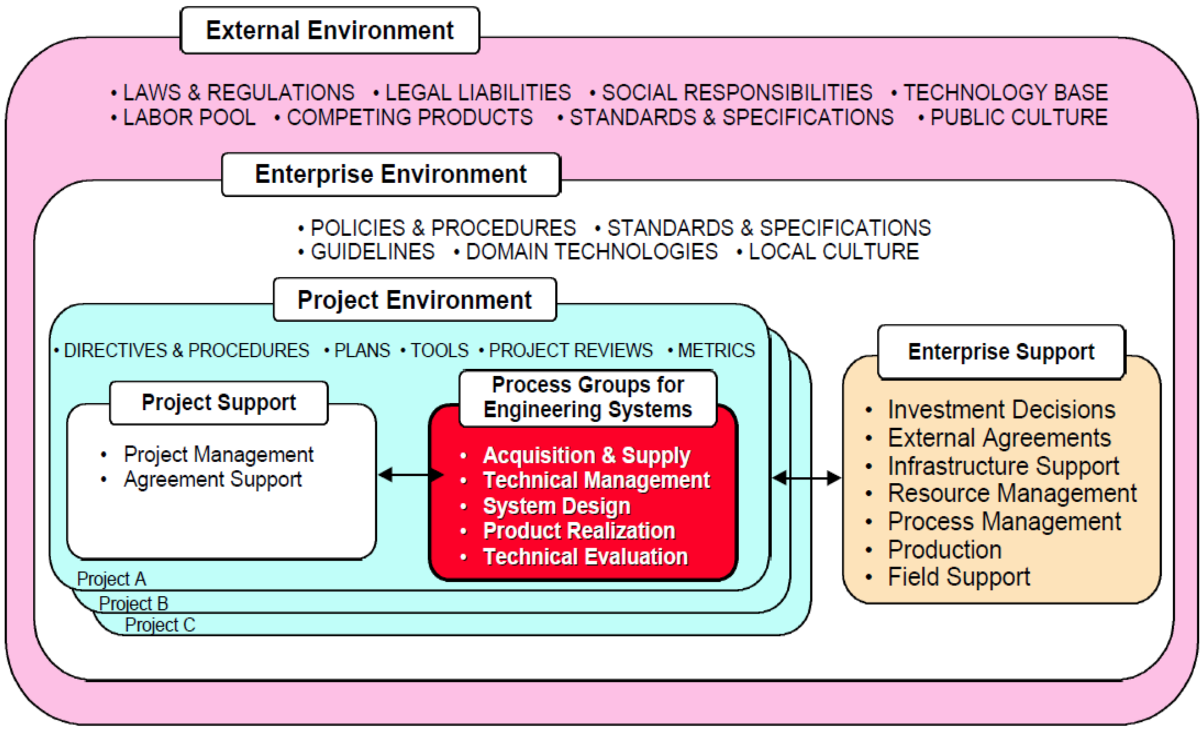

External Environment
• LAWS & REGULATIONS • LEGAL LIABILITIES • SOCIAL RESPONSIBILITIES • TECHNOLOGY BASE
• LABOR POOL • COMPETING PRODUCTS • STANDARDS & SPECIFICATIONS • PUBLIC CULTURE
Enterprise Environment
• POLICIES & PROCEDURES • STANDARDS & SPECIFICATIONS
• GUIDELINES • DOMAIN TECHNOLOGIES • LOCAL CULTURE
Project Environment
• DIRECTIVES & PROCEDURES • PLANS • TOOLS • PROJECT REVIEWS • METRICS
Project Support
Project Management
Agreement Support
Process Groups for Engineering Systems
Acquisition & Supply
Technical Management
System Design
Product Realization
Technical Evaluation
Project A
Project B
Project C
Enterprise Support
Investment Decisions
External Agreements
Infrastructure Support
Resource Management
Process Management
Production
Field Support

Figure 31. Enterprise and Project Context for Applying the EIA-632 SE Process Standard.

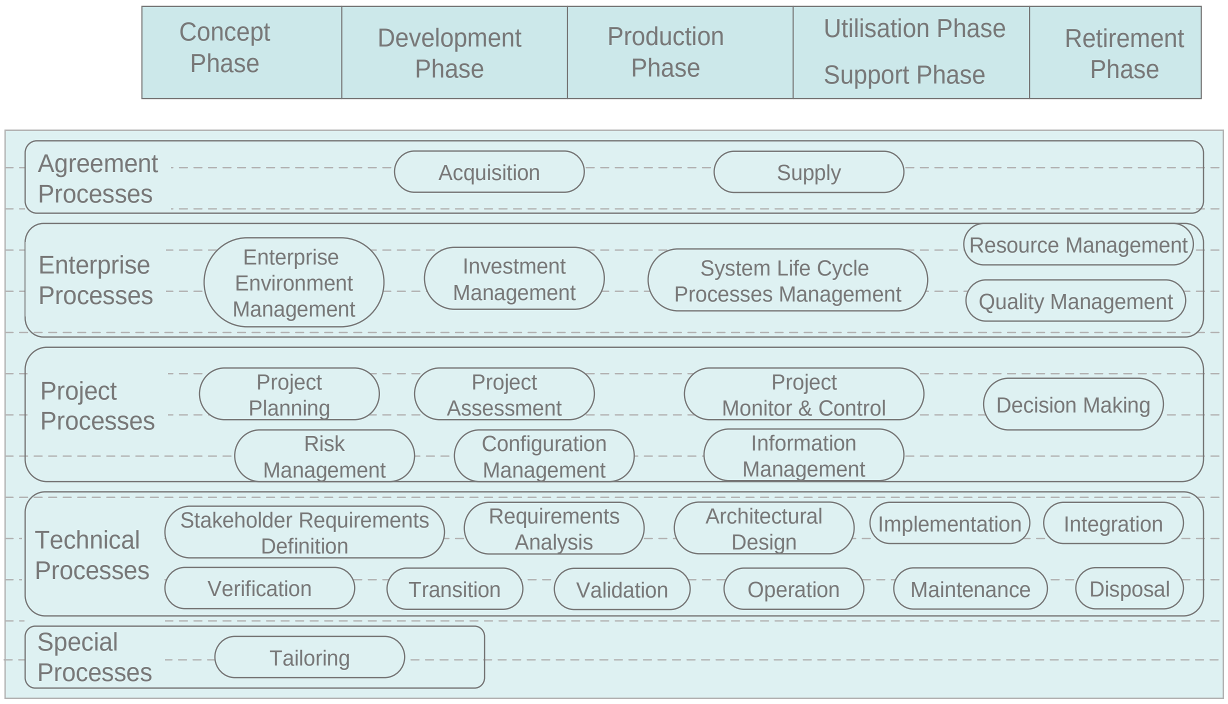


Figure 32. Enterprise Context for Applying EIA 15288 Process Standard.

While there are still many and varied definitions of SE, the version offered by INCOSE in their SE Handbook based on ISO/EIA 15288:2008 is widely accepted internationally as encapsulating the essence of SE. SE is defined by INCOSE as:

"an interdisciplinary approach and means to enable the realization of successful systems. It focuses on defining customer needs and required functionality early in the system development cycle, documenting requirements, then proceeding with the design synthesis and system validation while considering the complete problem including:

- *operations*
- *performance*
- *test*
- *manufacturing*
- *cost and schedule*
- *training and support and*
- *disposal.*

SE considers both the business and technical needs of all customers with the goal of providing a quality product that meets the user needs" (INCOSE 2012).

In his 1997 paper, Mar asks probing questions such as what is SE, whether it is a process or skill code, and what is the role of SE in the engineering of complex systems (Mar 1997). These questions are still relevant today to Defence.

The INCOSE SE Handbook suggests that SE is a combination of concepts:

- a perspective based on systems thinking;
- a process; and
- a profession (as discussed in para. 5.3.2).

Key tenets include recognising it is a disciplined approach for systematically addressing the engineering of human-made or technical systems, and it requires the application of formalised technical and management processes over a system life cycle to achieve an intended engineering outcome.

5.4 The Systems Engineering Process

Contemporary SE manifests where formalised technical processes are used to define the requirements for a technology-based system (which needs to be engineered to satisfy an identified need); to transform the requirements into an effective product imbued with the required attributes; to permit consistent reproduction of the product where necessary; to use the product to provide the required services; to sustain the provision of those services; and to dispose of the product when it is retired from service (ISO/IEC 15288:2008). The system solution is to be developed on a basis that balances cost, schedule, performance and risk.

There are many and varied instantiations of the SE process, depending on many and varied factors including the source and nature of the originating requirement, resourcing considerations, organisation competencies, regulatory and governance requirements, and end user considerations. An overview of a typical SE process is illustrated in Figure 33 (Blanchard & Fabrycky 1998). A more detailed and explicit description of the SE process is provided in IEEE standard 1220-2005 as shown in Figure 34.

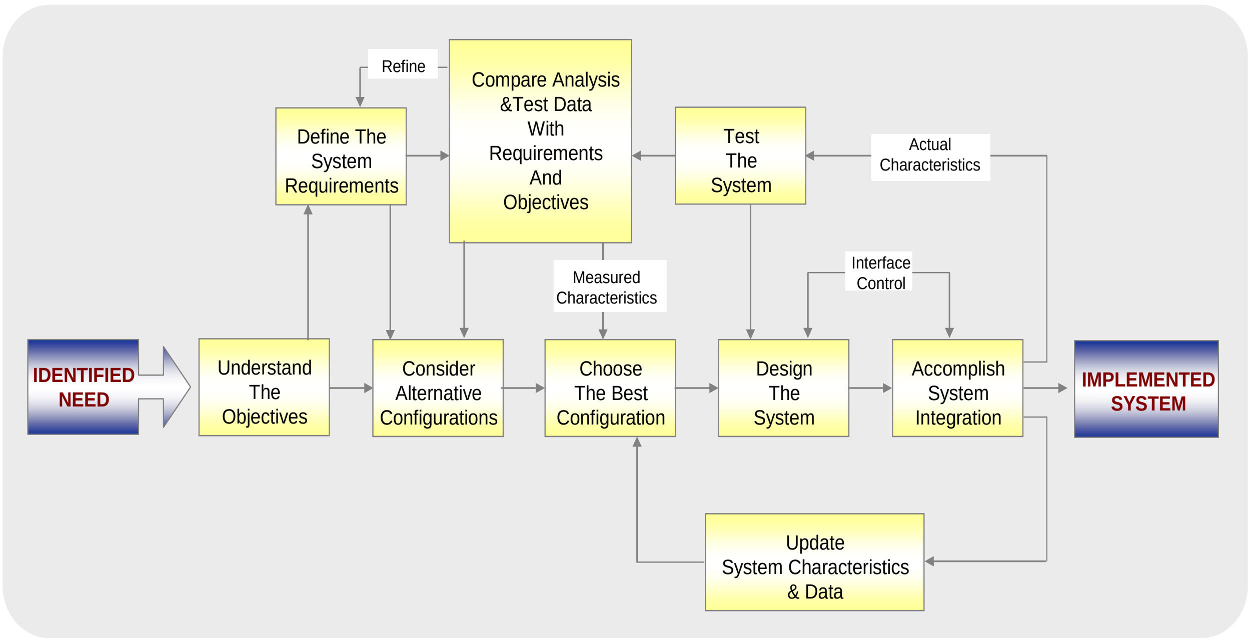
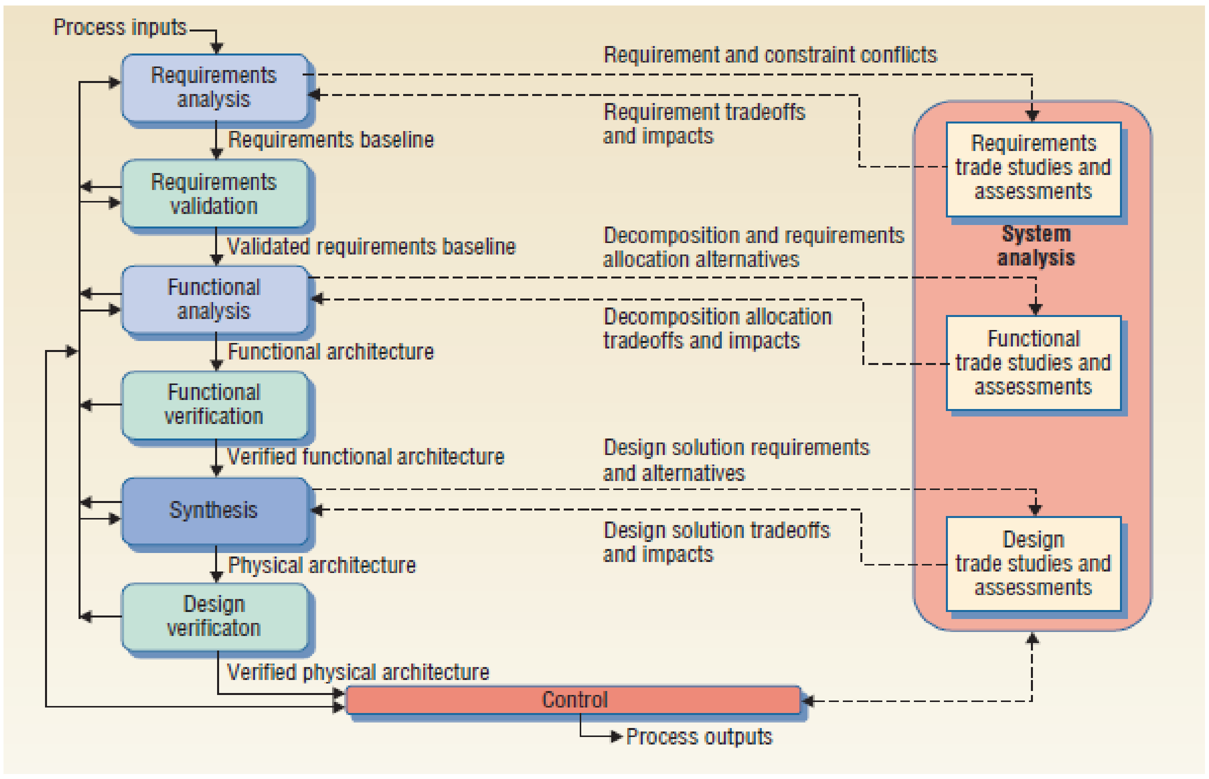

Refine
Compare Analysis &Test Data With Requirements And Objectives
Define The System Requirements
Test The System
Actual Characteristics
Measured Characteristics
Interface Control
IDENTIFIED NEED
Understand The Objectives
Consider Alternative Configurations
Choose The Best Configuration
Design The System
Accomplish System Integration
IMPLEMENTED SYSTEM
Update System Characteristics & Data

Figure 33. Typical SE Process Overview (Blanchard 2010).


Process inputs
Requirements analysis
Requirement and constraint conflicts
Requirement tradeoffs and impacts
Requirements baseline
Requirements validation
Requirements trade studies and assessments
System analysis
Validated requirements baseline
Decomposition and requirements allocation alternatives
Functional analysis
Decomposition allocation tradeoffs and impacts
Functional architecture
Functional trade studies and assessments
Functional verification
Verified functional architecture
Design solution requirements and alternatives
Synthesis
Design solution tradeoffs and impacts
Physical architecture
Design trade studies and assessments
Design verificaton
Verified physical architecture
Control
Process outputs

\

Figure 34. Recursive Process Representation (IEEE-1220-2005).

Significantly, the process is not linear; it typically is recursive in nature, where increasing detail is revealed as the requirements and functions are successively decomposed and verified as the design synthesis is progressed towards detailed implementation. Integration and verification is also recursive, where the system solution is progressively built up as a product hierarchy, and verified against the applicable requirements at the respective layers of decomposition and integration until the final implementation comprising the verified and validated completed system assembly is achieved (Mar 1992).

The key IEEE 1220 system structural concepts showing the basic system building block, the resultant product hierarchy produced, and the life cycle processes are illustrated in Figure 35 (Doran 2006). Here the hierarchical nature of the system is prominently shown.

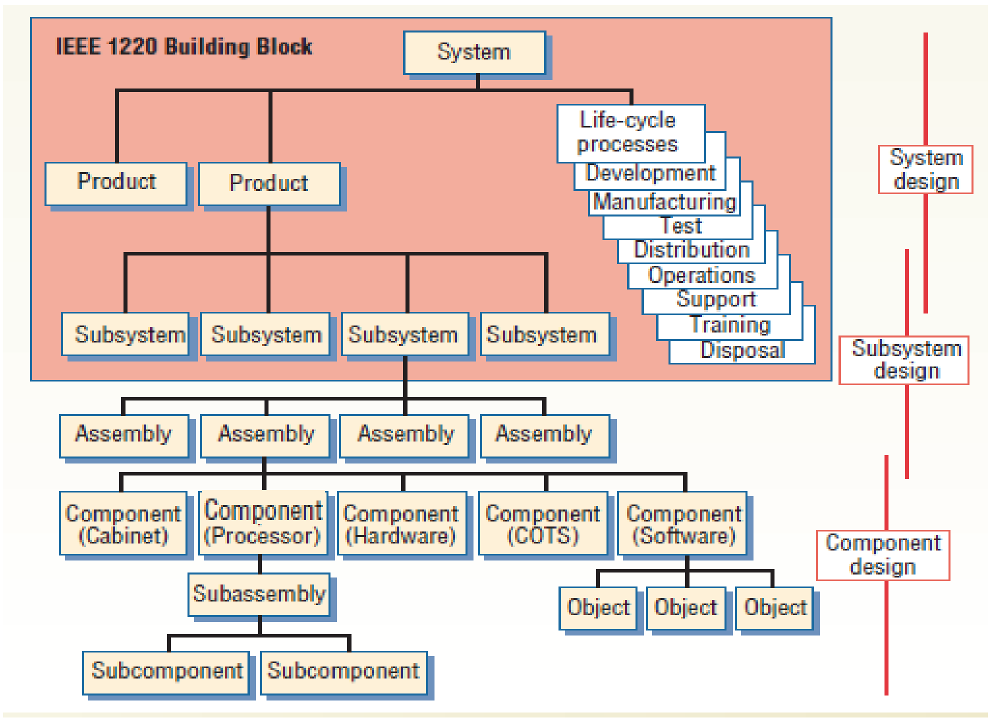


Figure 35. IEEE 1220-2005 Key System Structural Concepts.

The scope of consideration in respect of the problem space and the solution space of relevance to the organisation or enterprise as described in IEEE 1220-2005 is provided in Figure 36.

A commercial example of SE technical and management processes and reviews recommended to manage and control project activity is shown in Figures 37a, 37b, and 37c. The SQA 2000 Software Quality Assurance methodology©, developed by Coopers and Lybrand (Coopers & Lybrand 1991, 1995), provides a detailed list of activities, reviews, deliverables and audits as might typically be used to plan and manage the undertaking of complex software engineering development and production activity.

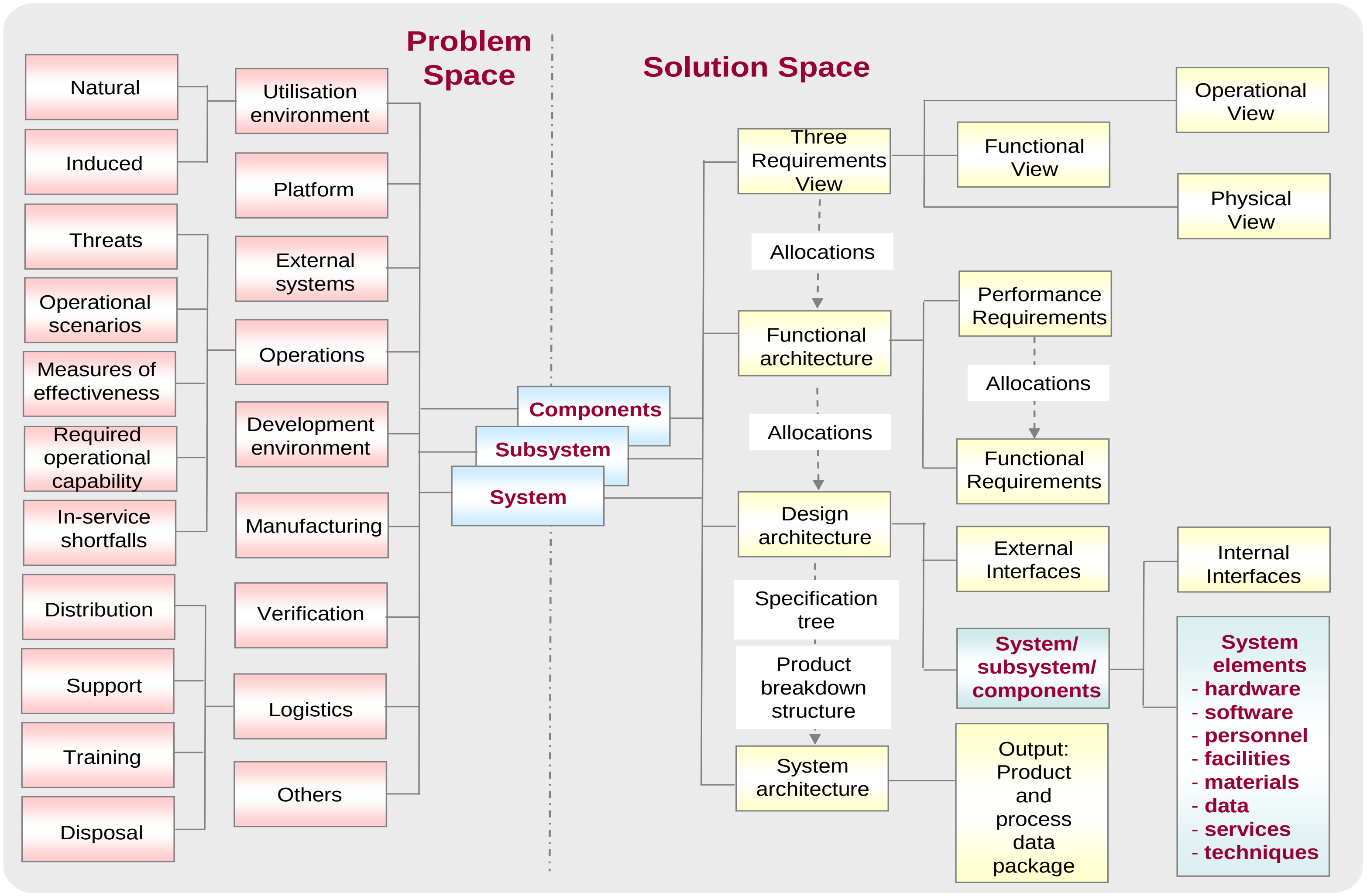
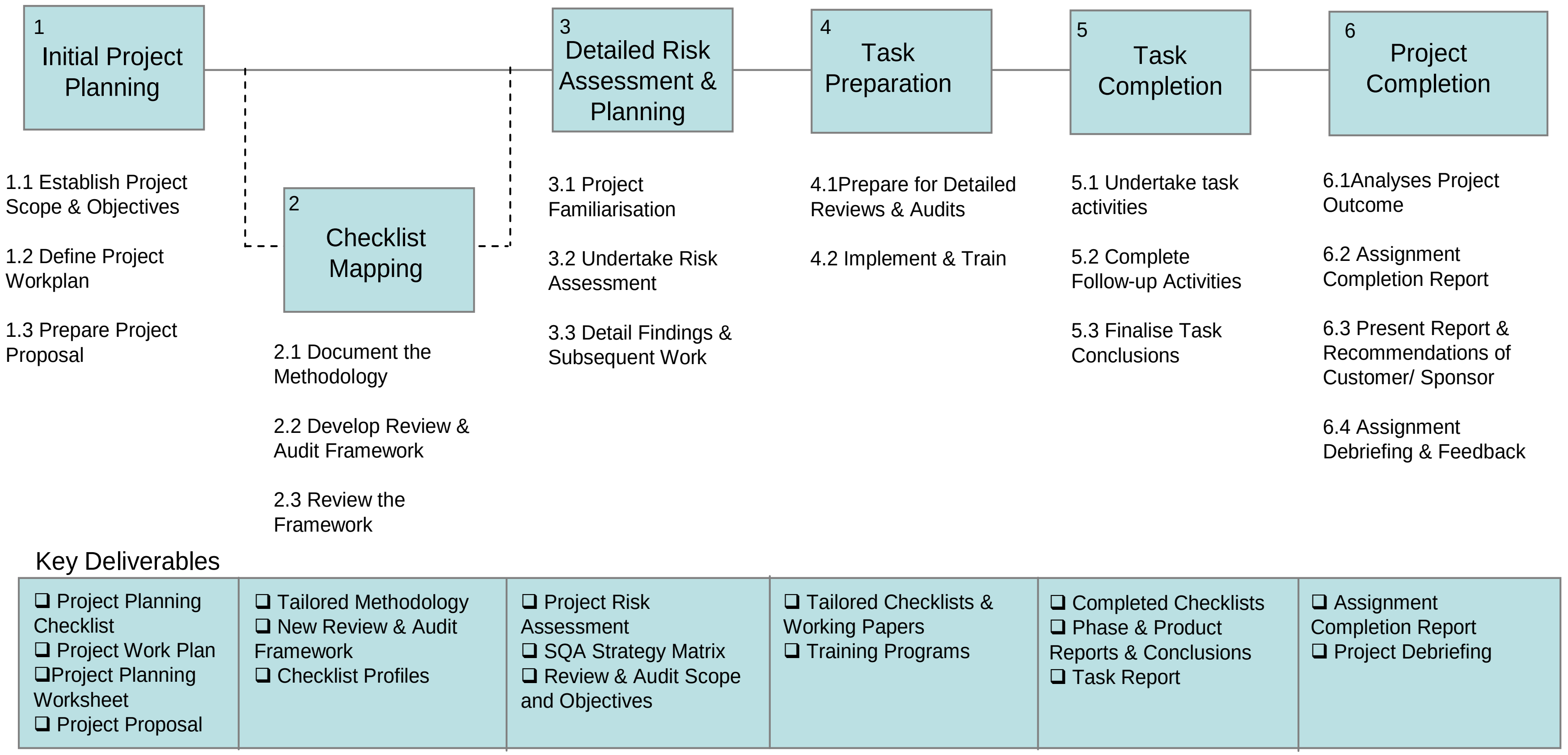


Figure 36. Key Problem and Solution Space Structuring Concepts (IEEE-1220-2005).


1 Initial Project Planning
1.1 Establish Project Scope & Objectives
1.2 Define Project Workplan
1.3 Prepare Project Proposal
2 Checklist Mapping
2.1 Document the Methodology
2.2 Develop Review & Audit Framework
2.3 Review the Framework
3 Detailed Risk Assessment & Planning
3.1 Project Familiarisation
3.2 Undertake Risk Assessment
3.3 Detail Findings & Subsequent Work
4 Task Preparation
4.1Prepare for Detailed Reviews & Audits
4.2 Implement & Train
5 Task Completion
5.1 Undertake task activities
5.2 Complete Follow-up Activities
5.3 Finalise Task Conclusions
6 Project Completion
6.1Analyses Project Outcome
6.2 Assignment Completion Report
6.3 Present Report & Recommendations of Customer/ Sponsor
6.4 Assignment Debriefing & Feedback
Key Deliverables

❑ Project Planning Checklist ❑ Project Work Plan ❑Project Planning Worksheet ❑ Project Proposal	❑ Tailored Methodology ❑ New Review & Audit Framework ❑ Checklist Profiles	❑ Project Risk Assessment ❑ SQA Strategy Matrix ❑ Review & Audit Scope and Objectives	❑ Tailored Checklists & Working Papers ❑ Training Programs	❑ Completed Checklists ❑ Phase & Product Reports & Conclusions ❑ Task Report	❑ Assignment Completion Report ❑ Project Debriefing

Figure 37a. Coopers & Lybrand SQA 2000 Methodology Overview©.

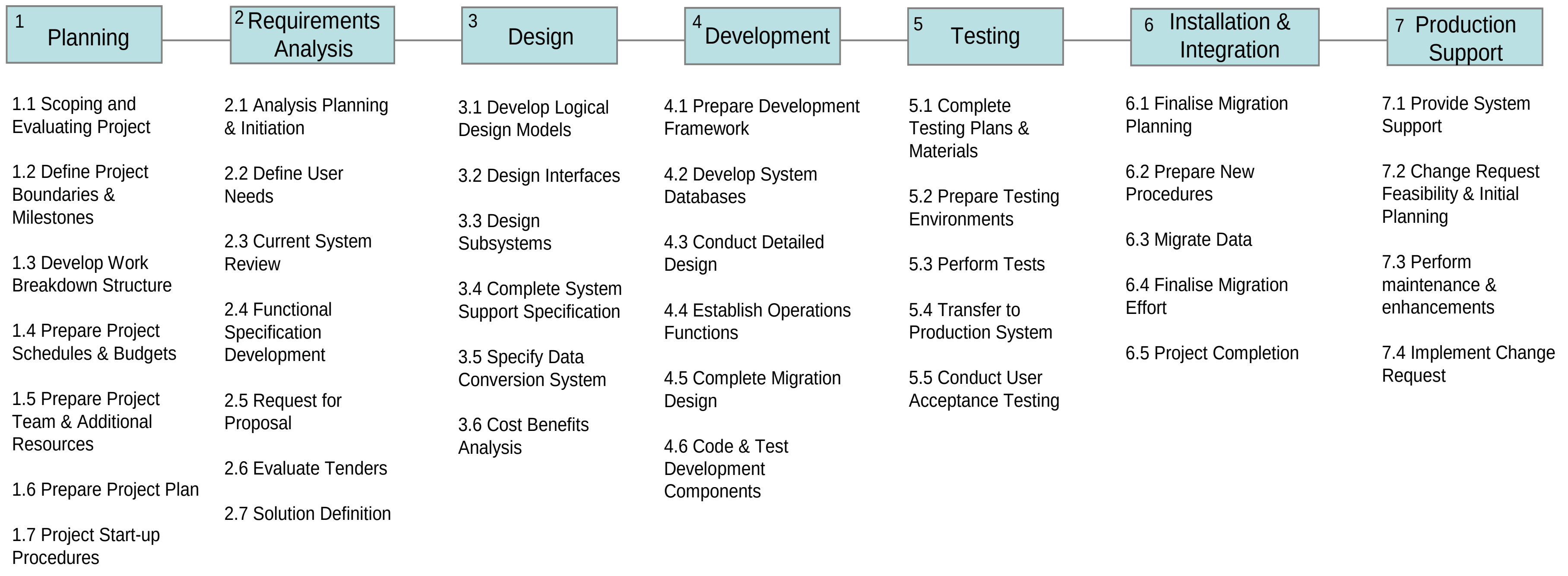
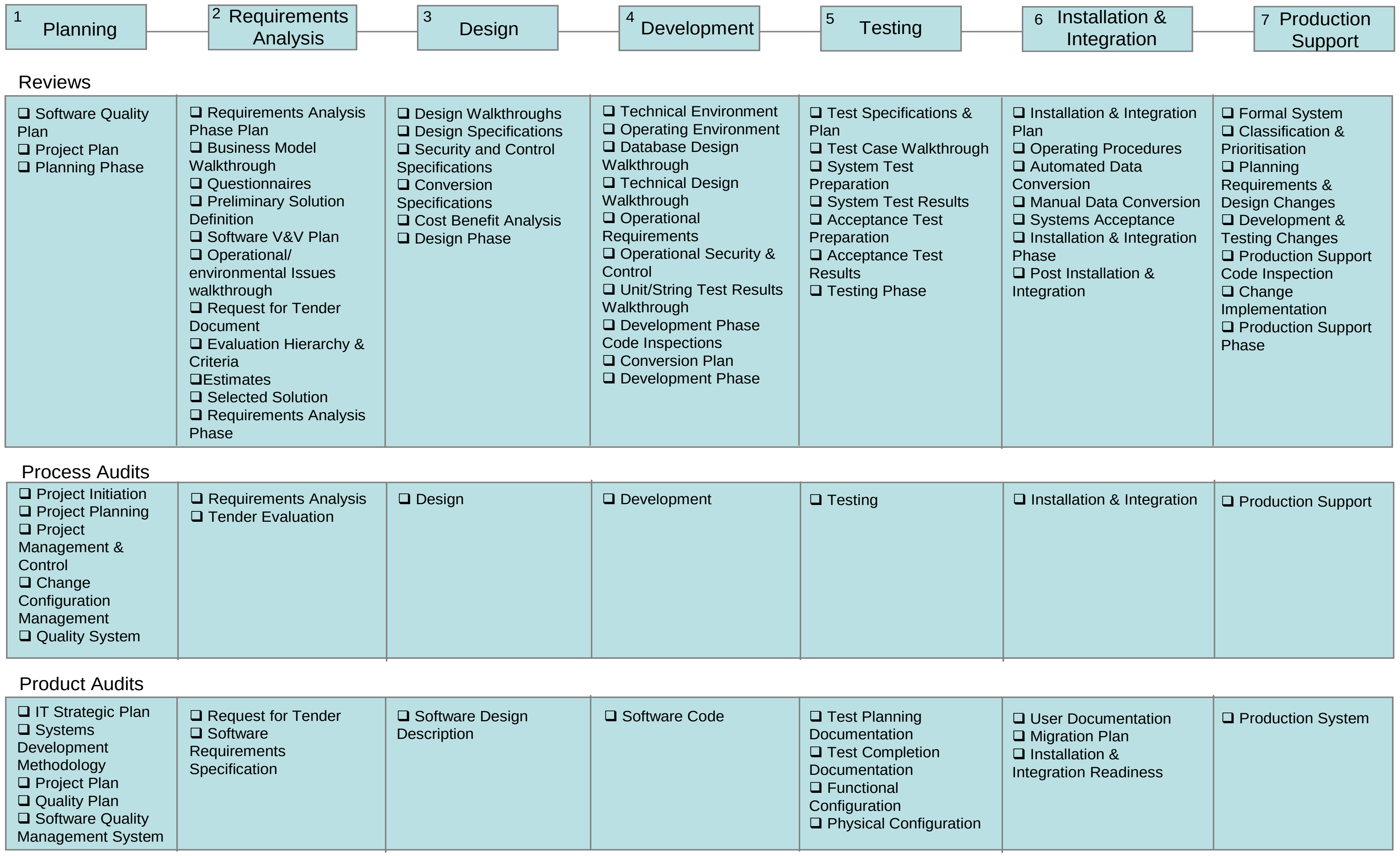

1 Planning
2 Requirements Analysis
3 Design
4 Development
5 Testing
6 Installation & Integration
7 Production Support
1.1 Scoping and Evaluating Project
1.2 Define Project Boundaries & Milestones
1.3 Develop Work Breakdown Structure
1.4 Prepare Project Schedules & Budgets
1.5 Prepare Project Team & Additional Resources
1.6 Prepare Project Plan
1.7 Project Start-up Procedures
2.1 Analysis Planning & Initiation
2.2 Define User Needs
2.3 Current System Review
2.4 Functional Specification Development
2.5 Request for Proposal
2.6 Evaluate Tenders
2.7 Solution Definition
3.1 Develop Logical Design Models
3.2 Design Interfaces
3.3 Design Subsystems
3.4 Complete System Support Specification
3.5 Specify Data Conversion System
3.6 Cost Benefits Analysis
4.1 Prepare Development Framework
4.2 Develop System Databases
4.3 Conduct Detailed Design
4.4 Establish Operations Functions
4.5 Complete Migration Design
4.6 Code & Test Development Components
5.1 Complete Testing Plans & Materials
5.2 Prepare Testing Environments
5.3 Perform Tests
5.4 Transfer to Production System
5.5 Conduct User Acceptance Testing
6.1 Finalise Migration Planning
6.2 Prepare New Procedures
6.3 Migrate Data
6.4 Finalise Migration Effort
6.5 Project Completion
7.1 Provide System Support
7.2 Change Request Feasibility & Initial Planning
7.3 Perform maintenance & enhancements
7.4 Implement Change Request

Figure 37b. Coopers & Lybrand SQA 2000 Systems Development Life Cycle Overview, Reviews and Audits Schedule© – Part 1 (adapted).



	1 Planning	2 Requirements Analysis	3 Design	4 Development	5 Testing	6 Installation & Integration	7 Production Support
Reviews	❑ Software Quality Plan ❑ Project Plan ❑ Planning Phase	❑ Requirements Analysis Phase Plan ❑ Business Model Walkthrough ❑ Questionnaires ❑ Preliminary Solution Definition ❑ Software V&V Plan ❑ Operational/ environmental Issues walkthrough ❑ Request for Tender Document ❑ Evaluation Hierarchy & Criteria ❑Estimates ❑ Selected Solution ❑ Requirements Analysis Phase	❑ Design Walkthroughs ❑ Design Specifications ❑ Security and Control Specifications ❑ Conversion Specifications ❑ Cost Benefit Analysis ❑ Design Phase	❑ Technical Environment ❑ Operating Environment ❑ Database Design Walkthrough ❑ Technical Design Walkthrough ❑ Operational Requirements ❑ Operational Security & Control ❑ Unit/String Test Results Walkthrough ❑ Development Phase Code Inspections ❑ Conversion Plan ❑ Development Phase	❑ Test Specifications & Plan ❑ Test Case Walkthrough ❑ System Test Preparation ❑ System Test Results ❑ Acceptance Test Preparation ❑ Acceptance Test Results ❑ Testing Phase	❑ Installation & Integration Plan ❑ Operating Procedures ❑ Automated Data Conversion ❑ Manual Data Conversion ❑ Systems Acceptance ❑ Installation & Integration Phase ❑ Post Installation & Integration	❑ Formal System ❑ Classification & Prioritisation ❑ Planning Requirements & Design Changes ❑ Development & Testing Changes ❑ Production Support Code Inspection ❑ Change Implementation ❑ Production Support Phase
Process Audits	❑ Project Initiation ❑ Project Planning ❑ Project Management & Control ❑ Change Configuration Management ❑ Quality System	❑ Requirements Analysis ❑ Tender Evaluation	❑ Design	❑ Development	❑ Testing	❑ Installation & Integration	❑ Production Support
Product Audits	❑ IT Strategic Plan ❑ Systems Development Methodology ❑ Project Plan ❑ Quality Plan ❑ Software Quality Management System	❑ Request for Tender ❑ Software Requirements Specification	❑ Software Design Description	❑ Software Code	❑ Test Planning Documentation ❑ Test Completion Documentation ❑ Functional Configuration ❑ Physical Configuration	❑ User Documentation ❑ Migration Plan ❑ Installation & Integration Readiness	❑ Production System

Figure 37c. Coopers & Lybrand SQA 2000 Systems Development Life Cycle Overview, Reviews and Audits Schedule© – Part 2 (adapted).

From a systems perspective, the approach in a SE context can therefore be summarised as:

1. Understand the problem in the broader context before attempting to solve it as a technical system (including metrics, priorities and constraints, to establish the feasible solution envelope).
2. Identify and rank (as far as reasonable), all possible technical solutions prior to selecting an answer (within the feasible solution envelope).
3. Look for hybrid solutions to add to the set of alternatives.
4. Select a technical solution, capture the supporting analysis, and formulate the subsequent problem at a lower level of decomposition or implement the solution (adapted from Mar 1997).

5.5 The System Life Cycle

The precise nature of the detailed engineering activities and the order in which they are performed are encapsulated in the concept of a life cycle model. As previously illustrated in Figure 38, the standard ISO/EIA 15288:2008 provide a simple, high-level summary of significant stages in the life of a product or system that is relevant to the organisations responsible for conceptualising, implementing, using, and supporting the product or system over its useful life.

Different life cycle models follow different process steps, decision points, and governance, with different consequences at each step in terms of cost, schedule, risk, and achieved progress in system implementation. Well known life cycle models include the Royce Waterfall Model, the Forsberg and Mooz V-Model, and the Boehm Spiral Model as shown in Figure 38(a), 38(b), and 38(c) respectively (Royce 1970), (Forsberg & Mooz 1992), (Boehm 1988).

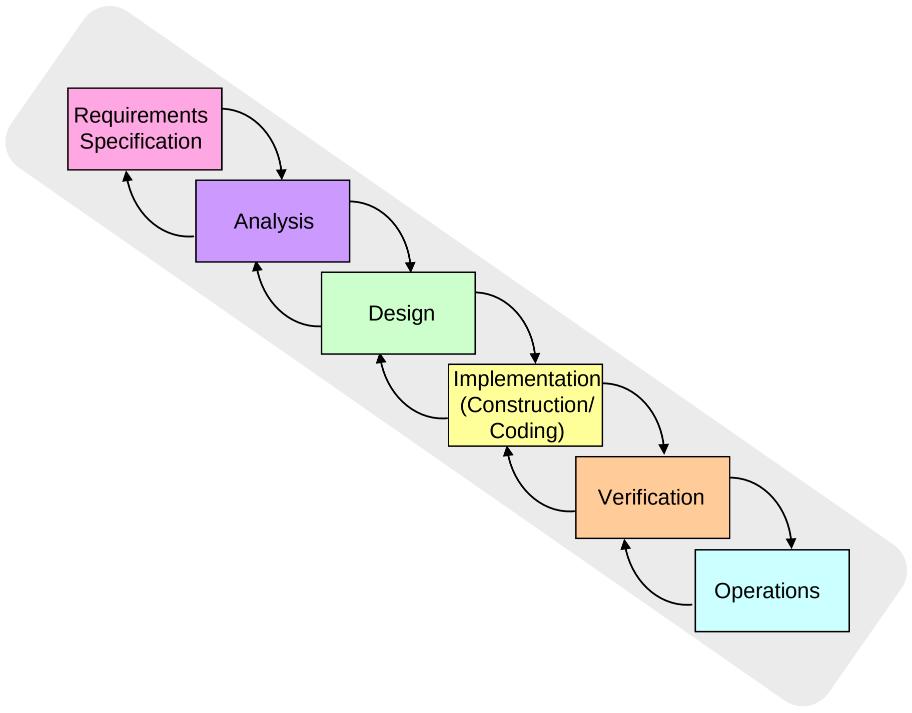

Requirements Specification
Analysis
Design
Implementation (Construction/ Coding)
Verification
Operations

Figure 38(a). SE Life Cycle Development Models – Waterfall Model (Royce 1970).

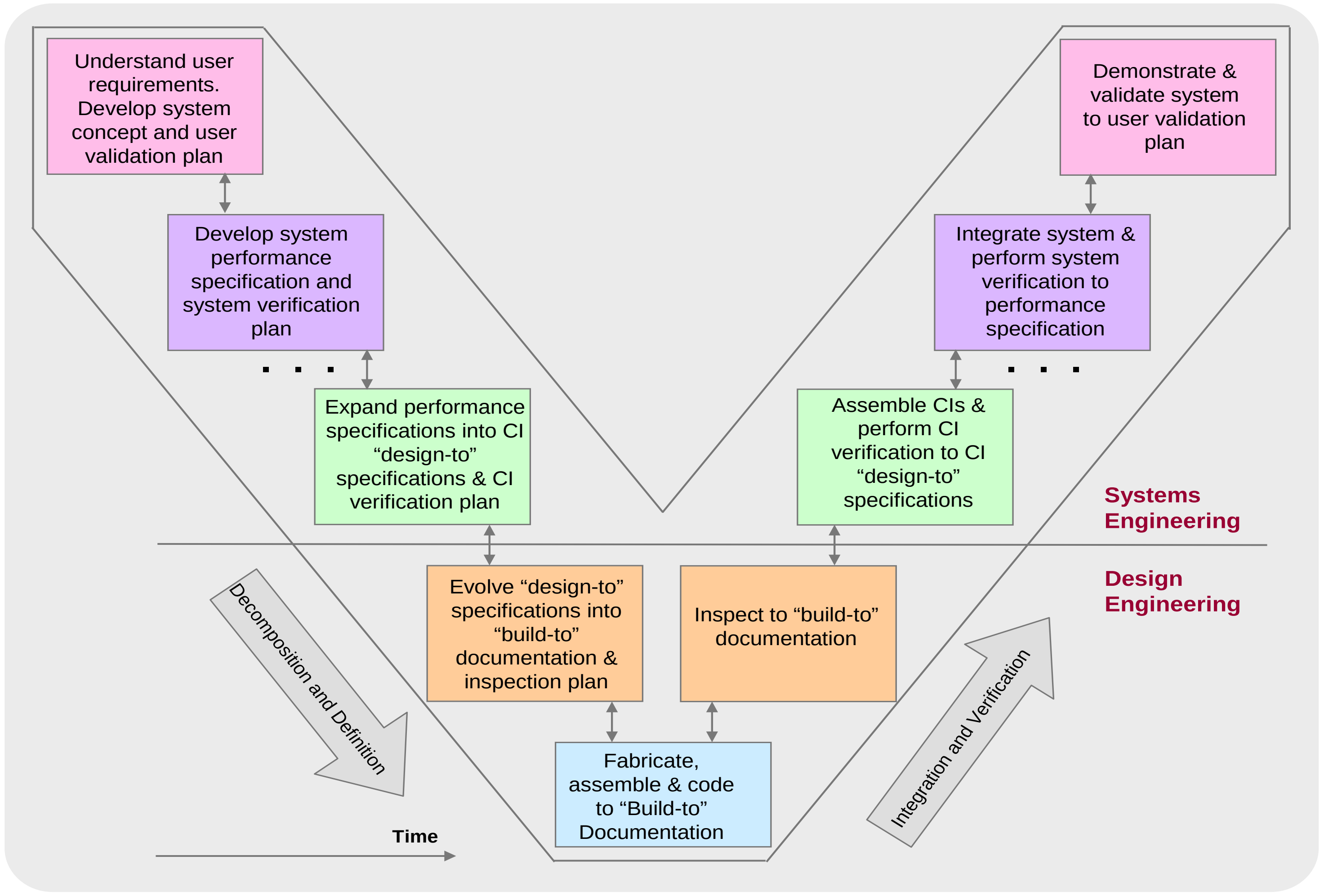

Understand user requirements. Develop system concept and user validation plan
Develop system performance specification and system verification plan
. . .
Expand performance specifications into CI "design-to" specifications & CI verification plan
Evolve "design-to" specifications into "build-to" documentation & inspection plan
Fabricate, assemble & code to "Build-to" Documentation
Inspect to "build-to" documentation
Assemble CIs & perform CI verification to CI "design-to" specifications
. . .
Integrate system & perform system verification to performance specification
Demonstrate & validate system to user validation plan
Systems Engineering
Design Engineering
Decomposition and Definition
Integration and Verification
Time

Figure 38(b). SE Life Cycle Development Models – Vee Model (Forsberg & Mooz 1992).

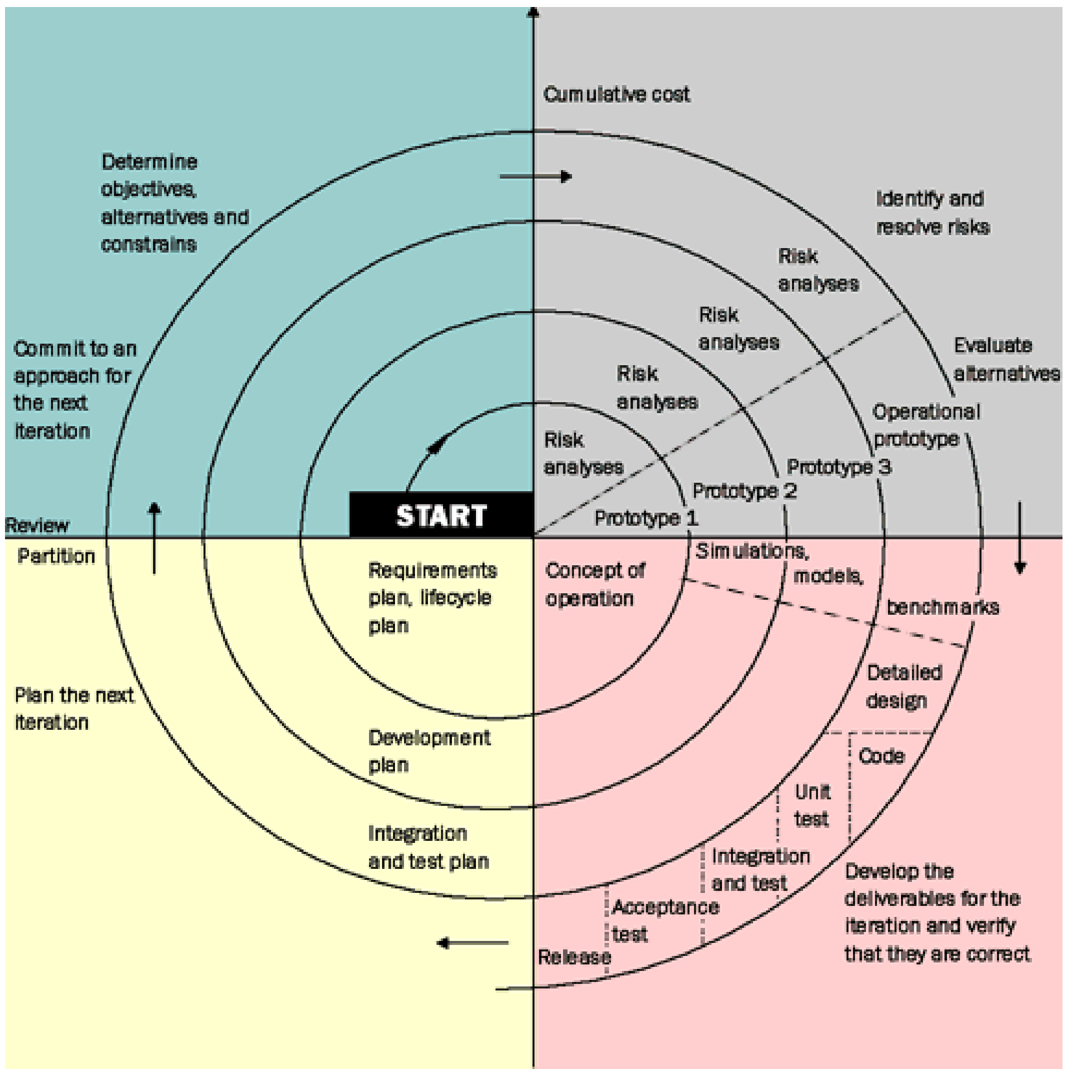


Figure 38(c). SE Life Cycle Development Model- Spiral Development (Boehm 1988).

Variations of these basic models, supporting both structured and iterative development, provide the basis for undertaking SE and SW development activity over the entire system life cycle. The high-level depiction of life cycle development activities as described in Figure 38 provide a meta-model for the major SE activities over the duration of the system life cycle, revealing the presence of significant decision-points or governance milestones. It therefore provides a framework for deliberate detailed planning and management of activity to attain the technical, schedule, and cost goals for the system[57].

The Vee-model as illustrated in Figure 38(b) is typically used within the context of an Australian Defence Project used to acquire Major Capital Equipment (MCE) for the Australian Defence Force (ADF).

[57] A detailed description of the life cycle methodology for SE is provided in Sage (1992).

5.6 The Significance of System Architecture in SE

The term architecture has been used on numerous occasions in this report, with many different facets; heavily shaped by SW engineering and IEEE 1471 influences.

The notions of systems architect and system architecture have emerged in recent times, displacing earlier notions of system designer and system design, even though they are not analogous. The term system architecture is widely used in modern SE, SW and EA parlance but not necessarily as widely understood.

(Hitchen 2007) offers that basic notions of system architecture arise around ideas of binding and coupling. Where a number of piece-parts all mutually interact, there is said to be tight functional binding (i.e. forms a cluster). Where such tight functional binds (i.e. clusters) are interconnected, the clusters are said to be loosely coupled.

A useful way of conceptualising system architecture suggested by Hitchens is to envisage the pattern formed by linking clusters of systems and subsystems. Since such clustering and linking can occur in many different ways, there can be many different patterns, all of which can ostensibly called system architectures. These are referred to as "viewpoints" of the system architecture in IEEE 1471 since all connections are simultaneously present. If the system is particularly large or complex, its system architecture (in its broadest sense) may be difficult to discern, and may change dynamically over time.

Hitchen suggests that in deliberately engineered systems, the system functional architecture emerges unaided as an intrinsic part of the system conception and design process; however, physical configurations are shaped by the constraints of the solution space. System implementation is therefore concerned with maintaining an effective functional architecture when mapping it onto a suitable physical configuration, without impeding functional interactions and functional behaviour. The system architecture indicates connectivity and potential cohesion since it shows the extent of interconnection, which is the fundamental aspect of any system – that all its piece-parts are interconnected and contribute towards the "whole".

In a SE context, the architecture is delineated by the connections between the piece-parts of the system (i.e. at the interfaces), since interruption of the connections could prevent the isolated piece-parts from contributing to the operation of the "whole", and could therefore impair performance of the "whole".

If there are multiple connections such that the severing of one connection did not impair piece-part interactions, then it may be possible for the piece-parts to continue to operate as a unified "whole". This illustrates the principle of redundancy to improve system resilience (Hitchens 2007).

5.7 Notions of System Hierarchy

The concept of system hierarchy is also very important in SE, manifesting as a physical architecture. The INCOSE SE Handbook describes a recursiveness within the life cycle process model that is applicable at each level of the system hierarchy (as previously shown in Figure 34) as the system implementation progresses through from initial system design to detailed subsystem, element, component and configuration item design, development and test.

Typical recursive interactions with stakeholders, and process inputs and outputs are illustrated in Figure 39, with the initial considerations providing a constant backdrop for consideration throughout the successive process steps and decision points.

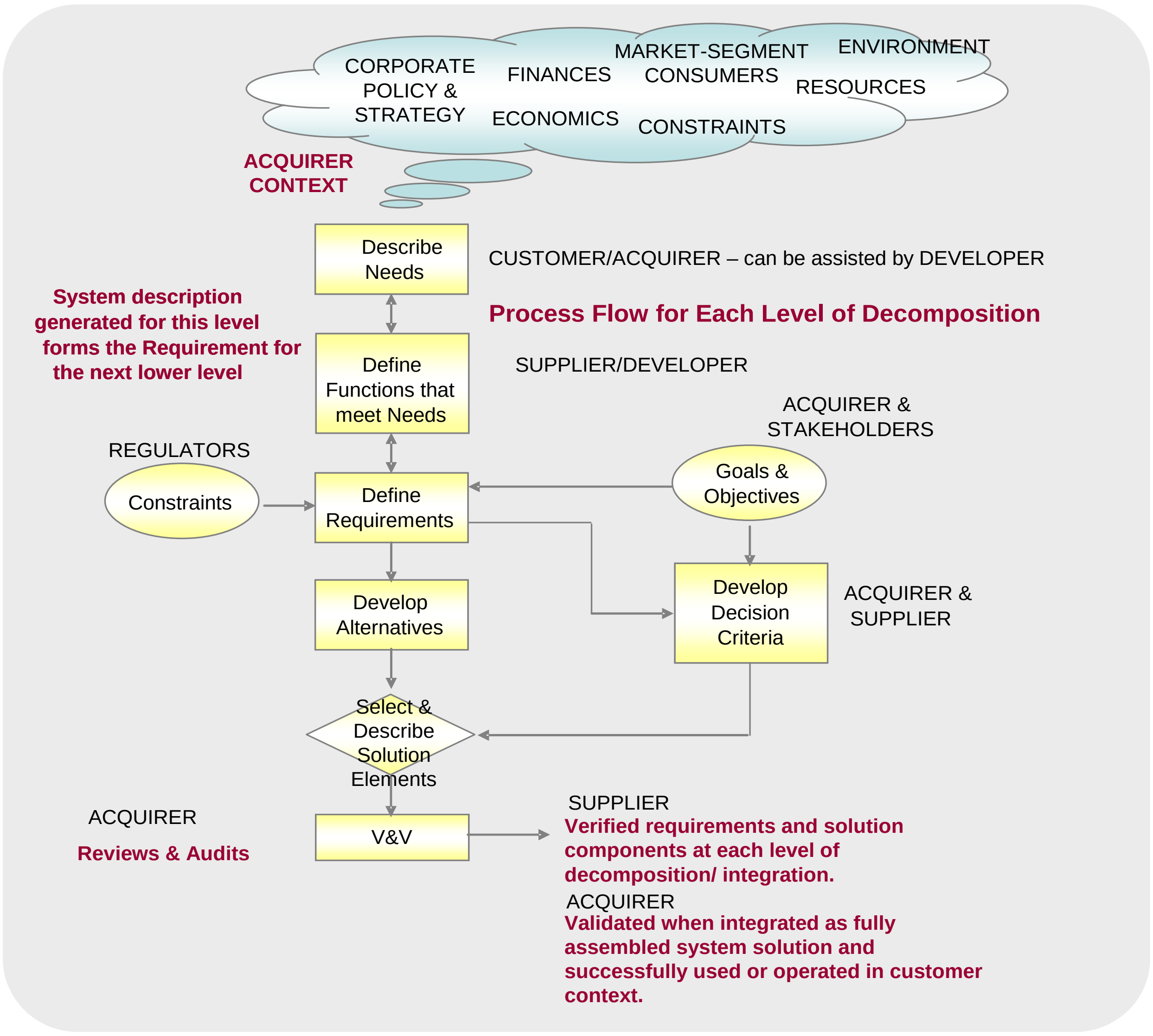

CORPORATE POLICY & STRATEGY
FINANCES
MARKET-SEGMENT CONSUMERS
ENVIRONMENT
RESOURCES
ECONOMICS
CONSTRAINTS
ACQUIRER CONTEXT
Describe Needs
CUSTOMER/ACQUIRER – can be assisted by DEVELOPER
Process Flow for Each Level of Decomposition
System description generated for this level forms the Requirement for the next lower level
Define Functions that meet Needs
SUPPLIER/DEVELOPER
ACQUIRER & STAKEHOLDERS
REGULATORS
Constraints
Define Requirements
Goals & Objectives
Develop Alternatives
Develop Decision Criteria
ACQUIRER & SUPPLIER
Select & Describe Solution Elements
ACQUIRER
Reviews & Audits
V&V
SUPPLIER
Verified requirements and solution components at each level of decomposition/ integration.
ACQUIRER
Validated when integrated as fully assembled system solution and successfully used or operated in customer context.

Figure 39. Context for Typical Process Flow for Each Level of Decomposition. (Blanchard & Fabrycky 1998).

The systems engineering design process includes defining all of the system's requirements and then bundling them by segmenting and refining into successive specifications for each of the system's segments, elements, components, and Configuration Items (CI) respectively.

The development process in SE can therefore be represented as a purposeful decomposition (or design) process followed by a re-composition (or integration process) (Buede 2000). The progression of the design occurs as a decomposition of requirements and the operational architecture, and performing physical to functional allocations at successive stages of decomposition as shown in Figure 40.

Trade-off studies carried out at each level of decomposition during the design phases are used to inform of the benefits and pitfalls of design alternatives at each level. Implementation decisions at each level then inform the set of requirements to direct implementation of the next lower level in the hierarchy until the design synthesis is completed.

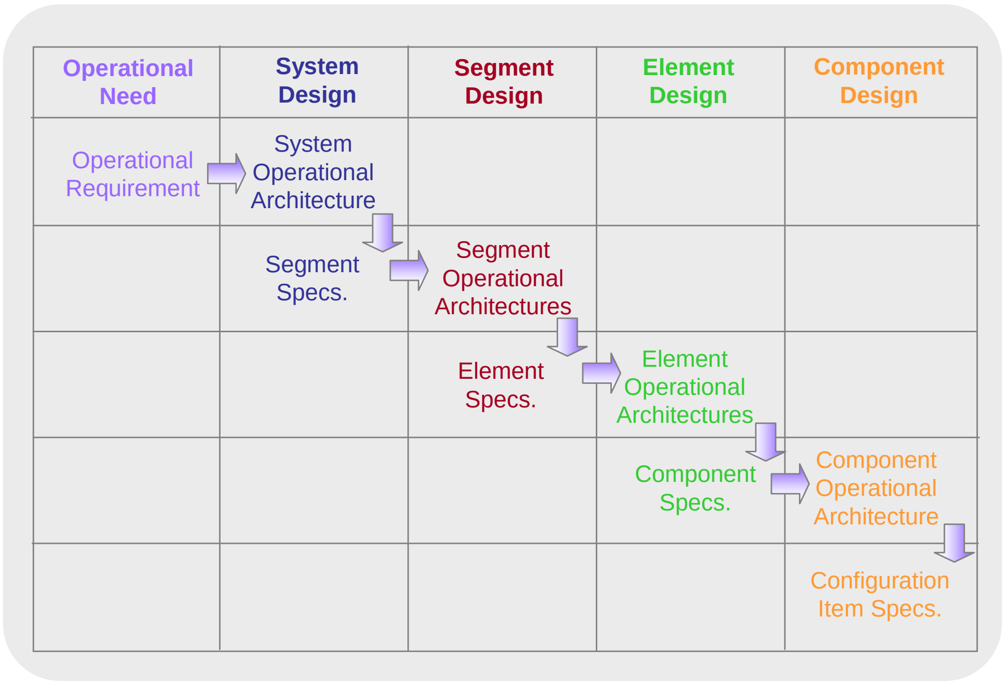

Operational Need
System Design
Segment Design
Element Design
Component Design
Operational Requirement
System Operational Architecture
Segment Specs.
Segment Operational Architectures
Element Specs.
Element Operational Architectures
Component Specs.
Component Operational Architecture
Configuration Item Specs.

Figure 40. Design Decomposition of Architectures and Specifications (Buede 2000)

The system hierarchy is often expressed in the form of the resultant design architecture, where the design architecture is the structure of the components (i.e. solution system elements) in the system in terms of their interface boundaries, arising as a purposeful system partitioning into components, or deliberate assignment of components to the designated system (Hoban & Lawbaugh 1993b). This allows differentiation between those components over which the acquirer has some semblance of control, as opposed to those external components that the system has no control over.

Thus three distinct architectures are derived from the initial system's operating concept or operational requirement (functional, physical and operational) as part of this decomposition as shown in Figure 41 (Buede 2000).

The functional architecture defines what the system must do, i.e. the system's functions and the data that flows between them. The physical architecture represents the partitioning of the physical resources available to perform the system functions. The operational architecture is the mapping of functions to resources.

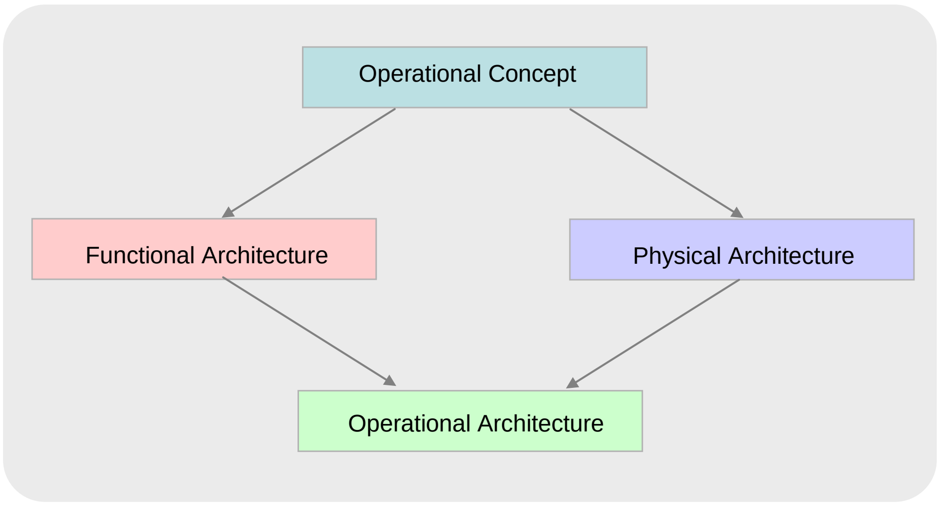


Figure 41. Architecture Development in the Engineering of a System (Buede 2000)

A life-cycle physical architecture is illustrated in Figure 42, showing both the physical architecture of the system solution as well as the physical architecture supporting the systems engineering activity (Buede 2000).

For many systems, a total of five distinct models are critical for capturing the totality of the system: environment, data (or information), process, behaviour and implementation.

- The environment model reflects the system boundary, the operational concept, and the objectives of the system performance;
- The data or information model captures the relationships between among the data elements that cross the system's boundary, as well those that are internal to the system;
- The process model captures the functionality of the system and is used to describe the functional architecture;
- The behaviour model reflects the control structures in which the systems functions are embedded;
- The implementation model reflects the overlay of processes and behaviours on the physical architecture; and
- The operational model reflects the operational architecture (Buede 2000).

These five models need to be integrated to properly define the three architectures of Figure 41.

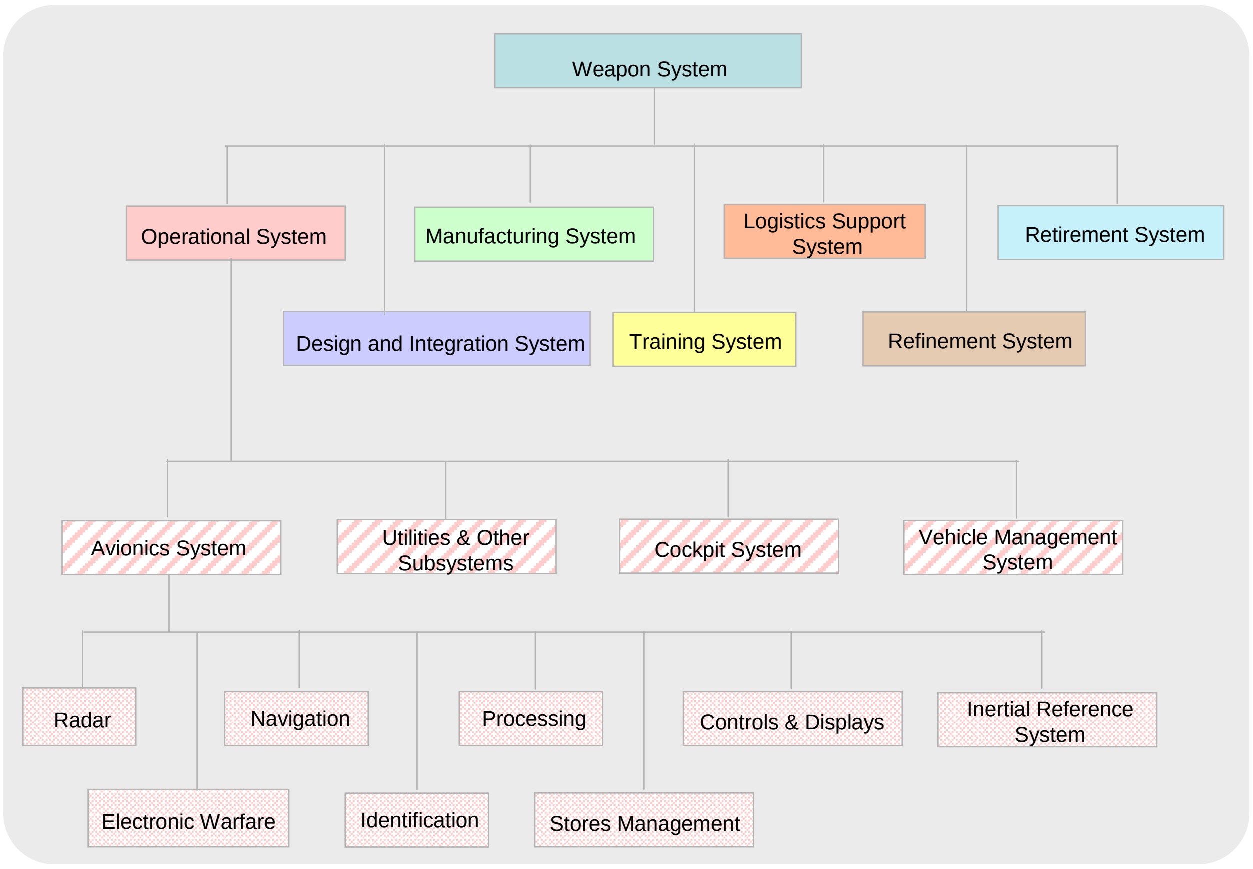


Figure 42. Life-cycle Physical Architecture – Weapons Systems Example (adapted from Buede 2000)

As the design synthesis progresses to lower levels of the hierarchy, and design integration and verification progresses up the hierarchy towards the highest level, additional insights can reveal emergent properties which may detrimentally affect the ability to achieve the desired outcomes. In fact, it may be found belatedly that the desired outcome may not achievable as initially conceived with the selected implementation.

This may necessitate revisiting the requirements and constraints at each level in the hierarchy, including the originating need, to reconsider trade-offs in light of the additional insights. ***This notion of feedback and iteration to refine the system concept and system implementation to optimise the outcome in terms of capability, cost, and schedule, is fundamental to SE.***

5.8 System of Interest and Systems of Systems

Your system is someone else's subsystem and someone else's system is your subsystem" (Mar 1992).

The term "system of interest" is also used interchangeably with the term "system", where this is defined in ISO/IEC 15288:2008 as "the system whose life cycle is under consideration in the context of the standard". This definition is also used in the INCOSE SE Handbook

(INCOSE 2012).

This infers for the purpose of applying the SE process, it is also important to:

- Identify the boundary of the "system of interest",
- Identify the inputs and outputs to the "system of interest ", and
- Identify the functions and performance of the "system of interest" to convert inputs to outputs to meet the originating requirements (Mar 1997).

External systems are those entities outside the boundary of the "system of interest" which can interact with the "system of interest" through its external interfaces. All the inputs and outputs from the "system of interest" flow through its external interfaces to these external systems, many of which may be legacy (existing) systems. It is therefore important to properly articulate the system context, which identifies those external entities that can impact on the "system of interest", but cannot be impacted by the "system of interest", as shown in Figure 43 (adapted from Buede 2000).

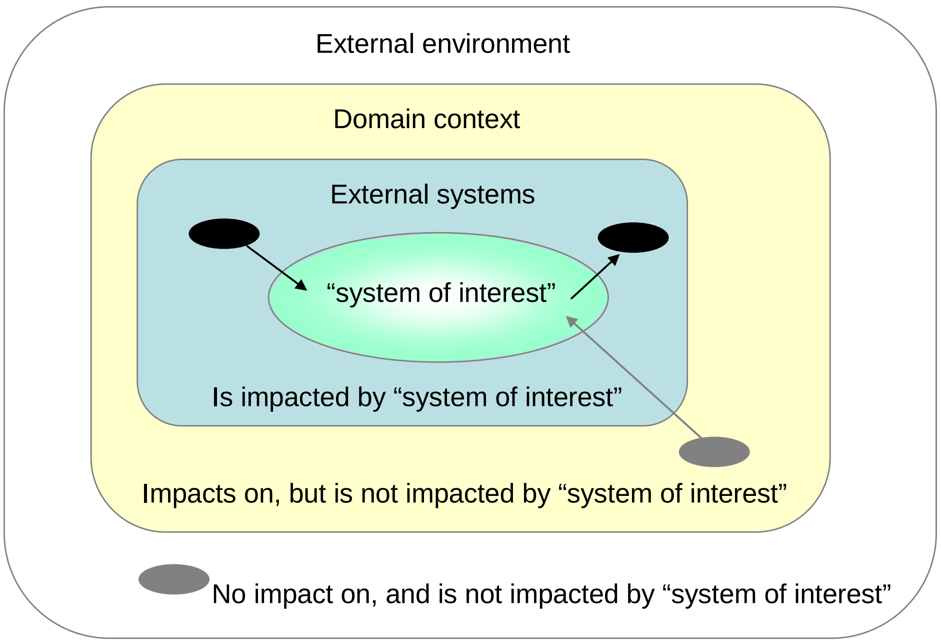

External environment
Domain context
External systems
"system of interest"
Is impacted by "system of interest"
Impacts on, but is not impacted by "system of interest"
No impact on, and is not impacted by "system of interest"

Figure 43. Depiction of "System of Interest", External Systems and Context

Failure to define the "system of interest" appropriately will result in considerable confusion and the misapplication of SE precepts in the wrong context (Mar 1997). The concepts of system identity and system boundary, thus exposing the respective external system interfaces, are therefore essential for the articulation of a system in the SE context. Similarly, concepts of identity and boundary for the respective system elements (i.e. components) are also essential for exposing the appropriate internal interfaces.

An example of a high-level context diagram showing the "system of interest" interfacing to multiple legacy systems which may impact on the design of a car is shown in the following diagram in Figure 44.

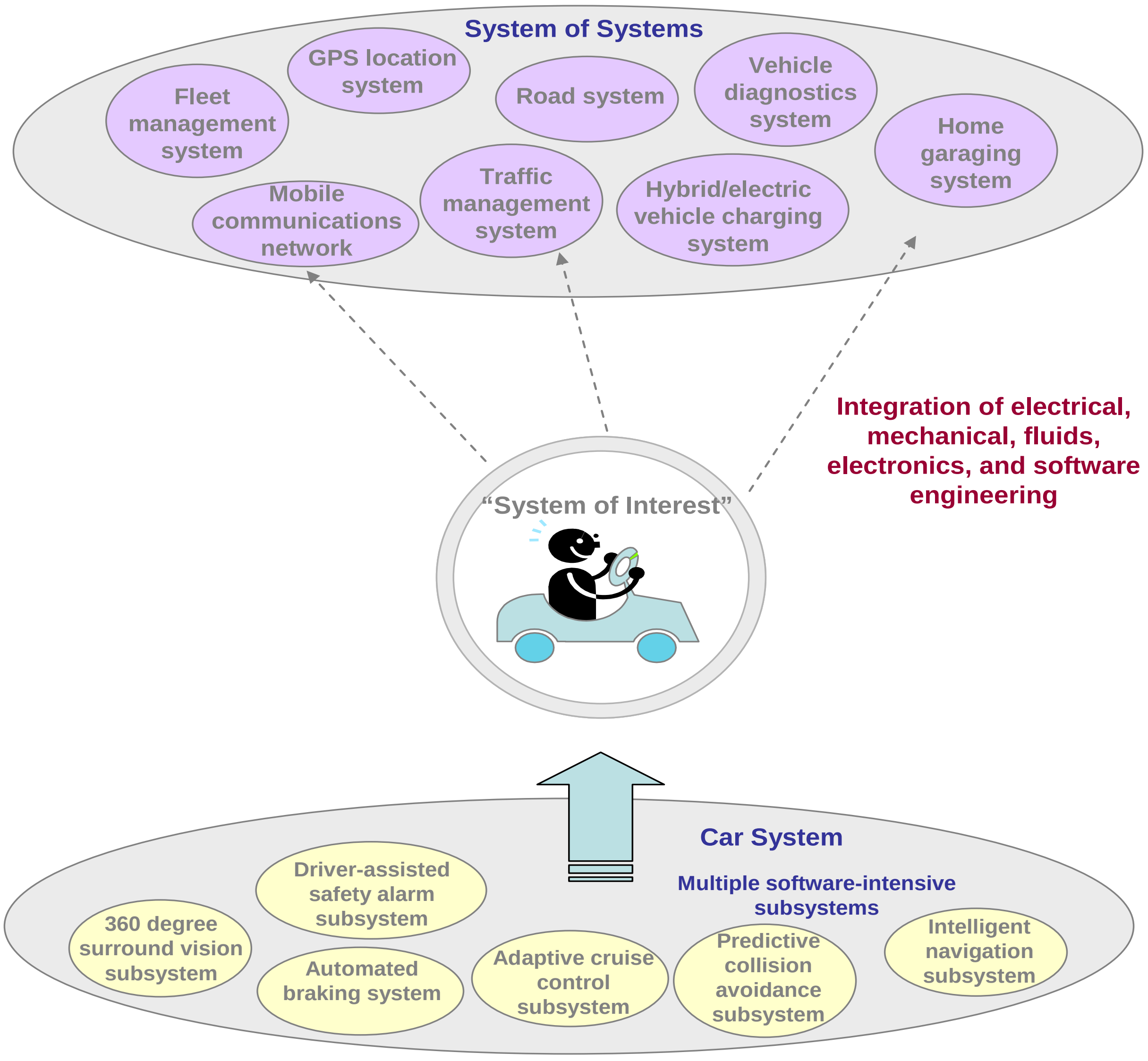

System of Systems
GPS location system
Fleet management system
Road system
Vehicle diagnostics system
Home garaging system
Mobile communications network
Traffic management system
Hybrid/electric vehicle charging system
Integration of electrical, mechanical, fluids, electronics, and software engineering
"System of Interest"
Car System
Multiple software-intensive subsystems
Driver-assisted safety alarm subsystem
360 degree surround vision subsystem
Automated braking system
Adaptive cruise control subsystem
Predictive collision avoidance subsystem
Intelligent navigation subsystem

Figure 44. Example of System of Interest in Relation to Car Design

The INCOSE Handbook introduces the term "Systems of Systems" (SoS) to differentiate those "systems of interest" whose system elements are themselves systems – typically associated with large scale inter-disciplinary problems with multiple, heterogeneous, distributed systems. SoS are systems in their own right in that they perform functions and have common purpose that do not reside in any component systems, and properties that cannot be localised to any component system (Maier 1998).

Rather than designing from top down, as suggested in the simple Vee life cycle model for a system, SoS are typically formed by bringing together specific individual systems such that the emergent properties of the collective systems meet the higher level requirements of the SoS. Systems engineering carried out at the SoS level is sometimes referred to as SOS engineering (SOSE).

Originally developed by Maier in 1998, the INCOSE SE Handbook offers the following characteristics and challenges inherent in SoS:

1. System elements (i.e. component systems) operate independently – if the SoS is disassembled into component systems, each system within a SoS has the ability to be independently operational in its own right, (and hence may be subject to differing requirements, constraints, and priorities);
2. System elements have different life cycles – older system elements may be scheduled for disposal before newer system elements are deployed;
3. The initial requirements are likely to be ambiguous – the SoS requirements may be no more explicit than the system element requirements. Thus requirements for a SoS mature as the system elements mature;
4. Complexity is a major issue – as system elements are added, the complexity of the system interactions grows in a non-linear manner;
5. Management can overshadow engineering – System elements normally operate independently in their own right, and are managed independently. Since each system element has its own product/project office, the coordination of requirements, budget constraints, schedules, shared resources, interfaces and system upgrades adds further complexity to the development of a SoS;
6. Fuzzy boundaries can cause confusion – unless the scope of the SoS is specifically defined and controlled, and the the boundaries of the systems elements are managed, there is no control of the definition of the external boundaries;
7. SoS engineering is never finished – the SoS does not appear fully formed. Its existence is evolving with functions being added, modified, or deleted over time. This means product and project management activity must continue to account for changes in the various system element life cycles (e.g. introduction of new technologies or replacement of obsolete system elements). (INCOSE 2012).

Inherent in the definition offered by Maier in 1998, SoS component systems are also geographically distributed, and therefore can only readily exchange information and not substantial quantities of mass or energy. This latter characteristic is particularly significant in terms of its relationship to enterprise architecture practice, which primarily focuses on information exchange (Hue 2011).

Since a SoS is a conceptual entity rather than a physical entity, there is substantial discretion in selection of SoS boundaries in an organisation or enterprise as shown in Figure 45, depending on the binding and coupling criteria elected, but the notion of system hierarchy is still preserved (Hue 2011).

The usefulness of this conceptual construct is its ability to abstract away from overly complex detail while conveying key themes and linkages. The level of abstraction at which a problem is to be solved will determine the boundary of the system of interest. Thus, the interactions of the system of interest with its wider system context, immediate environment and wider environment need to be identified and understood.

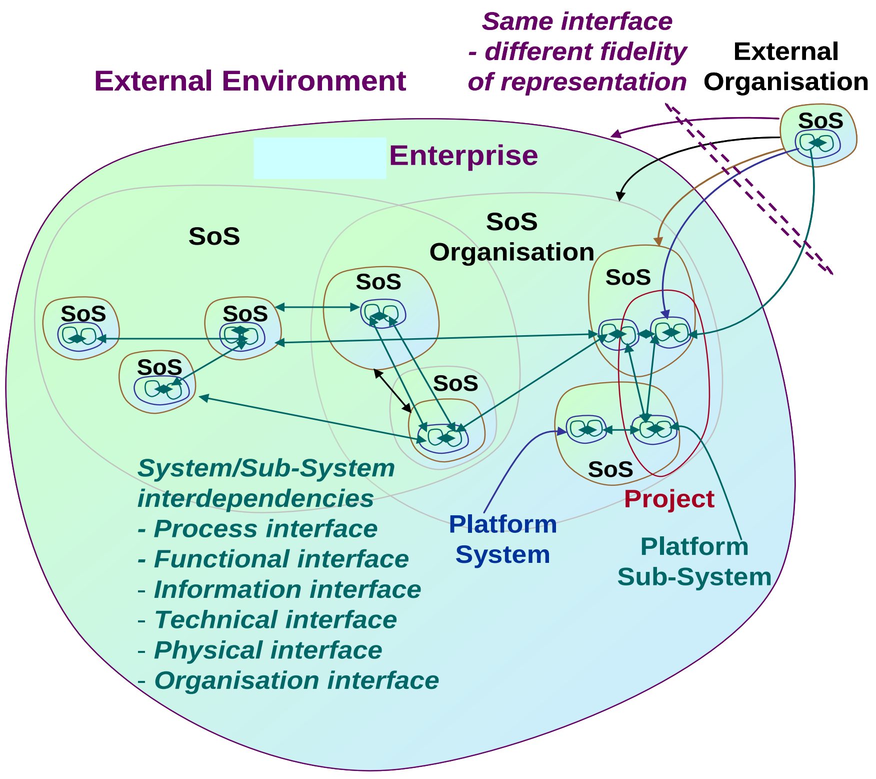


Figure 45. SoS Representation within an Enterprise.

The perception and definition of a particular system or SoS, its architecture, and its components depends on an observer's interests and responsibilities. Figure 46 illustrates important principles including:

- The importance of defined boundaries that encapsulate meaningful need and practical solution;
- The hierarchical perception of system physical structure;
- That an entity at any level in a hierarchical structure can be viewed as a system;
- That a system comprises a fully integrated, defined set of subordinate systems, (i.e. components);
- The interactions between components give rise to characteristic properties at a system's boundary;
- That humans can be viewed as users external to a system (e.g. car driver and braking system), and/or as components within a system (e.g. car driver and car)
- That a system may be viewed as both a product (looking inwards at its boundary) and a set of services (when viewed from outside of its boundary) (ISO/IEC 15228).

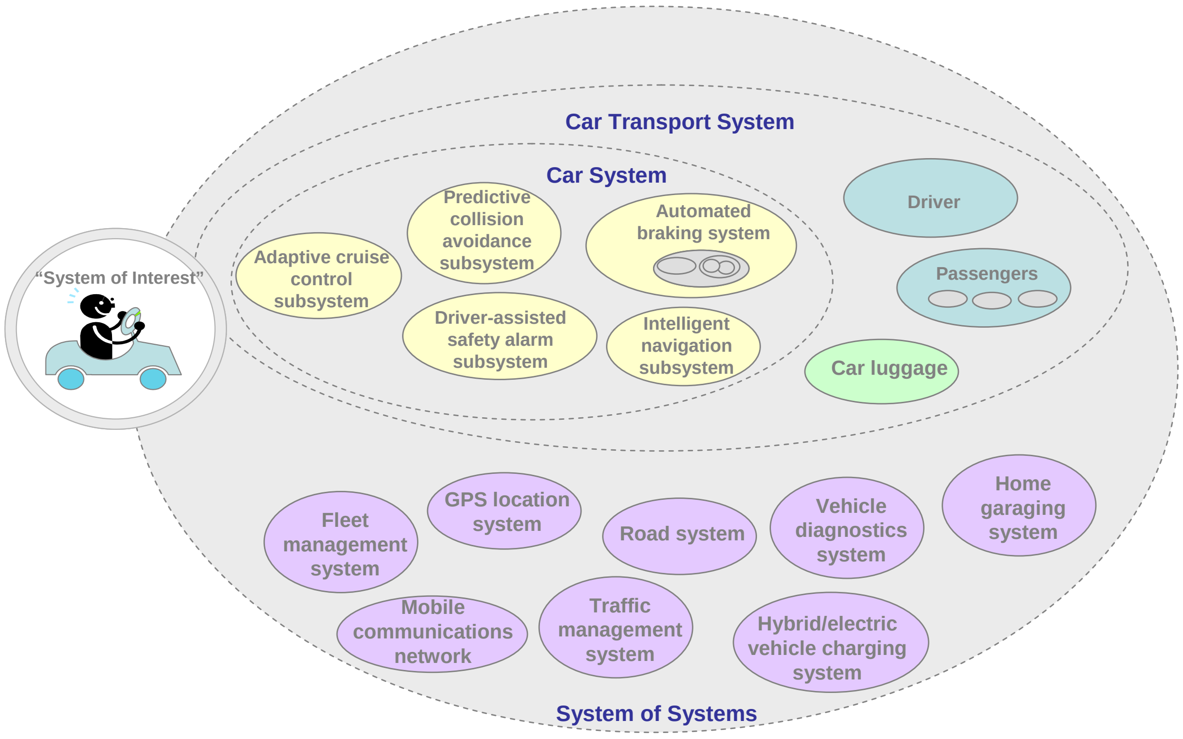

Car Transport System
Car System
"System of Interest"
Adaptive cruise control subsystem
Predictive collision avoidance subsystem
Automated braking system
Driver-assisted safety alarm subsystem
Intelligent navigation subsystem
Driver
Passengers
Car luggage
Fleet management system
GPS location system
Road system
Vehicle diagnostics system
Home garaging system
Mobile communications network
Traffic management system
Hybrid/electric vehicle charging system
System of Systems

Figure 46. Example System View of a Car in SoS Context

To ensure all system implications are exposed, a complete and holistic system view must be taken in full context (Sparks 2011). Since the behaviour of a system is dependent on external interactions and unforeseen circumstances, it behoves to explore all attributes of interest, including the effect of failures on system behaviour, at the appropriate level of abstraction.

Key systems of purpose can be component systems in multiple SoS that have complementary functionality. The greater the degree of integration required, the greater the component system interdependencies, and thus the greater the number of acquisition interdependencies and other socio-technical interdependencies to be acknowledged and managed.

The "system of interest" can therefore be significantly larger than the individual component system being engineered, so that other relevant socio-technical factors including organisation, process, personnel, economic and political influences are also accounted for in informing engineering decisions. Thus each key component system in a SoS can no longer be managed solely in the context of its own engineering environment, but needs to be examined in a much broader context with respect to the larger encompassing SoSs and their external environments (Hue 2011).

Additional and ongoing SoS risk management activities, including V&V at the SoS level, are therefore essential to ensure the desired emergent properties of the SoS are achieved and sustained, and that no unexpected or detrimental properties emerge, nor required properties lost, as individual component system elements progress through their respective life cycle stages.

Broader integration of SoS (referred to as SoSI) within their external environments falls more within the realms of the social sciences and management science rather than engineering science, and is therefore out of scope of an engineering-centric process. However, SoSI still needs to be appropriately managed.

5.9 Systems Integration and SoS Integration

In the engineering context, system integration is the bringing together of the detailed elements of the overall system design into one system through a process of assembly and testing, and ensuring the elements function together as a system meeting the needs of the stakeholders with the expected features and attributes, as shown in Figure 47 (Buede 2000), (Shameih 2011).

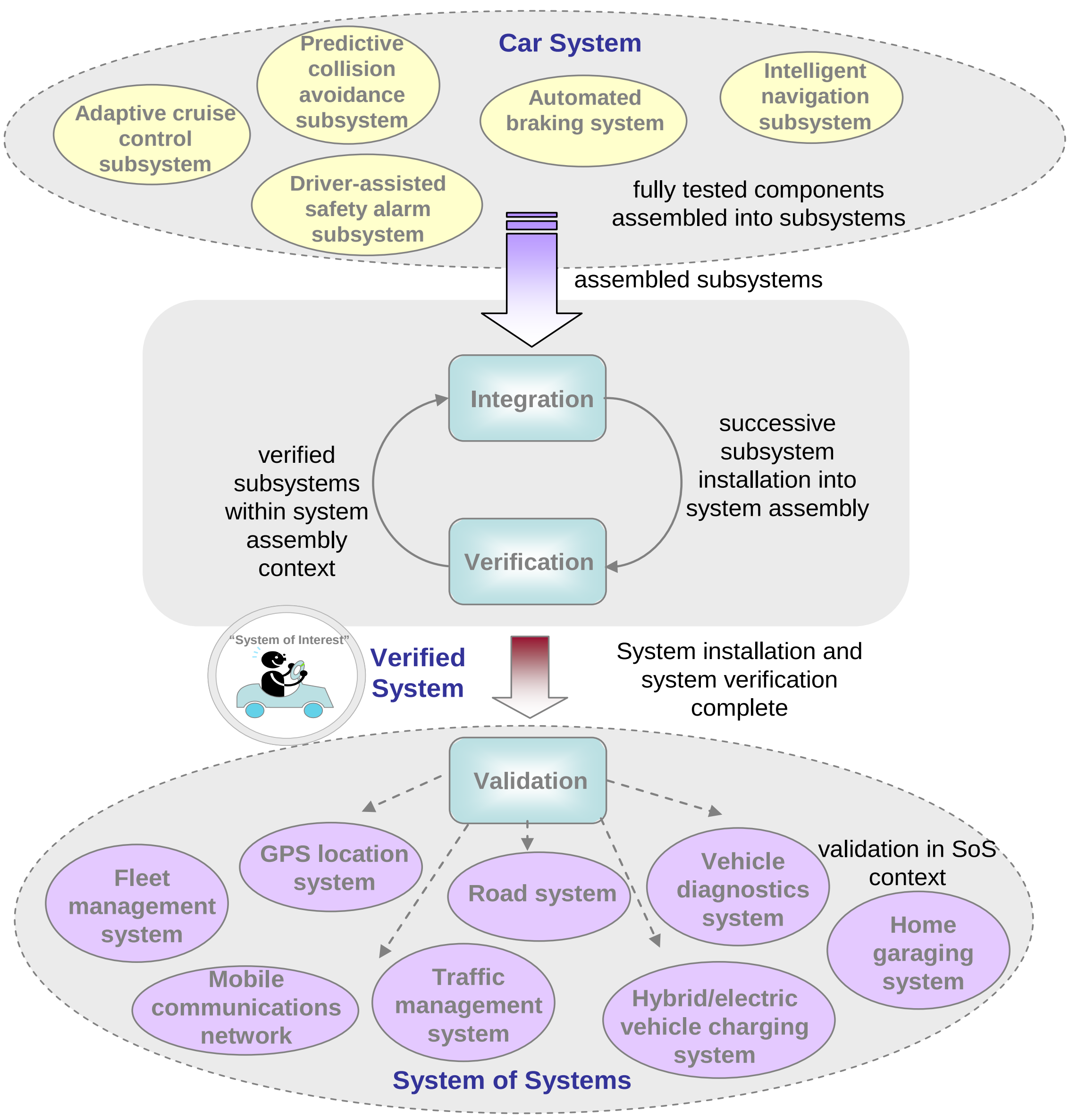

Car System
Predictive collision avoidance subsystem
Adaptive cruise control subsystem
Automated braking system
Intelligent navigation subsystem
Driver-assisted safety alarm subsystem
fully tested components assembled into subsystems
assembled subsystems
Integration
Verification
verified subsystems within system assembly context
successive subsystem installation into system assembly
"System of Interest"
Verified System
System installation and system verification complete
Validation
validation in SoS context
Fleet management system
GPS location system
Road system
Vehicle diagnostics system
Home garaging system
Mobile communications network
Traffic management system
Hybrid/electric vehicle charging system
System of Systems

Figure 47. Integration and Verification – an Iterative Process (adapted from Shamieh 2011).

In IT, systems integration is the process of linking together different computing systems, SW applications, and hardware host platforms physically and functionally to act as a coordinated whole.

For decades, the notion of System Integration (SI) was regarded as a process step within the SE process; being the final verification activity of the assembled system components against the originating system specification. Thus, SI test was the penultimate major SE process step prior to acceptance and delivery of the implemented solution to the customer, where the solution has required design synthesis over multiple layers of decomposition prior to building and assembling the solution components.

The concept of SI has morphed in recent times from a specific SE process step to embrace a much wider notion of purposely bringing together designated piece-parts (principally drawn from current inventory, or acquired as COTS or MOTS, requiring little or no adaptation) to form a system with specific emergent properties to fulfil a specific operational need. This can include circumstances such as retro-fitting an extant system; replacing an obsolete component with a new component with equivalent form, fit and function; or evolving a system by adding more components; or forming an entirely new system by bringing together extant components which were developed for different purposes.

System integration then includes the activity of joining the selected off-the-shelf subsystems and components together. If the interfaces are selected are already designed to connect together, then SI can be reduced to simple assembly and verification activity. If the interfaces were not explicitly designed to connect together, then additional design activity is required to provide suitable interface conversion or mapping, entailing either additional HW engineering or SW engineering activity or both.

This alternate notion of SI is consistent with that from the IT discipline. However, it does not address the scope of the integration activity required, nor the methodologies that may be employed to achieve the SI objectives for system solutions that may overlap between the IT discipline and other technical disciplines.

The latter concept of SI is applicable, for example, when deciding to replace obsolete batteries in high cost equipment, as a lower cost, shorter schedule, and lower risk alternative in lieu of re-designing or replacing an entire equipment suite. Here the requirements may be simple to articulate, without requiring significant engineering documentation, nor significant broader consideration of other technical or socio-technical factors. The engineering task may only require minor effort to undertake the design synthesis, select an off-the-shelf (OTS) solution from a number of suitable alternatives, and perform system integration test to verify the implemented solution meets the originating requirement.

In this example, if the replacement batteries were designed for operation in a similar context to the equipment suite, then emergent properties when installed in the equipment might be expected to be similar to the obsolete batteries. Thus, only limited design evaluation and integration test may be warranted. If the range of OTS potential replacements were designed for operation in a markedly different context, then additional enquiry, test, and documentation may be warranted to ensure the desired attributes can be realised, and to minimise the likelihood of latent defects where unexpected and detrimental emergent properties are encountered in the operating environment.

If no suitable OTS replacements can be found then it may necessitate the raising of a new

SE project to design and build a new replacement battery. This approach would therefore require the more classical application of SI to verify that the implemented solution is fit for purpose as intended, with a low likelihood of latent defects.

The same notion of SI can also be applied, for example, for the addition of a new radio capability to an existing fleet of land vehicles. Here, the land vehicles can be considered as independent systems deployable in their own right, thus integration of a radio into each vehicle in the fleet will occur at the SoS level. Again, the decision to design and build or buy OTS will determine the scope of SoS integration (SoSI) required to verify the implemented solution is fit for purpose as intended.

On a larger scale at the SoS level, a Deployed Force typically comprises extant military capability elements that are brought together for a specific purpose such as a military exercise or deployment in a theatre of operations. Here, the Deployed Force can be considered a SoS, since it is typically comprised of capability elements which are independent systems in their own right.

In this case, a set of requirements for the deployment may be articulated, but the design of the Deployed Force is accomplished by military planning processes rather than through engineering based design synthesis. Some SoS verification may be warranted under some circumstances, for example, if a military platform is a new capability being introduced into service; if it has undergone a major upgrade; or if it is being deployed under markedly different circumstances to that previously encountered or considered.

Because of the large scale of endeavour, it may not be practical to undergo extensive V&V activity of each potential combination of component systems comprising a Deployed Force. However, the Deployed Force can undertake at least some SoSI activity to ensure, at least to a notional level of confidence, that the assembled force will be capable of performing adequately, with minimal likelihood of unexpected and detrimental emergent properties which have the potential to compromise military capability.

This V&V activity can include, but is not limited to "Shakedown" activity. Shakedown activity is sometimes undertaken where a military platform can undertake a series of OT&E style validation activities when newly deployed into an extant theatre of operations, prior to commencing active duty. The purpose of the "Shakedown" is to flush out and remedy any latent defects that manifest under the new circumstances prior to commencing new operations.

The Program of Major Service Activities (PMSA), a series of regularly scheduled military exercises undertaken by ADF, not only provides an important venue for training, but also for performing extended V&V activity supporting SoSI. The PMSA military exercises allow evaluation of different Force deployment combinations from the respective Services and selected coalition partners, under different operational conditions, so both strengths and weaknesses of Force configurations can be identified. This in turn can be used to inform future Force deployment planning at the SoS level, as well as to inform future capability development activity at an individual systems level.

5.10 Human Systems Integration

While consideration of human factors has been integral to SE for decades, an integrating discipline, human systems integration (HSI) has emerged to encourage organisations to take a more considered view of people interactions. HSI provides a number of principles and methods to help integrate people, technology and organisations with a common objective towards designing, developing, and operating systems effectively and efficiently.

HSI scope includes management and organisational concepts and processes as they may interact with the SE processes to achieve cost, safety and performance benefits that might not otherwise have been considered (Sparks 2012). By considering people as a separate capability as well as within a capability, the impact of human properties such as human mental and physical performance, ways of thinking, and reasoning can be explicitly accounted for (Booher 2003).

Historically, human factors has applied scientific knowledge about human psychological , social, physical and biological characteristics to the design and operation of systems in order to achieve the desired human performance, health, safety and overall system effectiveness. Human factors is considered by HSI practitioners as a subset of HSI, spanning the engineering design of equipment, facilities, systems and environments; systems safety; training; manpower; personnel; health hazards; survivability and mobility. However, HSI operates in concert with SE principles to ensure human factors are adequately addressed in the broader context, by providing a systematic process for specifically identifying, tracking and resolving human related issues, seeking a balanced development of both technological and human aspects (Sparks 2012).

6. Defence Notions of a System

6.1 The Capability System

Defence capability is core to the defence of Australia against direct armed attack and to protect its strategic interests. In the Defence context, capability is the capacity or ability to achieve an operational effect. The Defence Capability Development Handbook describes an operational effect in terms of the nature of the effect and how, when, where and for how long it is produced (DCDH 2012). As such, Defence capability is a broad socio-technical concept rather than a prescriptive entity. A socio-technical system is deliberately abstract in nature, characterised as technical works involving the participation of groups of people in ways that significantly affect the architectures and the design of those technical works (Maier & Rechtin 2002).

The Defence Capability Plan (DCP) and the Defence Capability Guide (DCG) are key planning documents guiding acquisition of new Defence capability (DCP 2012), (DCG 2012). Capability acquisition is managed as a portfolio of projects as described in the DCP and DCG. For example, the DCP 2012 is the Major Capital Investment Program for Australian Defence focussing on planned project expenditure over the next four years of the Government's Forward Estimates period 2011-12-2015-16. The DCG 2012 provides an overview of general capabilities and scope of planned future major projects whose approval dates lie beyond the Forward Estimates period of 2015-16, but can change at any time. The principle aim of capability development is to develop and maintain the most operationally effective and cost-efficient mix of Defence capabilities to achieve the Australian Government's strategic objectives (DCDH 2012).

Defence has a maturing capability development process drawing from systems engineering principles as described in the Defence Capability Development Handbook (DCDH 2012). A capability life cycle is ascribed to each capability system to visualise the life of the capability system from the identification of a need (i.e. an existing or emerging capability gap) to the acquisition of a physical capability system which is operated and supported over the life of the capability system until its eventual disposal. Capability development entails those activities involved in defining requirements for future capability.

While the SE fundamentals are widely understood within the international SE community, it is pertinent to ask whether they are shared and widely understood within the Defence capability development and acquisition community, and if they are applicable in the Defence context.

What is a system in the Defence context? The term system is used widely in Defence for many and varied purposes. For the most part, the term is used in a non-technical sense. However, the DCDH provides expansive definitions of terms used in the context of the Defence capability development process.

With reference to the capability systems being acquired, the DCDH minimises the use of the term "system", instead focussing on the notion of capability. Capability is further elaborated in the DCDH in terms of a specific set of "fundamental inputs to capability" (FIC), comprising organisation; personnel; collective training; major systems; supplies; facilities and training areas; support; and command and management. This is distinct from, and not to be confused with "Military Capability", which is the "combination of force structure and preparedness that enables the nation to exercise military power".

System is defined in (DCDH 2012) as "an integrated composite of people, products, and processes that provide a capability to satisfy a stated need or objective". It goes further to say that "a system is a combination or assembly of HW, SW, principles, doctrines, methods, ideas, procedures and workforce, or a combination of them, arranged or ordered towards a common objective". Thus, the capability system is the combination of these assembled individual elements in order to achieve the desired operational effect.

Operational effects in a specific theatre of operation are ephemeral in support of achieving more longer term strategic objectives. This implies the notion of capability system relating to the deployed force is also ephemeral. This is indeed the case, where the deployed force is designed and assembled from its component parts to be fit for purpose for each successive operation.

The capability system does not actually result as an engineering outcome, achieved through the application of a SE process, complete with engineering style products. Rather it results from the application of a military planning process. In effect, this means the capability system is assembled and operated by the end user, although responsibility for individual components may be assigned to different capability managers. For much of Defence capability, the capability managers vested with component responsibility reside within the military services (i.e. Army, Navy, Air Force).

However, with no enduring notion of specific capability systems, there will be no explicit linkage articulated between individual piece-parts and the capability systems they might be considered part of. It will therefore be difficult to formally manage any of the piece-parts in an engineering sense across a managed life cycle if any piece-part in inventory does not become an actual component of a capability system until it is selected for inclusion. Thus issues such as interface compatibility, configuration management, and other technical management considerations cannot be managed on a capability system basis; it is apparent that they can only be managed at the piece-part level.

For a more enduring notion of a capability system, it would be necessary to prescribe a set of system elements or components as belonging to particular capability systems in a SoS context. Only then would it be meaningful to compose a suite of engineering products to enable technical management of the capability system. However, the precise instantiation of such a capability system might never be deployed as each deployed force requires specific tailoring for the specific circumstances at hand.

Notably, most elements of FIC are outside the remit of engineering processes, particularly with regard to personnel management. So different elements of FIC are subject to different management processes and governance mechanisms, depending on which organisation(s) in Defence is responsible for that element.

While the concept of capability may be useful to provide higher-level guidance to assist framing of requirements for acquisition, it is thus apparent the DCDH notion of a capability system is not particularly useful in a SE context. It does not relate to a specific system that can be deliberately engineered and subsequently managed in an engineering sense, particularly with regard to technical management across an explicit life cycle.

6.2 The Materiel System

The materiel system is described in the DCDH as a subset of the capability system, comprising those aspects of the FIC that are supplied by the acquisition agency. The notion of the materiel system as a combination of the mission system and the support system is consistent with the notion described in the ISO/EIA-632 standard, which differentiates between the end products and the enabling products within the system being acquired.

Since the materiel system only exists for the duration of the acquisition activity, it bears no relationship to any instance of a specific capability system assembled by the user. Only the piece-parts being acquired have this association. As such, each materiel system being acquired would appear to have only indirect relevance to any particular capability system. The materiel system as such does not have its own managed life cycle, where issues such as interface compatibility, configuration management and other technical management considerations are not managed on a materiel system basis.

Since materiel system piece-parts are acquired through the application of a SE process, engineering style documentation may be available to assist with ongoing support and technical management of these piece-parts over an explicit life cycle. Specific documentation is prescribed by the DCDH, where Capability Development Group (CDG) has responsibility for preparation of Capability Development Documentation (CDD), comprising an Operational Concept Document (OCD), a Function and Performance Specification (FPS), and a Test Concept Document (TCD). CDD must be provided for each MCE project listed in the DCP (DCDH 2012). However, this relates to the materiel system in its entirety, rather than to individual piece-parts.

Additional SE documentation for materiel system piece-parts may be delivered by the supplier if listed on the Project Contract Data Requirements List (CDRL). The CDRL is typically tailored using contracting principles prescribed by the acquisition agency, in most cases, Defence Materiel Organisation (DMO), which outline mandatory and discretionary contract activity and product deliverables. The quality and completeness of SE documentation for any particular project will therefore be dependent both on that supplied by the Commonwealth, as well as that specified by the Commonwealth, but supplied by the Supplier for individual piece-parts.

6.3 Major Systems

The term "major system" is also used in the DCDH to describe one of the elements of FIC within the definition of a capability system. A description rather than a definition is provided, where it suggests that major systems include significant platforms, fleets of equipment, and operating systems designed to enhance Defence's ability to engage military power. It also notes that major systems can also comprise systems of principal items in their own right, or equipment that regularly requires more detailed reporting and management.

It is thus apparent that there are many and varied definitions and usages of the term system in the context of the capability development process. A SE process is applied to acquire capability system piece-parts within major project boundaries, but not to acquire or upgrade an actual capability system. It is not so readily apparent which systems are actually being "system engineered", who has SE responsibility, and how to differentiate between SE activity and other stakeholder related activity from a basic SE perspective.

Notably, none of the usages of the term system pertaining to DCDH process activity acknowledge the concept of system hierarchy, nor the pre-existence of any architectural relationships between systems or system elements of a hierarchical nature. In the absence of a systems architectural context, this means during acquisition, entities within the materiel system and the major system can be designated as a system or system element at the discretion of each project.

For example, a system can be a major platform; a combat system within a major platform; a radar system within a combat system; or a radar antenna array within a radar system. No specific criteria is applied. Any such designation attributed during acquisition may well be retained after delivery to the capability manager. It is therefore likely to retain the designation during operational service, regardless of its hierarchical relationship to other piece-parts of related Defence capability.

6.4 Project vs. System Context

6.4.1 Specification Considerations

The starting point for each DCP Project phase is the outcome of a capability gap analysis. This is performed independent of any originating requirements for an extant capability being replaced "like for like" or if the capability is to be modified.

Originating requirements are expressed by a project phase in terms of its CDD, including an OCD and a FPS prepared specifically for that phase. The CDD are frequently not updated after contract signature for acquisition, and are not maintained after delivery of the capability. Thus, there is no notion of a persistent set of requirements associated with each materiel system or capability system that is explicitly managed and evolved over the life cycle of the capability.

Similarly, for introduction of new Defence capability, during the early phases of the life cycle prior to the acquisition phase, the capability is described in conceptual form. Defence's desire to remain largely solution independent prior to the acquisition to provide open competition to industry means that the actualised external interfaces to the physical implementation can differ from the initial abstract form. However, there is little impetus to update the project CDD to reflect the actualised form as the scope is outside the acquisition contract.

This is particularly the case where the solution contains COTS components. These may present external interfaces that may require adaptation elsewhere outside the project boundary because of funding arrangements, raising the question whether the adaptation is inside or outside the system of interest, and whether or how the adaptation might be documented.

This is in contrast to the notion of "system of interest" as described in ISO/IEC 15288 which has enduring system identity and boundary, with explicit external interfaces (i.e. explicit input and output interfaces) and set of requirements which are explicitly managed over the life of the system of interest.

6.4.2 Life Cycle Considerations

Since individual systems, subsystems, system elements and equipments are currently modified or replaced within the auspices of designated Defence project phases, notions of ‘system of interest’ identity and ‘system of interest’ boundaries are project-centric, and shaped by funding availability and investment priority. They are therefore ephemeral and can be quite volatile. They have no life cycle as such that can be managed beyond the life of the project phase, which ceases once the delivered capability is accepted into operational service. Thus, each capability component can move in and out of different systems of interest, and different life cycle stages, depending on the individual imperatives of the subsequent projects and project phases.

Verification and Validation (V&V) activity undertaken within the auspices of a project will not necessarily be extensible beyond the project boundaries once the materiel system is delivered and accepted into operational service. Any differences between the delivered materiel system configuration and any operational system, which comprises different combinations of capability components, may not be subject to directed V&V activity that exposes the differences in emergent properties, and hence shortfalls in expected emergent capability. Thus there may be latent defects which can lie hidden for extended periods of time until circumstances arise which bring together system elements in new ways which finally expose the latent defects.

Remediation of more significant latent defects can be problematic as they are typically reported as capability shortfalls, which can trigger a lengthy and potentially costly process starting with the next Force Structure Review (FSR). This can require re-evaluation of strategic guidance, a new entry into the DCP for planning and resourcing, renewed Government approval, and restart of the capability development process to remedy.

Thus selecting the system boundary of interest for each project will determine the scope of systems analysis undertaken and hence will affect the associated risk of an adverse capability outcome at a later date.

6.5 Defence vs. INCOSE System Definition

The Defence definition of system in the DCDH as described above is markedly similar to that offered in the INCOSE Handbook. The differences may be subtle, but an important distinction between the two definitions is the extent to which the system of interest can or is intended to be explicitly identified, in terms of specific system element composition, system boundary location, input and output interface identification, and explicit functions within the system boundaries, all suggested by Mar as having particular significance.

7. Capability Development Process Context

7.1 Defence Capability Life Cycle Model

What is SE in the Australian Defence context? The Department of Defence utilises a SE-like process to undertake acquisition of new Defence capability based around the concept of a Defence capability life cycle as described in Figure 48.

A meticulous description of process and governance requirements for capability acquisition is published in the DCDH (DCDH 2012). Ostensibly, the generic representation is consistent with the waterfall model of system acquisition, originally described by Royce in 1970 for managing the development of large scale SW systems, although individual projects may adopt alternative acquisition models as described in the INCOSE SE Handbook for selected components of capability (Royce 1970).

The Capability Systems Life Cycle is used in the DCDH to visualise the life of a capability system from the initial identification of a capability gap to the acquisition of a physical capability system, which is operated and supported until disposed of. The key tenet of capability development is to develop and maintain the most operationally effective and cost-efficient mix of capabilities to achieve the Government's strategic objectives.

7.2 Defence Capability Planning Guidance

The Defence Capability Plan (DCP) is the key planning document guiding acquisition of MCE for Defence. The DCP comprises a list of projects proposed for Government first pass or second pass approval spanning a rolling 10-year window (DCP 2012). Individual MCE projects within the DCP have responsibility for undertaking SE activity within the confines of their respective project boundaries as shown in Figure 49, where the aggregate capability delivered by the DCP is expected to trend towards achieving that aspired in the Networked Force 2030 (DWP 2009).

7.3 Defence Capability Life Cycle Responsibilities

A plethora of stakeholders contribute to the capability development process as shown in Figure 50, with responsibility transitioning between different stakeholders throughout the capability life cycle.

Notably, while life cycle concepts and language are drawn from the SE discipline, the DCDH provides little explicit acknowledgement of its SE heritage. Instead, the document relies on providing explicit instructions relating to each process step to progress through each of the life cycle stages, identifying specific decision points and governance responsibilities relating to each of the decision points.

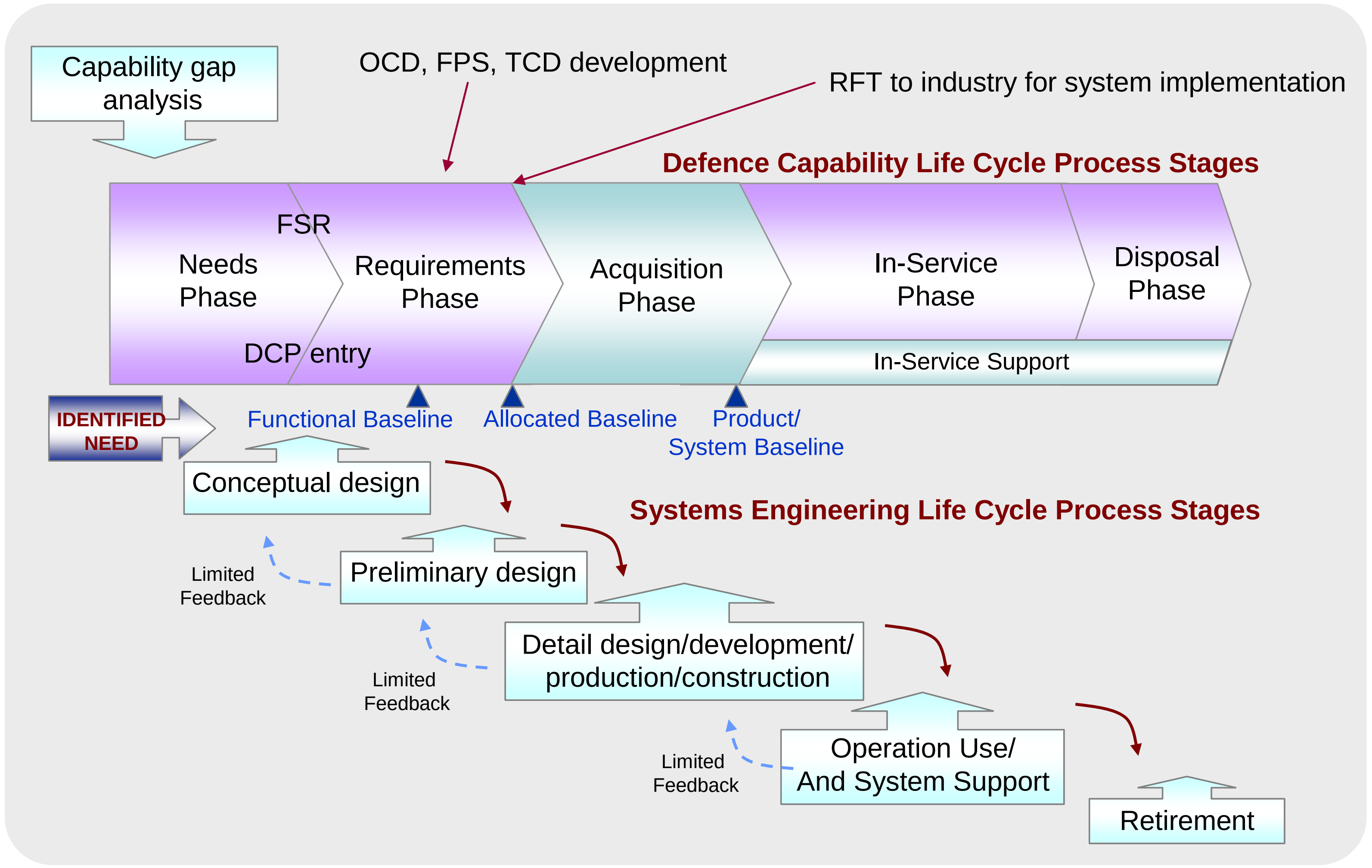


Figure 48. Defence Capability Life Cycle.

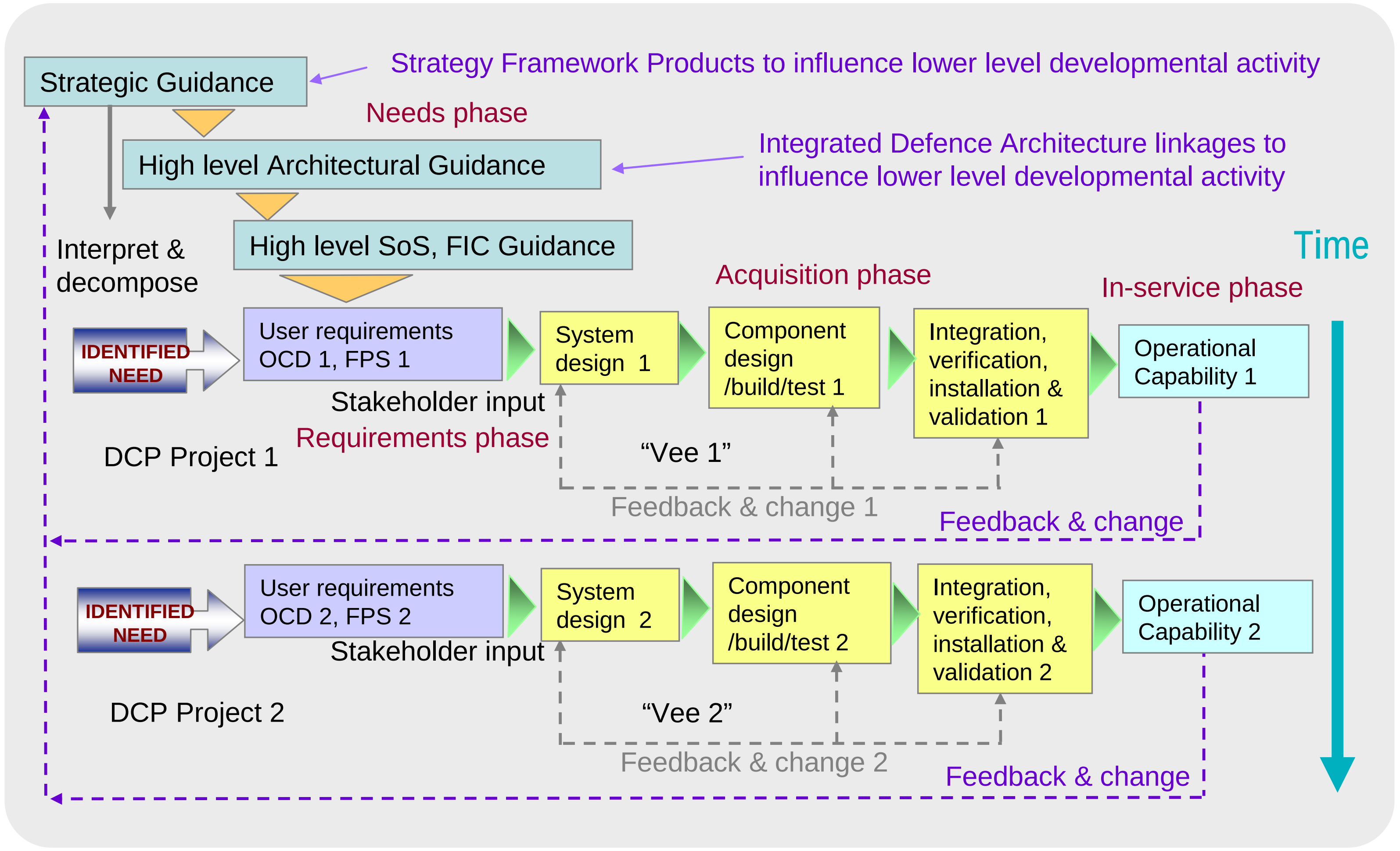

Strategic Guidance
Strategy Framework Products to influence lower level developmental activity
Needs phase
High level Architectural Guidance
Integrated Defence Architecture linkages to influence lower level developmental activity
High level SoS, FIC Guidance
Interpret & decompose
Time
Acquisition phase
In-service phase
IDENTIFIED NEED
User requirements OCD 1, FPS 1
System design 1
Component design /build/test 1
Integration, verification, installation & validation 1
Operational Capability 1
Stakeholder input
Requirements phase
DCP Project 1
"Vee 1"
Feedback & change 1
Feedback & change
IDENTIFIED NEED
User requirements OCD 2, FPS 2
System design 2
Component design /build/test 2
Integration, verification, installation & validation 2
Operational Capability 2
Stakeholder input
DCP Project 2
"Vee 2"
Feedback & change 2
Feedback & change

Figure 49. *Multiple Nested Capability Life Cycle Stages Embedded in the Defence Capability Plan.*

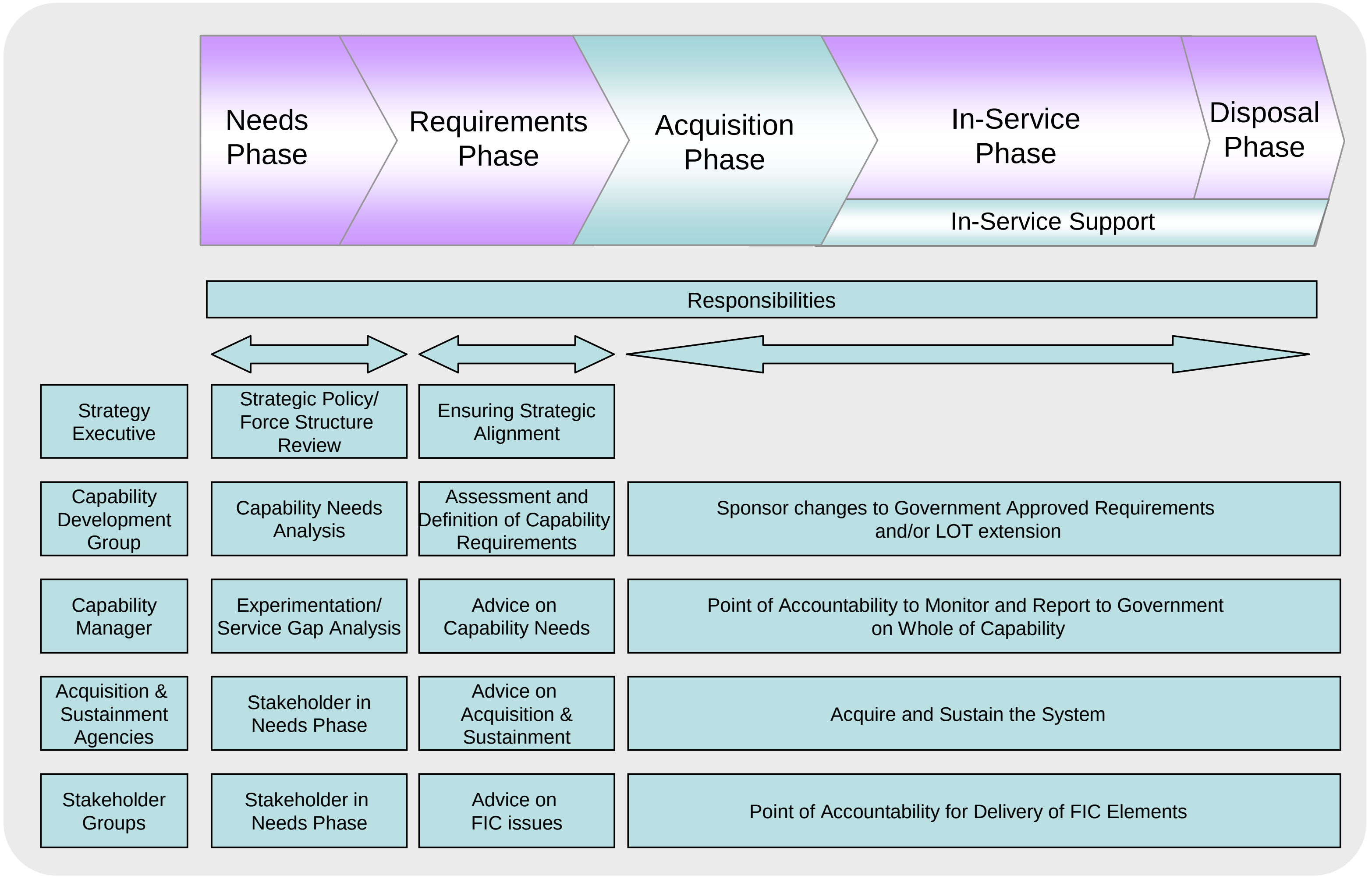

Needs Phase
Requirements Phase
Acquisition Phase
In-Service Phase
Disposal Phase
In-Service Support
Responsibilities
Strategy Executive
Strategic Policy/ Force Structure Review
Ensuring Strategic Alignment
Capability Development Group
Capability Needs Analysis
Assessment and Definition of Capability Requirements
Sponsor changes to Government Approved Requirements and/or LOT extension
Capability Manager
Experimentation/ Service Gap Analysis
Advice on Capability Needs
Point of Accountability to Monitor and Report to Government on Whole of Capability
Acquisition & Sustainment Agencies
Stakeholder in Needs Phase
Advice on Acquisition & Sustainment
Acquire and Sustain the System
Stakeholder Groups
Stakeholder in Needs Phase
Advice on FIC issues
Point of Accountability for Delivery of FIC Elements

Figure 50. *Defence Capability Life Cycle Responsibilities.*

The process also relies extensively on the use of documentation templates to generate the required products to support the decision points throughout capability development process. The use of specific documentation tools is prescribed and tailored training is provided both on the process as well as the tool environment.

The emphasis in the DCDH is therefore on process rather than perspective and profession as offered in the INCOSE SE Handbook. It is thus not readily apparent to what extent the SE body of knowledge is inferred, or whether it has any additional relevance over and above the specific information provided in the DCDH.

8. Defence vs. Industry SE Perspective

8.1 Legal and Political Process Influences

Facts of Life:

If the politics don't fly, the system never will.
Politics, not technology, sets the limits of what technology is allowed to achieve.
Cost Rules.
Affordability is decided by whichever side has the most votes.
The best engineering solutions are not necessarily the best political solutions.
Technical problems become political problems.
With few exceptions, schedule delays are accepted grudgingly; cost overruns are not.

(Maier & Rechtin 2002)

Two different types of decisions are critical to success in system design:

- Value judgements
  - Relative value decided by customers and clients, and
- Technical choices
  - Technical feasibility and implications decided by technical professionals (e.g. engineers and architects) (Rechtin 1991).

The client, as sponsor of a major project, can pre-empt or overrule the technical professionals responsible for implementing the solution, however, no complex system can be optimum to all stakeholders concerned, nor all functions and performance be optimised (Rechtin 1991).

Perhaps the most significant difference between Defence and industry is the influence of the political process in the way that the end-client, i.e. the general public, expresses its value judgements.

Importantly, the Federal Government is both the major sponsor and the major client of capability acquisition activity, acting on behalf of the general public. Decisions are made within the auspices of a formally constituted legal and political process. Ultimately the general public are able to express their judgement through the election process as to the perceived value of goods and services provided by the Federal Government. This infers that value judgements be made by Federal Government representatives on behalf the political constituency.

High-technology, high-budget, high-visibility, publicly supported programs typically offer far greater political challenge rather than just technical challenge. In terms of engineering outcomes, the political process can drive significant design and cost factors such as safety, security, quantity and reliability, and can influence the choice of technologies to be employed (Maier & Rechtin 2002).

This is in stark contrast to a commercial project or product development perspective, where a commercial organisation may sponsor the project or product system design, but the targeted customers may be different commercial businesses and/or the general public consumer, with different purchasing imperatives and different value judgements. The value judgement of the customer is exercised in terms of whether the product is purchased or not; the value to the sponsor is determined by factors such as return on investment, and future prospects for earnings.

8.2 Sourcing Defence Requirements

Responsibility for Defence capability over its life cycle is distributed across different organisational segments, with broad dispersion of authority as shown previously in Figure 50. However, responsibility for capability implementation is shared between Defence and industry.

While many process steps may in principle be common between Defence and industry, the specific implementation of the SE process, including tools and methodologies employed, can differ significantly. SE practice for defence applications will necessarily differ from that supporting commercial product development and support, not the least because of the nature of the Defence contracting environment, where Defence outsources MCE-related system implementation and some aspects of sustainment to industry. They also differ in terms of how the originating requirements are obtained.

A key area of differentiation occurs during system definition when sourcing and analysing needs, and determining system requirements. In Defence, capability guidance is formalised by Government approval at numerous stages during the capability development process.

Initial capability guidance is solicited through a formal strategic planning process that precedes the capability development process[58]; multiple sources of information are analysed in the context of Government Policy and fiscal guidance outside the auspices of the DCDH.

The DCDH prescribes a specific start point in the capability life cycle which occurs at the start of the Needs phase. A specific process is articulated for sourcing requirements through the Needs and Requirements phases as described in Figures 51, 52, and 53. Here the focus is on gathering information required to support Government approval at various stages, including generation and approval of the RFT documentation package.

Typically during these phases, a series of workshops are held with various stakeholders in Defence to garner their individual perspectives. These perspectives are aggregated, sorted, then prioritised through a governance process, where a number of committees of increasing authority successively review the submissions for approval. Finally, a submission is made to Government for authorisation to proceed with the next stage. The set of approved requirements generated through this process is summarised into an OCD and a FPS, which is included in the RFT documentation package to direct industry implementation.

[58] Known as the Strategy Framework 2010 (SF 2010).

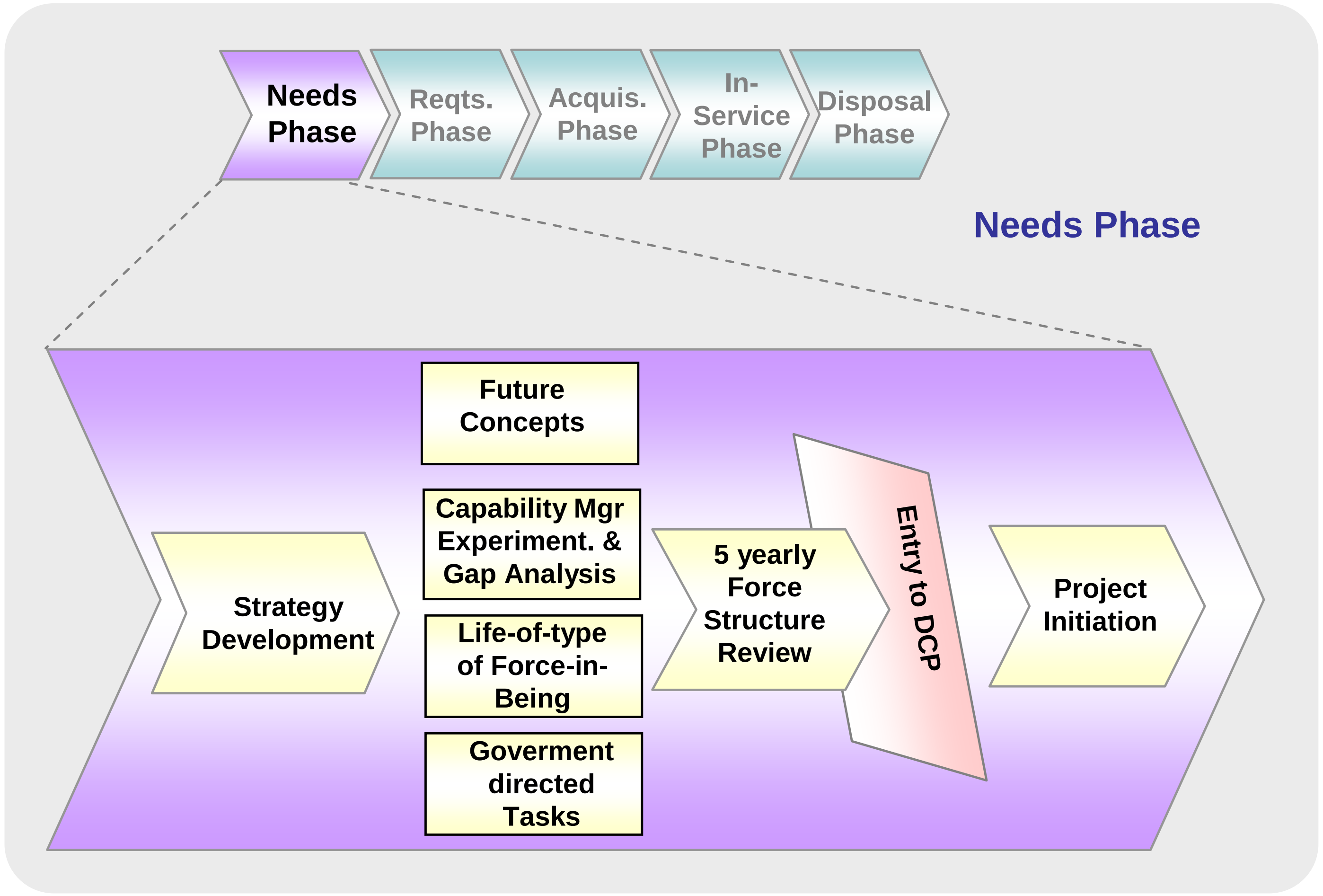


Figure 51. Capability Development Process Outline – Needs Phase.

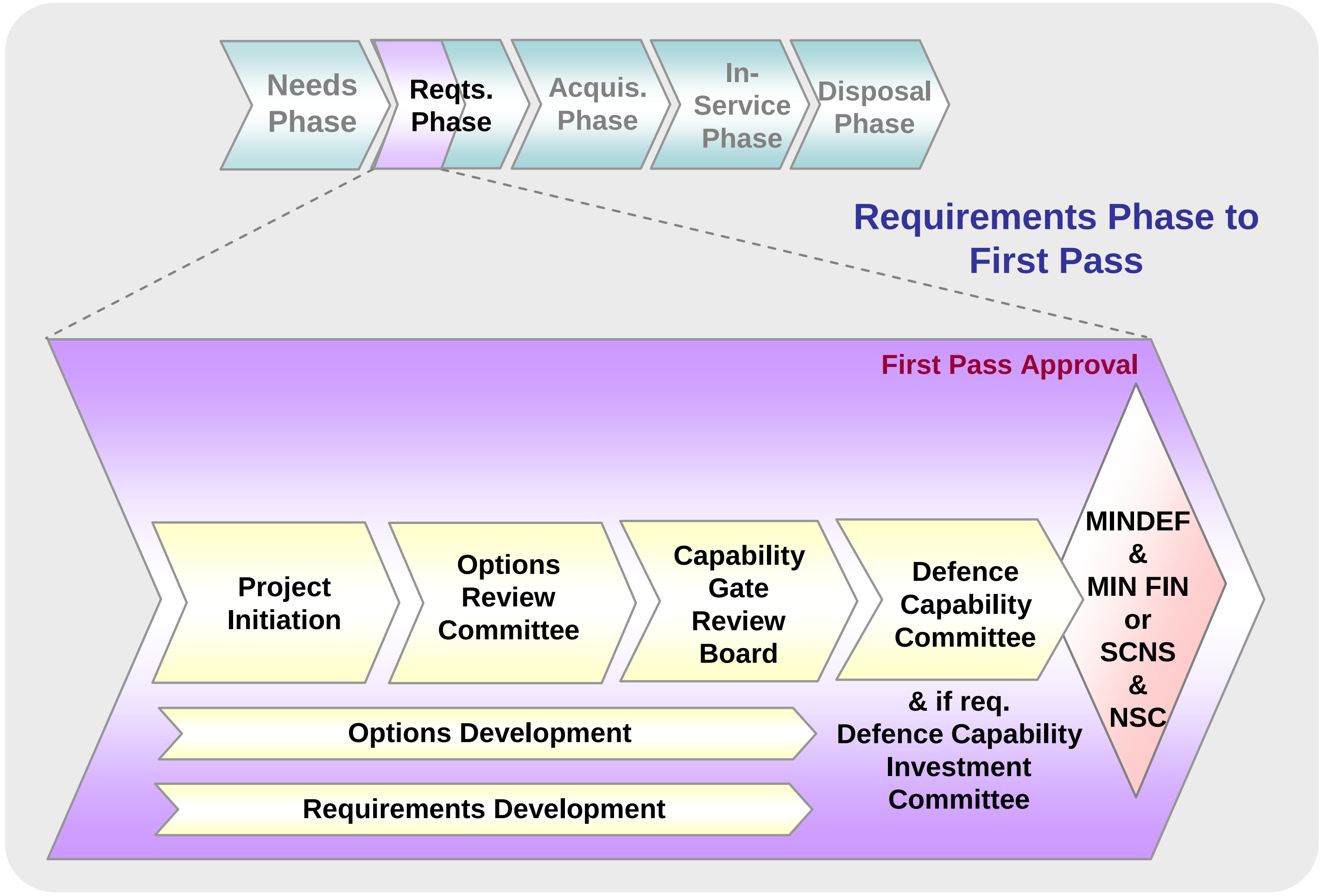


Figure 52. Capability Development Process Outline – Requirements Phase – Part 1.

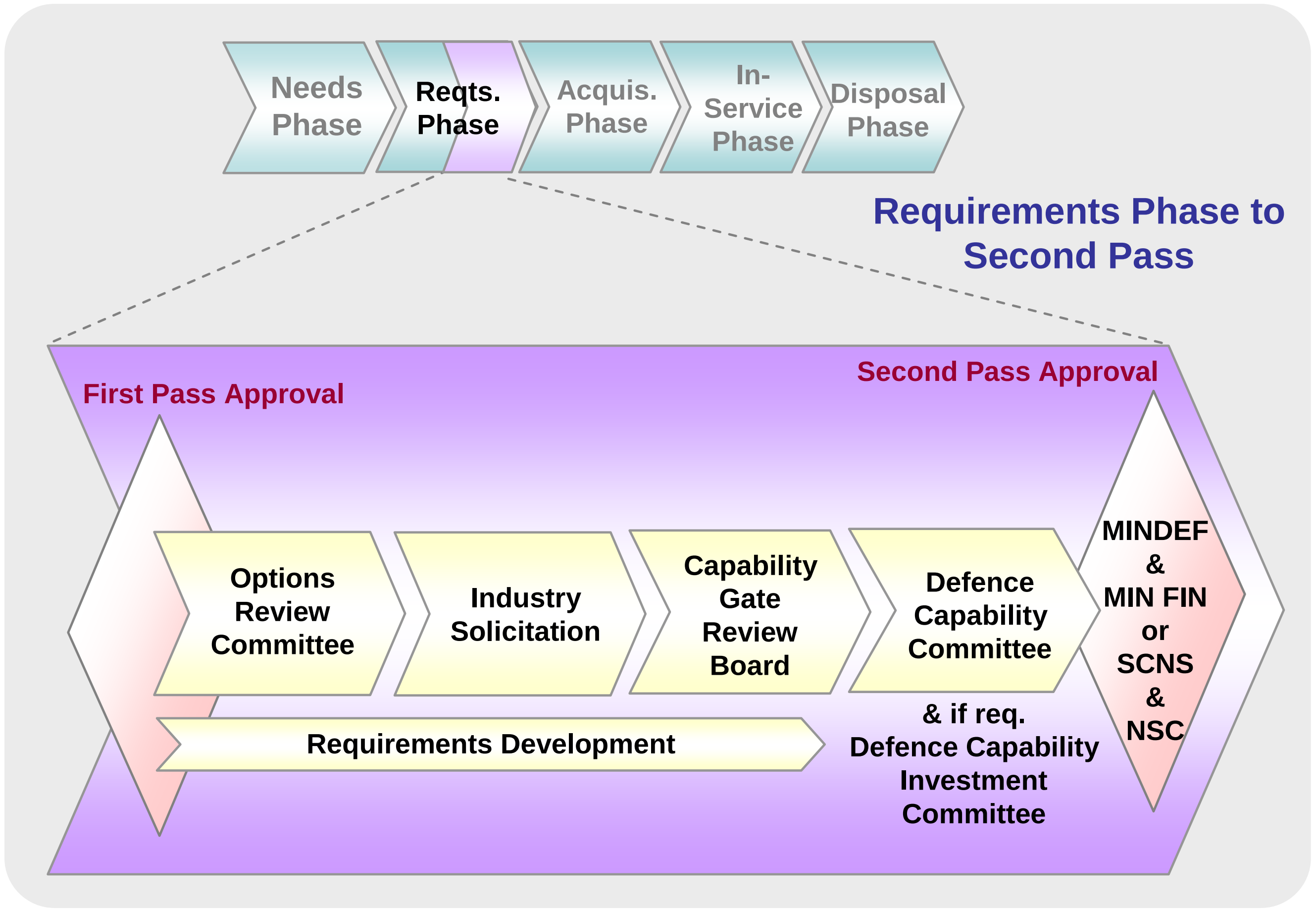


Figure 53. Capability Development Process Outline – Requirements Phase – Part 2.

8.3 Sourcing Industry-based Requirements

The Defence approach contrasts starkly to the approach typically undertaken in industry, particularly for commercial product development.

For commercial applications, market conditions and desired commercial outcomes have heavy influence on system definition and implementation. When developing products for a particular market segment, product features, cost, pricing, location, product promotion, distribution, selling and support all have consideration in shaping the commercial end product, and managing the product over its life cycle.

Commercial considerations such as market conditions, economic conditions, financial circumstances, government policy, and legal obligations also shape resource allocation and planning schedules in order to get the product to market with minimum commercial risk and with maximum return on investment.

This is regardless of whether the solution development is undertaken in-house (i.e. the acquirer is the developer), or whether the solution development is outsourced to an external developer, as previously shown in Figure 31.

8.4 Adaptability to Change

Opportunities to provide feedback and iteration to refine the system concept and its implementation also differ between Defence and commercial developments.

Within the Defence governance process, a number of capability options are developed by Defence with varying scope, risk, and costing. These options are offered to Government for consideration to determine the preferred option to proceed to tender with, thus setting the scope of the approved MCE Project. The RFT seeks to remain agnostic of the potential solution envelope as far as possible to allow industry the widest possible scope for proposing different potential solutions. The tender is awarded on consideration of the best value to the Commonwealth, and the extent of compliance of the offer against the originating requirements. Implementation within industry is driven by the need to deliver a cost-effective and timely solution that can be verified to meet the contracted requirements, within the agreed contract price and schedule. The approval process is, for the most part, linear, providing little or no opportunity to iterate and refine the system concept or scope based on feedback relating to specific implementation considerations, or cross-project or cross-capability inter-dependencies: a critical feature of the SE process.

Significantly, during the tendering process, each tender response submitted by industry is required to include a high-level system design proposal to indicate how their offered solution meets the requirement of the FPS. A form of solution trade-off evaluation is performed during tender evaluation across the respective tender submissions, where the successful tender will be the offered system solution that is deemed the most compliant to the FPS and the terms and conditions of contract, and offering the best value for money for the Commonwealth.

However, the evaluation process does not allow feedback and adaptation across the range of system solutions offered in the tender responses to obtain the optimum system implementation, as would be the case for a cost-benefit trade study or design trade study not bound by probity and tendering constraints.

Since the FPS forms part of the contract terms and conditions for the successful tenderer, a contract change may be necessitated to adapt to changing circumstances. A change in scope or increase in budget may require additional Government approval. This can also invalidate the basis on which the contract was awarded, and can thus expose the Commonwealth to liability for liquidated damages. These higher order considerations leave little flexibility to adapt to changing circumstances once the contract has been awarded.

Furthermore, Industry has very limited opportunity to change the implemented capability once accepted into service unless at the instigation of Defence. Defence, in turn, is primarily dependent on funding availability for further tendering activity, which is dependent on subsequent priorities of the Government of the day.

Legal and governance obligations in Defence provide little opportunity for feedback and adaptation once major decision milestones are achieved, thus limiting flexibility to provide feedback and thus reshape the passage of implementation. This can present major problems where prototyping is required for risk mitigation to address areas of high uncertainty.

Funding availability to undertake various studies is also shaped by the governance process, where different studies are undertaken to support the decision making process as a project progresses from one process step to the next. Once funds are expended to support a particular major decision, it is difficult to revisit the supporting argument on which the decision was made. Additional resources may not be available to revisit the decision criteria in a timely manner; the required adaptation is typically accommodated by a different project, at an earlier point in the project acquisition cycle, which still amenable to change.

Since commercial product development is at the discretion of the commercial entity, there is greater opportunity for feedback and adaptation during initial product implementation and subsequent management of the product over its life cycle. This allows more rapid concept and implementation refinement and adaptation to maturing perceptions of market needs, competitor behaviour, and changing commercial and environmental circumstances over the product life cycle, to ensure the desired business outcomes are achieved.

Typical SE process steps within the major life cycle stages for Defence and industry are contrasted in Figures 54, 55 and 56[59].

[59] Typical industry activities are drawn from (Blanchard & Fabrycky 1998). Defence activities are drawn from the DCDH (DCDH 2012).

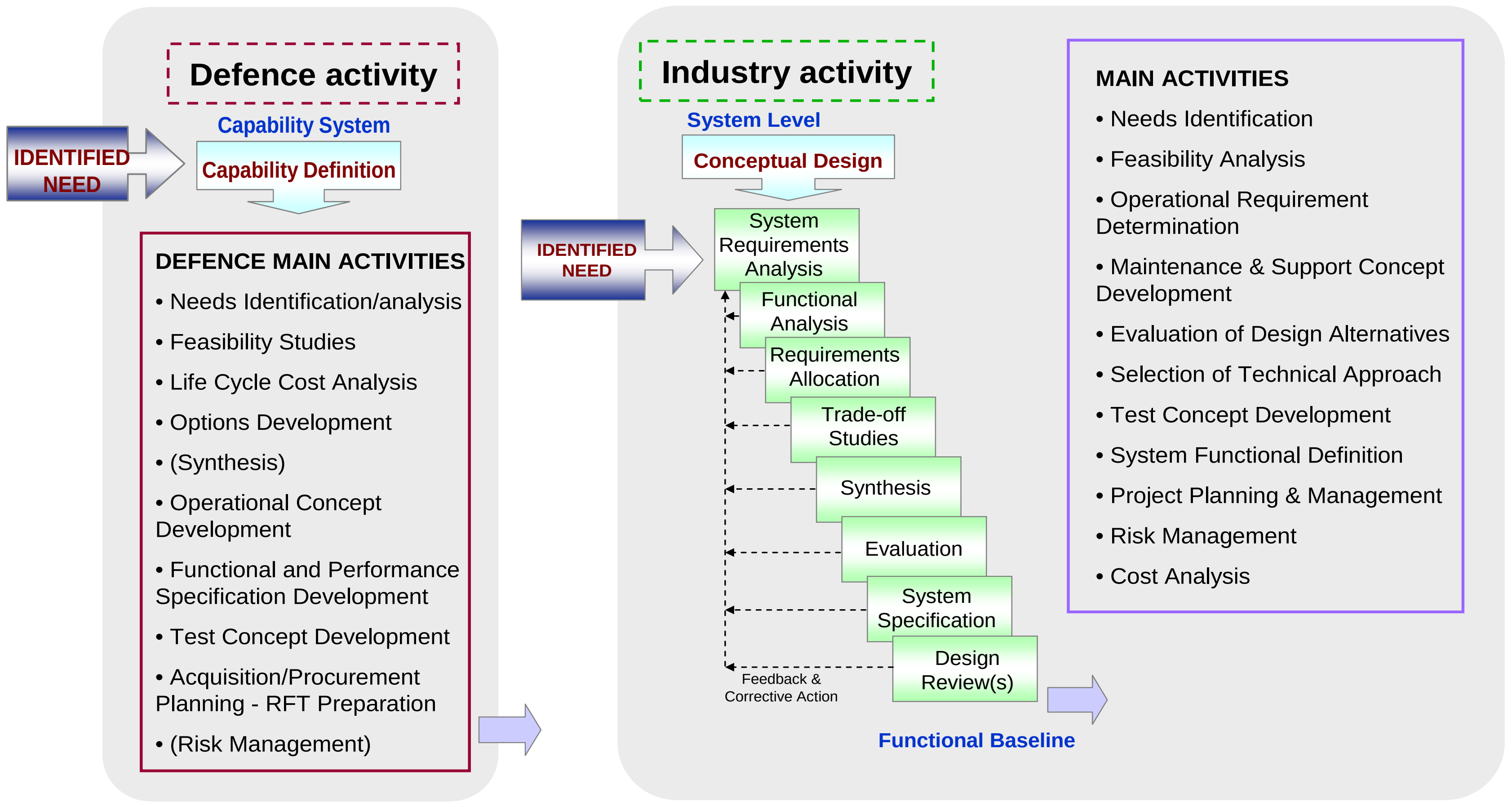

Defence activity
IDENTIFIED NEED
Capability System
Capability Definition
DEFENCE MAIN ACTIVITIES
• Needs Identification/analysis
• Feasibility Studies
• Life Cycle Cost Analysis
• Options Development
• (Synthesis)
• Operational Concept Development
• Functional and Performance Specification Development
• Test Concept Development
• Acquisition/Procurement Planning - RFT Preparation
• (Risk Management)
Industry activity
System Level
Conceptual Design
IDENTIFIED NEED
System Requirements Analysis
Functional Analysis
Requirements Allocation
Trade-off Studies
Synthesis
Evaluation
System Specification
Design Review(s)
Feedback & Corrective Action
Functional Baseline
MAIN ACTIVITIES
• Needs Identification
• Feasibility Analysis
• Operational Requirement Determination
• Maintenance & Support Concept Development
• Evaluation of Design Alternatives
• Selection of Technical Approach
• Test Concept Development
• System Functional Definition
• Project Planning & Management
• Risk Management
• Cost Analysis

Figure 54. Defence and Industry Life Cycle Activities – Needs and Requirements Phases.

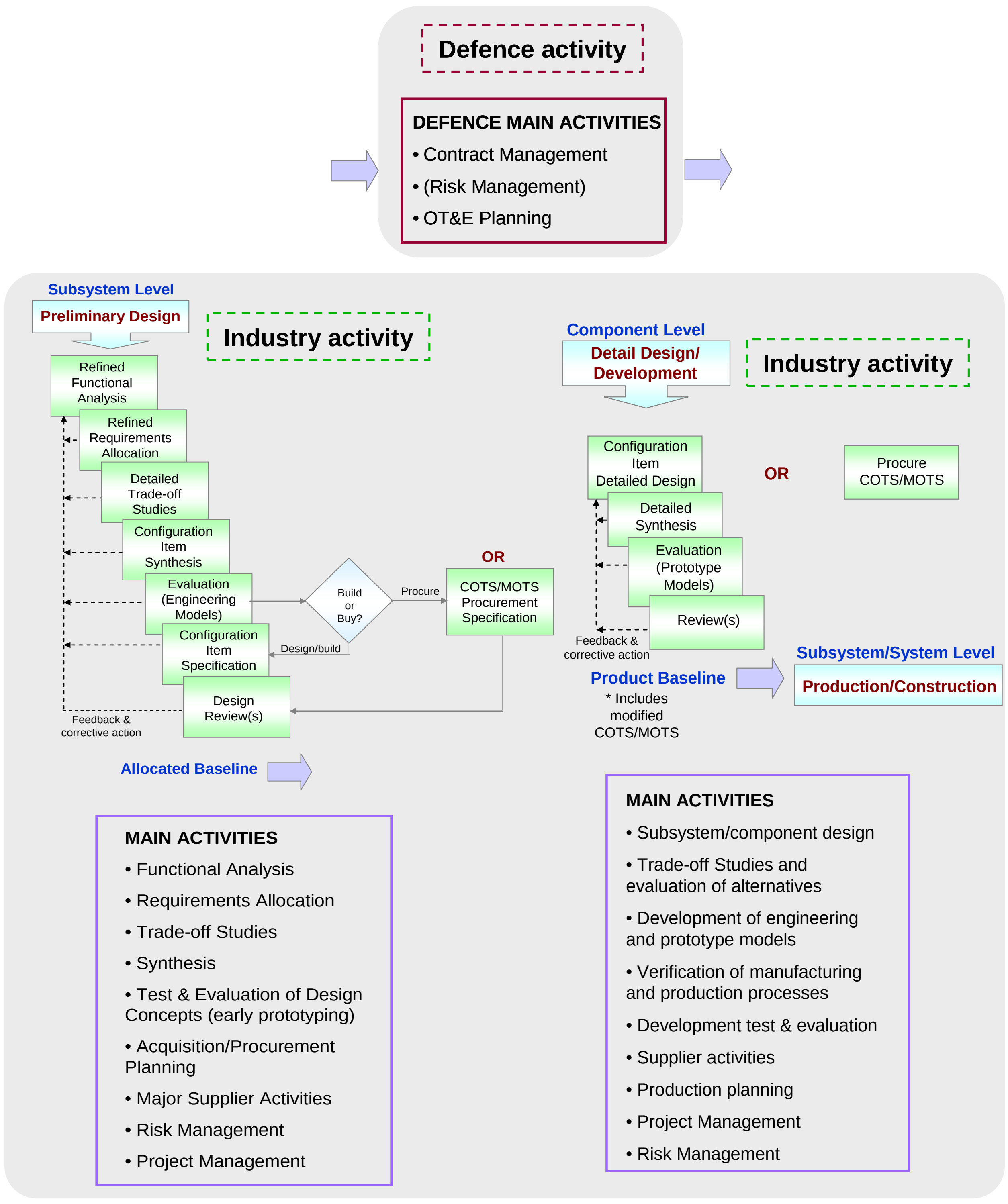

Defence activity
DEFENCE MAIN ACTIVITIES
• Contract Management
• (Risk Management)
• OT&E Planning
Subsystem Level
Preliminary Design
Industry activity
Refined Functional Analysis
Refined Requirements Allocation
Detailed Trade-off Studies
Configuration Item Synthesis
Evaluation (Engineering Models)
Configuration Item Specification
Design Review(s)
Feedback & corrective action
Build or Buy?
Procure
Design/build
OR
COTS/MOTS Procurement Specification
Allocated Baseline
MAIN ACTIVITIES
• Functional Analysis
• Requirements Allocation
• Trade-off Studies
• Synthesis
• Test & Evaluation of Design Concepts (early prototyping)
• Acquisition/Procurement Planning
• Major Supplier Activities
• Risk Management
• Project Management
Component Level
Detail Design/ Development
Industry activity
Configuration Item Detailed Design
OR
Procure COTS/MOTS
Detailed Synthesis
Evaluation (Prototype Models)
Review(s)
Feedback & corrective action
Product Baseline
* Includes modified COTS/MOTS
Subsystem/System Level
Production/Construction
MAIN ACTIVITIES
• Subsystem/component design
• Trade-off Studies and evaluation of alternatives
• Development of engineering and prototype models
• Verification of manufacturing and production processes
• Development test & evaluation
• Supplier activities
• Production planning
• Project Management
• Risk Management

Figure 55. Defence and Industry Life Cycle Activities – Acquisition/Build Phase – Part 1.

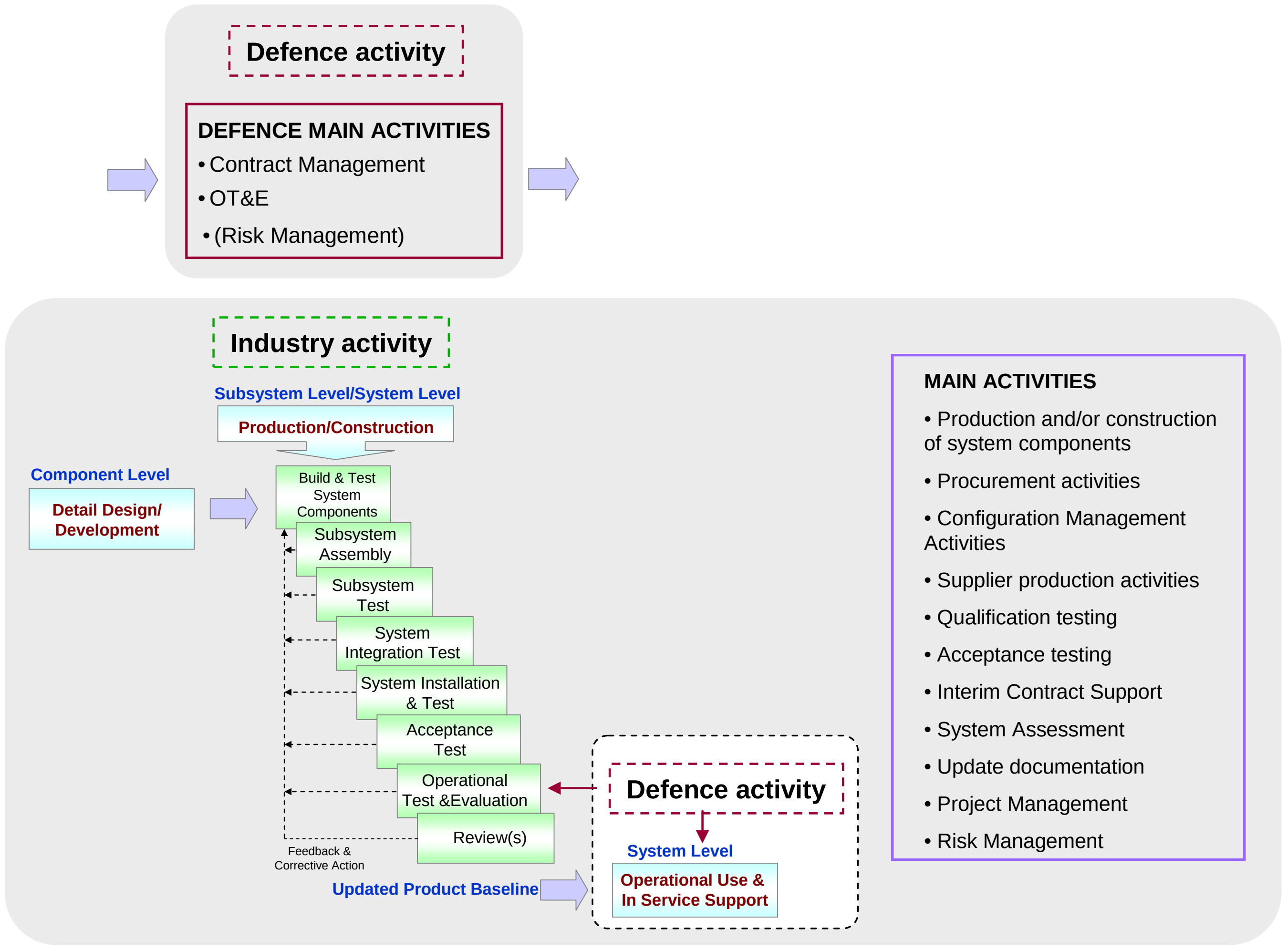


Figure 56. Defence and Industry Life Cycle Activities – Acquisition/Build/Support Phases – Part 2.

9. Enterprise Architecture Concepts

The Defence capability development process operates in tandem with numerous other management, policy, regulatory and governance frameworks. Of particular significance, the concept of EA was introduced into Defence some ten years ago to support ICT acquisition and for MCE acquisition for Defence projects with a significant ICT component, operating in tandem with the capability development process.

The notion of EA was originally developed in the 1980s as a methodology to aid ICT technologists to better understand the business needs of their organisations, and hence better align ICT investment to support the business needs – typically in the absence of more formalised SW engineering methods[60].

EA has since evolved to embrace much wider notions of business process analysis to aid corporate management. EA principally focuses on the business enterprise (i.e. the organisation) rather than the notion of a system, and typically spans people, information, technology and business operations. Since the analysis paradigm is process focussed, it offers few formalisms to consider non-functional aspects relating to the physical implementation (e.g. technological, environmental).

EA focuses on relationships between components, with the architecture comprising the set of relationships of interest within the enterprise in a similar manner to that used SW engineering methodologies such as the Rational Unified Process (RUP) (Peraire et al. 2007). Here, the term "component" is used in a general sense of being a part of the enterprise, without offering further elaboration or definition, or clarification as to what characteristics or attributes might be of significance.

Typically, an EA manifests as a collection of artefacts, comprising lists, drawings, documents and/or models which are used to describe the structure and function of an enterprise in useful ways. Since the EA is inherently conceptual, the architecture descriptions are also typically conceptual in nature, used for communication purposes to support management investment decisions rather than to drive a technical process to implement a specific technical solution.

EA practice can utilise systems thinking, and similar analysis and modelling techniques and tools can be employed as used in operations research (OR) and SE as described previously herein.[61] However, the notion of EA is still very young compared to established scientific disciplines, and there is broad variability in EA concepts and application which have yet to converge to a widely accepted and contemporary body of knowledge. The term EA is used in a variety of contexts, both as a verb and as a noun, including framework, classification schema or taxonomy, methodology, and analytic model.

Various EA frameworks have been published by commercial organisations, principally offering management consulting services or selling computer-based SW applications that provide tools to support business analysis. Commercial EA initiatives include the Zachman Framework for Enterprise Architecture (ZF), offered by John Zachman (Zachman 1987); The

[60] E.g. IEEE/EIA-12207:2008 Standard for Information Technology – Software Life Cycle Approaches.

[61] Differences between systems engineering approaches and enterprise architecture in the Defence context are examined in more detail in (Hue 2011).

Open Group Architecture Framework (TOGAF) offered by The Open Group (Josey 2009), and the Gartner Enterprise Architecture (GEA) offered by Gartner Inc. (formerly known as the Meta Framework)(Lapkin 2005), (Bittler & Kreizman 2005)[62]. Definitions of EA vary from framework to framework. Commercial EA frameworks typically offer reference models to frame thinking about the enterprise from different perspectives, however, they typically are not prescriptive with regards to the range and type of artefacts to be produced, nor with the artefact format or information content.

There has been strong uptake of EA concepts within Government Defence Organisations in particular in a number of countries around the world. These include the Department of Defense Architecture Framework (DoDAF) in the US DOD[63]; the Ministry of Defence Architecture Framework (MODAF) in the UK MOD[64]; the NATO Architecture Framework (NAF)[65] employed by NATO countries; and the Department of National Defence Architecture Framework (DNDAF)[66] developed by the Department of National Defence and Canadian Forces in Canada. [67]

A key differentiator between the commercial EA frameworks and the Defence EA frameworks is in the provision of definitions and formalisms; the Defence EA frameworks having far greater emphasis on explicitly defining terms in common usage and providing supporting formalisms to provide consistency in application to aid governance.

The notion of architecture used by the DoDAF (and widely used in the SE community) is drawn from the SW-centric definition of the term architecture, defined in IEEE-610.12 and IEEE-1471 as "the fundamental organisation of a system embodied in its components, their relationships to one another, and to the environment, and the principles guiding its design and evolution". The system in the context of the definition is the same as that used by the SE community except that the system referred to in the standard is a *software-intensive* system. Importantly, the IEEE-1471 standard defines a *software-intensive* system as any system where software contributes essential influences to the design, construction, deployment and evolution of a system as a whole[68].

The respective Defence EA frameworks make extensive use of object-oriented modelling and graphical presentation techniques drawn from the SW engineering discipline as previously described herein, including those described by the SW graphical modelling language UML and system graphical modelling language SysML, managed under the auspices of Object Management Group (OMG).

Recently there has been a move to merge aspects of the various Defence frameworks,

---

[62] A comparison of the Zachman, Gartner, FEA and TOGAF enterprise architecture methodologies is provided in (Sessions 2007) [online]: http://msdn.microsoft.com/en-us/library/bb466232.aspx.

[63] Information on the DoDAF can be found at [online] URL: http://dodcio.defense.gov/dodaf20.aspx (DoDAF 2010) and at *https://www.us.army.mil/suite/page/454707* (DoDAF 2009).

[64] Information of the MODAF can be found at [online] URL: http://www.modaf.org.uk/ (MODAF 2010).

[65] Information on the NAF can be found at [online] URL: http://www.nhqc3s.nato.int/ARCHITECTURE/_docs/NAF_v3/ANNEX1.pdf (NAF 2007)

[66] Information on the DNDAF can be found at [online] URL: http://www.img-ggi.forces.gc.ca/pub/af-ca/index-eng.asp.

[67] A detailed analysis of the respective enterprise architecture frameworks is provided in (Hue 2008).

[68] Eeles provides a detailed explanation of various terms referred to in IEEE-1471 including architecture, system and environment, and their relationship to the RUP and RUP SE modelling environments and software architecture modelling in a series of articles on architecting (Eeles 2006a), (Eeles 2006b) and (Eeles 2006c). A detailed description of the process of software architecting is provided in (Eeles & Cripps 2009).

resulting in the creation of the Unified Profile for DoDAF and MODAF (UPDM) SDL. UPDM defines a standard usage of UML and SysML language constructs across these Defence architecture frameworks and their respective tool sets to generate the respective framework artefacts (Hause 2010), (McDaniel 2012), (Okon 2012). A meta-model of UPDM is provided in Figure 57.

Planning for a Unified Architecture Framework (UAF) to supersede the DoDAF, MODAF, NAF and DNDAF is already underway, and has been embraced by the broader international Defence community including US DoD, UK MOD, Swedish DOD, Canadian DND, and NATO. A timeframe circa 2013 was initially sought for the planned codification of the UAF as an international standard under the auspices of OMG (Okon 2012), (DoDAF TWG 2012).

The commercial EA framework TOGAF leverages both MODAF and DoDAF by suggesting the Defence artefacts can provide useful representations of the enterprise developed using the TOGAF Architecture Development Method (ADM). They use the tool sets provided by vendors who support DoDAF and MODAF. However, TOGAF belies the relationships between the data populating the artefacts, the underpinning analytical methods, and the engineering processes used to determine the data under the auspices of the MODAF and DoDAF.

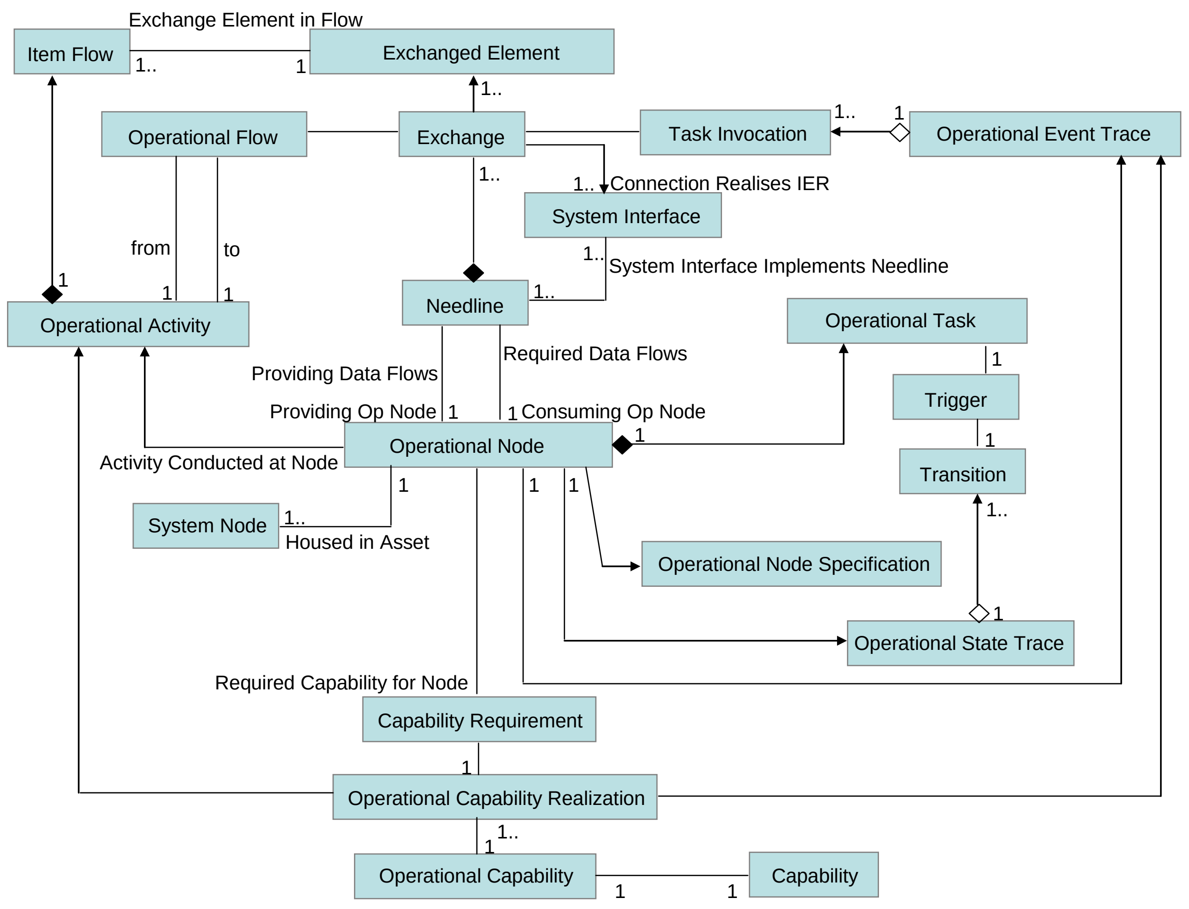


Figure 57 UPDM Meta-Model Based on UML (Dickerson & Mavris 2010)

10. Defence Enterprise Architecture Context

10.1 Defence Architecture Framework

Along with other allied nations, Australia has also embraced EA, first by Defence, then across broader Government. EA was introduced into the Australian Department of Defence circa 2003 under the auspices of the Defence Architecture Framework (DAF) [69]. The initial DAF concept was based on EA concepts developed by META Group[70].

While the EA concept in Defence is still evolving, three key thrusts have emerged:

- one relating to artefact presentation,
- one relating to methodology, and
- one relating to reference models and reference architectures.

The DAF, now known as the AUSDAF, essentially comprises a set of templates, containing specific diagrammatic forms and tables of prescribed information. These templates were derived from those originally developed by the US DoD under the auspices of the DoDAF. However, the AUSDAF implementation differs significantly from the DoDAF.

The notion of architecture used by the AUSDAF differs significantly to that used by the DoDAF. The AUSDAF draws from the IEEE-1471 definition of an architectural description for SW-intensive systems. The architecture description is a collection of products (i.e. artefacts or populated templates) to document an architecture. The DAF does not explicitly acknowledge the existence of an actual system architecture itself. Instead, attention is given to preparing artefacts drawn from a standardised set of scenario descriptions (i.e. use cases), re-cast from the perspective of the respective MCE Projects, with a focus on describing the applicable business (i.e. operational) processes.

In Defence, particular effort is taken to ensure the architecture descriptions remain conceptual in nature to support RFT preparation; conveying ideas rather than actual system implementations. They therefore remain largely solution independent. Consequently, this limits their utility in performing system trade-studies and hence limits the ability to drive real-world engineering implementation.

The US DoD approach provides significant guidance for the generation and management of DoDAF related information, and has sought to align architecture practice with their SE processes to assist in the system implementation, with strong emphasis on system and component identity, interface management, and information management (Okon 2012), (McDaniel 2012).

Architecting as described in IEEE-1471, is simply the activities of defining, documenting, maintaining, improving and certifying proper implementation of an architecture of a SW-intensive system. Thus, the data required to populate the artefacts can be extracted from analysis typically undertaken within the SE process; the artefacts providing specific viewpoints of the system and how it is used from an architecture framework perspective.

[69] The initial version of the AUSDAF was referred to as the DAF.

[70] META Group merged with Gartner in 2004. [online] URL: http://www.gartner.com/id=486650.

The Australian Defence approach is much simpler, comprising a mandate for inclusion of specific diagrammatic forms and tables in the OCD, one of the CDD documents required within the capability development process. Instead of similarly following the DoDAF guidance, AUSDAF guidance is provided in the form of a reference model to provide context to the AUSDAF templates, as shown in Figure 58.

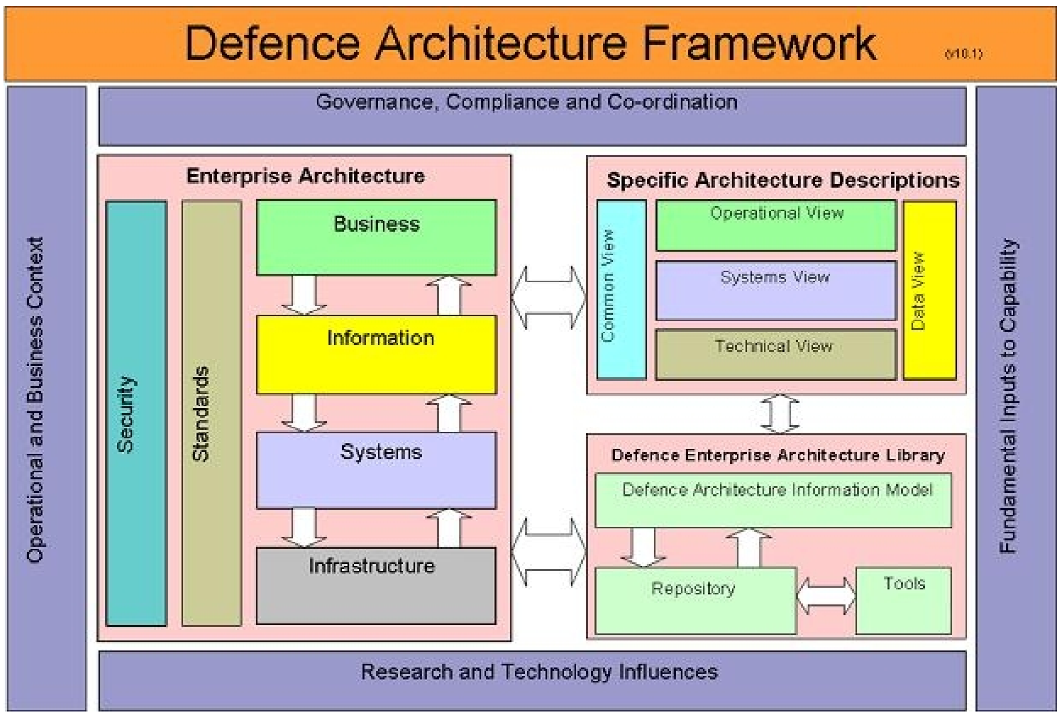

Defence Architecture Framework
(v10.1)
Governance, Compliance and Co-ordination
Operational and Business Context
Fundamental Inputs to Capability
Enterprise Architecture
Security
Standards
Business
Information
Systems
Infrastructure
Specific Architecture Descriptions
Common View
Operational View
Systems View
Technical View
Data View
Defence Enterprise Architecture Library
Defence Architecture Information Model
Repository
Tools
Research and Technology Influences

Figure 58. Defence Architecture Framework Reference Model (Purcell 2009).

A tool set is also prescribed to prepare AUSDAF artefacts. Guidance is provided on tool usage for artefact creation, where emphasis placed on the use of templates to provide consistency of presentation of information rather than on coherency or management of information content. Guidance is also provided for the use of a prescribed set of scenarios to assist populating OCD document templates, however, the method of collecting, managing and using the information is at the discretion of the respective projects.

Significantly, the AUSDAF artefacts are used for a different purpose than DoDAF artefacts. DoDAF artefacts are generated as outcomes of specific directed enquiry, and are used to inform aspects of the JCIDS capability acquisition process used by US DoD. For example, they are used extensively, for development and articulation of higher-level pan-organisational concepts such as their Global Information Grid (GIG), and for pan-organisational implementation guidance relating to specific system characteristics, relationships and standards (McDaniel 2012), (FORCEnet 2004a), (FORCEnet 2004b), (Ryder & Flanigan 2005).

AUSDAF artefacts are project-centric and solution-independent. Their primary use is to communicate conceptual information to senior decision makers for MCE project funding approval and governance purposes. They are not intended to form a technical specification nor to document particular analytical outcomes to drive real-world system and detailed design. AUSDAF artefacts also lack many of the formalisms required to drive real-world implementation, including the non-functional aspects (e.g. environmental), and SE process activity such as requirements specification and V&V.

Prior to July 2011, governance in Defence included formal examination of the AUSDAF artefacts prepared by each DCP Project phase with significant ICT content against a formalised NCW compliance framework (Knight et al. 2006). However, this review step was deleted in July 2011 from the capability development governance process, and responsibility for AUSDAF artefact review was transferred to an EA governance process.

Because of the conceptual nature of the EA diagrammatic representations in the OCD, and the absence of traceability to the FPS in an engineering sense, the EA artefacts do not provide information that can be accredited or warranted for use to provide specific technical guidance for system implementation. Instead, the FPS is developed as an interpretation of the OCD that is relevant to the required materiel system to be delivered under contract within the auspices of the particular DCP project phase; no EA artefacts are mandated or offered for inclusion in the FPS template.

Since the originating OCD is not included as part of the contractual basis for system implementation, and is not maintained under configuration control over the life cycle of the delivered materiel system, the EA artefacts do not have direct influence on system implementation and subsequent life cycle management. Since the FPS provides the technical basis for the contract, the information in the FPS is warranted for use to direct system implementation, and is maintained under configuration control until delivery of the materiel system.[71]

Since the AUSDAF is artefact focussed rather than data focussed, there is no concept of a data repository, central or otherwise, nor any concept of a data model. Instead, DCP Project generated artefacts are archived as project specific documentation (i.e. drawings and text) along with other project generated documentation. The architecture repository is in effect virtual, and comprises the aggregation of all the DCP Project-specific artefacts.

An expanded version of the DAF, known as AUSDAF2, to better support the notion of the Integrated Defence Architecture (IDA) was investigated to provide a greater range of artefacts, as shown in Figure 59 (Yannopoulos 2010). However, AUSDAF2 has since been recast to span the same artefacts as DoDAF 2.0. There has been no mention of plans to introduce a meta-model for AUSDAF2 akin to the meta-model DM2 underpinning the US DoDAF implementation.

Since the AUSDAF2 approach remains focused on providing consistency in presentation of information rather than consistency in data content, the changes from version 1.0 have no impact on analysis requirements for the same artefacts, nor on capability development and ICT acquisition process and governance.

Importantly, the AUSDAF2 developments do not embrace system-modelling concepts, and therefore do not provide a pathway towards supporting MBSE specific data-orientated constructs nor the proposed Unified Architecture Framework.

[71] The FPS is not maintained under configuration control subsequent to initial delivery and acceptance into service for the remainder of the delivered capability's life cycle.

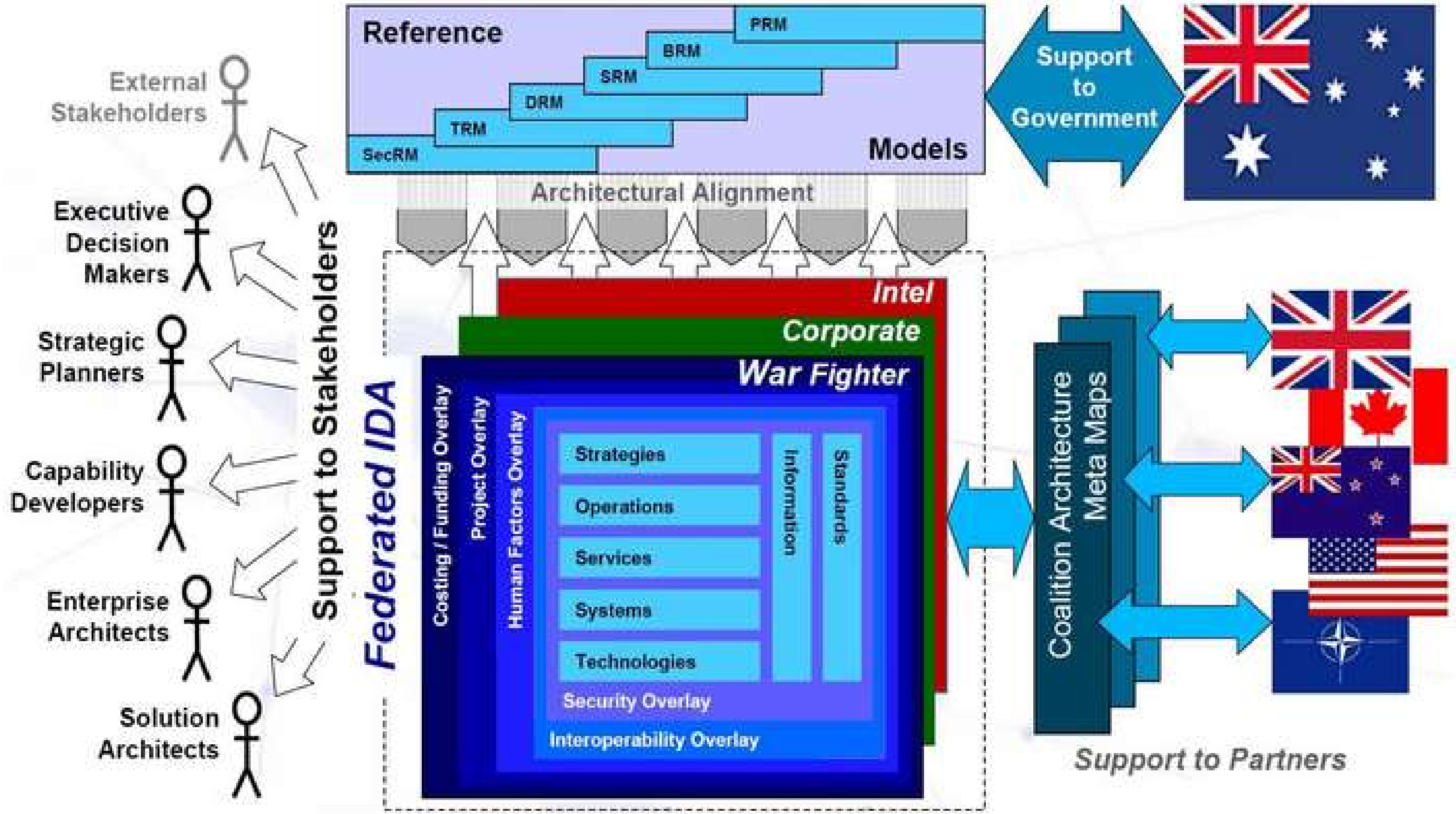


Figure 59. *Draft AUSDAF 2 Overview as of 12 November 2012.*

10.2 Integrated Defence Architecture

In 2008 in response to the Gershon Review (Gershon 2008), the Australian Government Information Management Office[72] (AGIMO), within the Department of Finance and Deregulation, EA Policy and Concepts, commenced an EA initiative to promote standardisation of ICT infrastructure and processes across whole-of Government. In support of this initiative, AGIMO has developed and mandated use of a set of reference models, collectively known as the Australian Government Architecture (AGA) (AGIMO 2011).

Similarly, the 2009 Defence ICT Strategy sought to strengthen the relationship between Defence capabilities and Defence ICT products and services, and to provide tighter cost control, greater efficiencies, and faster decision cycles for ICT investment (ICT 2009). An EA approach was directed to manage Defence's ICT portfolio and the notion of the Integrated Defence Architecture (IDA) was introduced to provide a basis for planning and governance for ICT architecture development, delivery and maintenance.

A companion document "The Single Information Environment: Architectural Intent 2010" was also released to detail the proposed architectural transformation to achieve the interoperable and networked Future Force (SIE 2010) as envisaged by the Defence White Paper 2009 (DWP 2009). While the initial impetus of the IDA and SIE was associated with ICT, the scope has since evolved to embrace a much more substantial portion of ADF's military capability. Its scope encompasses any military system that has an external interface which exchanges information (i.e. is not a mechanical interface - ranging in scale from an individual soldier to large-scale complex SoS such as the Air Defence Ground Environment (ADGE).

The IDA scope circa 2012 encompassed all common or shared ICT-related assets, business strategies, business processes, investment, data, systems and technologies across the whole of the Defence organisation, spanning the corporate environment, intelligence, and the warfighting environment. The initial implementation of the IDA focussed on documenting the extant business architecture, which it defined as the business strategy, governance, organisation, and key business processes information, as well as the interactions between these concepts. As such, there were no provisions within the scope of the IDA circa 2012 to describe extant, future planned, or conceptual ICT infrastructure or other related systems or components.

A specific initiative was launched to develop the concept of the IDA and the initial business reference models, and to start populating the business areas with data. The IDA itself is a reference model as shown in Figure 60, showing the relationships between different reference models comprising the IDA (Purcell 2010).

The IDA is also represented as shown in Figure 61, indicating the various domains where individual artefacts are generated, including the warfighter, intelligence and corporate domains (Yannopolous 2010). Architecture reference guidebooks are also provided for specific business sub-domains or architecture segments to provide guidance on using the AUSDAF to prepare architecture descriptions (JIA 2010), (NBA2020+2011), (DCA 2011).

[72] [online] URL: http://agimo.gov.au/

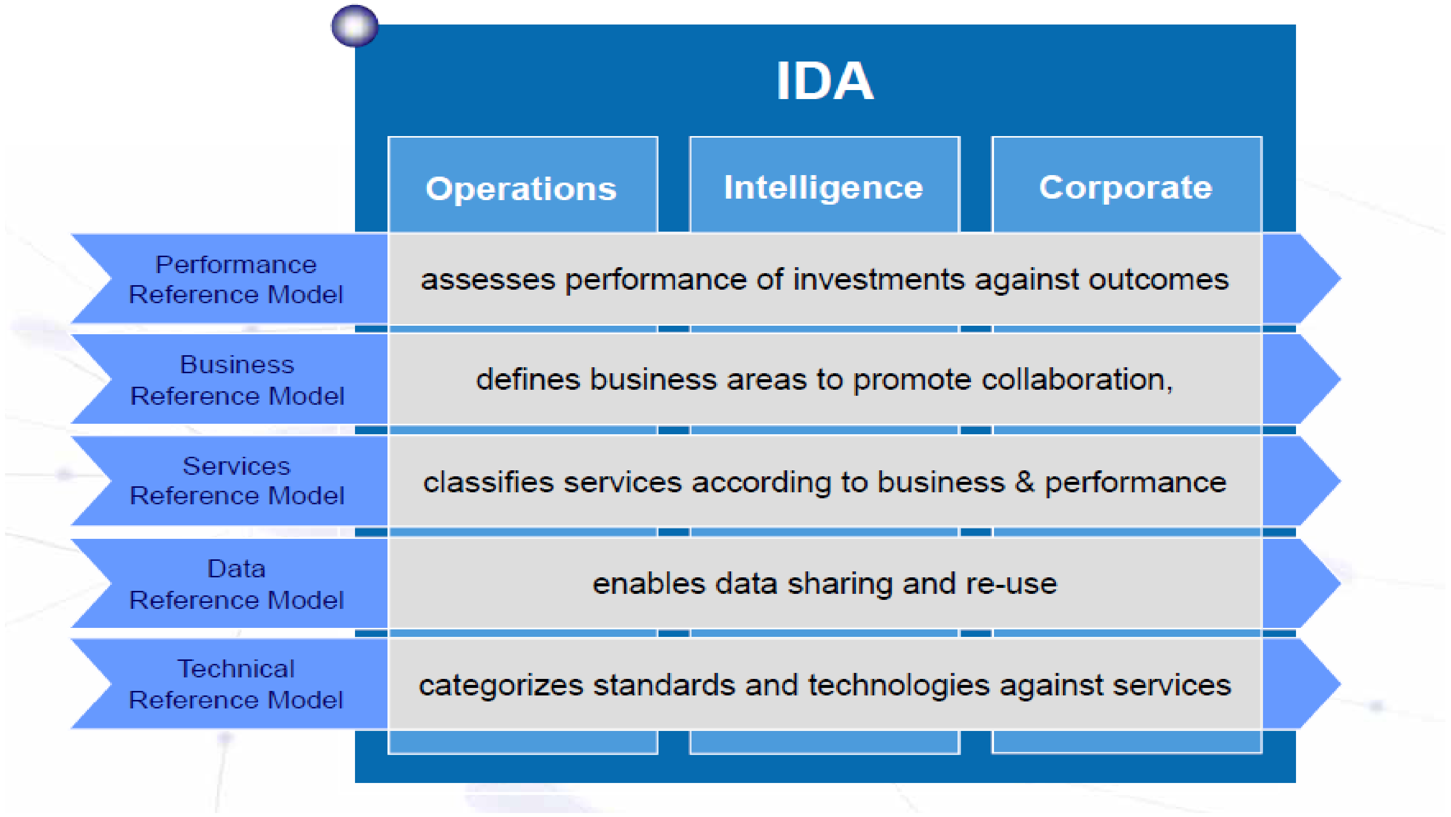
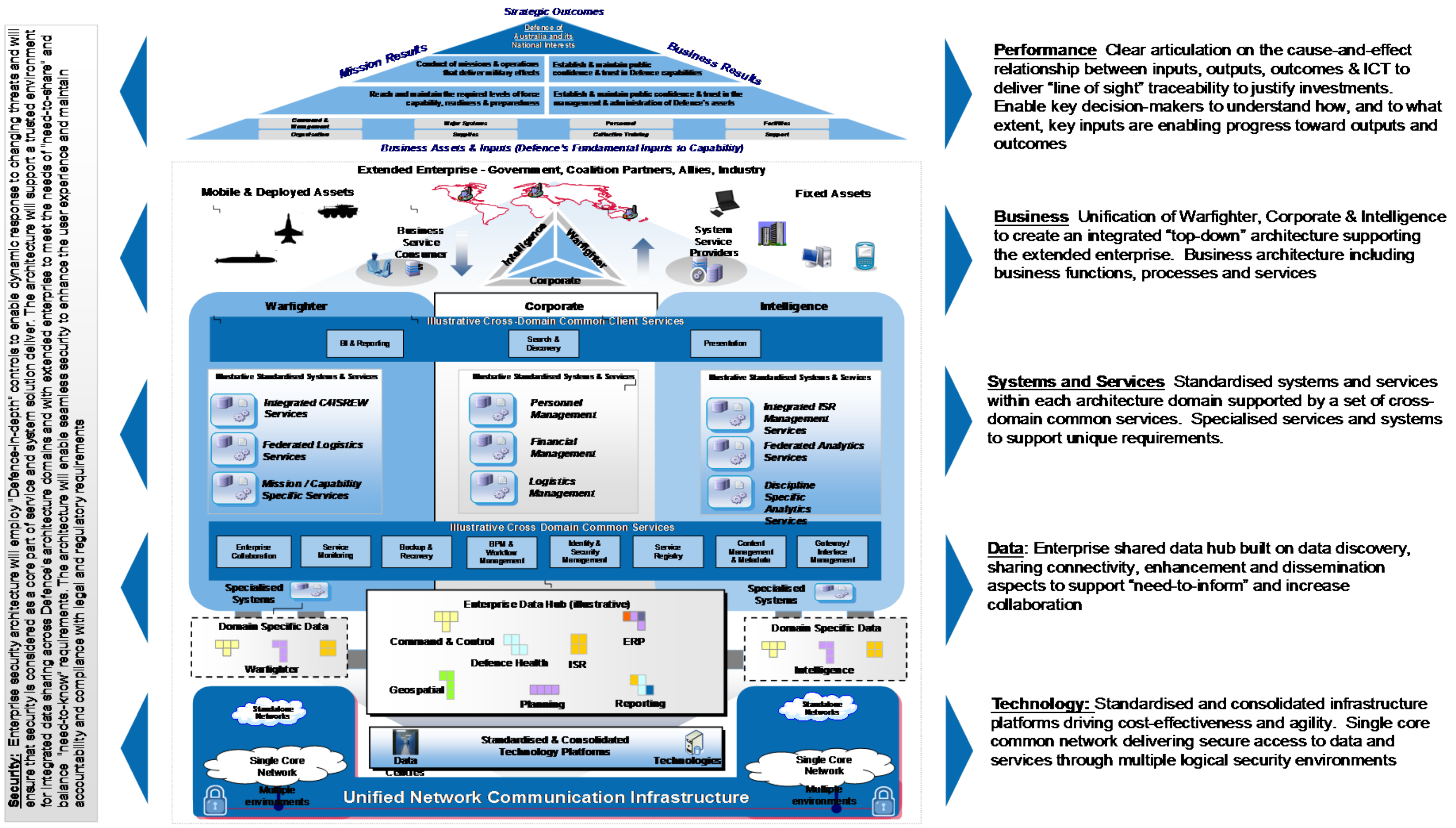

IDA
Operations
Intelligence
Corporate
Performance Reference Model
assesses performance of investments against outcomes
Business Reference Model
defines business areas to promote collaboration,
Services Reference Model
classifies services according to business & performance
Data Reference Model
enables data sharing and re-use
Technical Reference Model
categorizes standards and technologies against services

Figure 60. Integrated Defence Architecture Reference Mode (Purcell 2009).


Strategic Outcomes
Defence of Australia and its National Interests
Mission Results
Business Results
Conduct of missions & operations that deliver military effects
Establish & maintain public confidence & trust in Defence capabilities
Reach and maintain the required levels of force capability, readiness & preparedness
Establish & maintain public confidence & trust in the management & administration of Defence's assets
Command & Management
Organisation
Major Systems
Supplies
Personnel
Collective Training
Facilities
Support
Business Assets & Inputs (Defence's Fundamental Inputs to Capability)
Extended Enterprise - Government, Coalition Partners, Allies, Industry
Mobile & Deployed Assets
Fixed Assets
Business Service Consumer
System Service Providers
Intelligence
Warfighter
Corporate
Warfighter
Corporate
Intelligence
Illustrative Cross-Domain Common Client Services
BI & Reporting
Search & Discovery
Presentation
Illustrative Standardised Systems & Services
Integrated C4ISREW Services
Federated Logistics Services
Mission / Capability Specific Services
Personnel Management
Financial Management
Logistics Management
Integrated ISR Management Services
Federated Analytics Services
Discipline Specific Analytics Services
Illustrative Cross Domain Common Services
Enterprise Collaboration
Service Monitoring
Backup & Recovery
BPM & Workflow Management
Identity & Security Management
Service Registry
Content Management & Metadata
Gateway/ Interface Management
Specialised Systems
Specialised Systems
Domain Specific Data
Warfighter
Enterprise Data Hub (illustrative)
Command & Control
ERP
Defence Health
ISR
Geospatial
Planning
Reporting
Domain Specific Data
Intelligence
Standalone Networks
Single Core Network
Multiple environments
Standardised & Consolidated Technology Platforms
Data Centres
Technologies
Standalone Networks
Single Core Network
Multiple environments
Unified Network Communication Infrastructure

Security: Enterprise security architecture will employ "Defence-in-depth" controls to enable dynamic response to changing threats and will ensure that security is considered as a core part of service and system solution deliver. The architecture will support a trusted environment for integrated data sharing across Defence's architecture domains and with extended enterprise to meet the needs of "need-to-share" and balance "need-to-know" requirements. The architecture will enable seamless security to enhance the user experience and maintain accountability and compliance with legal and regulatory requirements

Performance Clear articulation on the cause-and-effect relationship between inputs, outputs, outcomes & ICT to deliver "line of sight" traceability to justify investments. Enable key decision-makers to understand how, and to what extent, key inputs are enabling progress toward outputs and outcomes

Business Unification of Warfighter, Corporate & Intelligence to create an integrated "top-down" architecture supporting the extended enterprise. Business architecture including business functions, processes and services

Systems and Services Standardised systems and services within each architecture domain supported by a set of cross-domain common services. Specialised services and systems to support unique requirements.

Data: Enterprise shared data hub built on data discovery, sharing connectivity, enhancement and dissemination aspects to support "need-to-inform" and increase collaboration

Technology: Standardised and consolidated infrastructure platforms driving cost-effectiveness and agility. Single core common network delivering secure access to data and services through multiple logical security environments

Figure 61. *Integrated Defence Architecture Representation (Purcell 2009).*

Although this EA approach uses certain principles drawn from TOGAF, the initial approach does not use a single integrated data model encompassing the respective views as described previously in Table 2. From a modelling perspective, a non-architectural approach has been taken where a different model is created for each view (i.e. product or artefact).

A suite of five EA reference models, as previously described in Section 3.3.5 has been crafted under the auspices of the IDA to provide a taxonomy for sorting, storing and understanding the information and artefacts that comprise the EA for Defence. Drawn from the AGAF, these include:

- a Business Reference Model (BRM);
- a Performance Reference Model (PRM);
- a Data Reference Model (DRM);
- a Services Reference Model (SRM); and
- a Technical Reference Model (TRM) (AGAF 2009).

The architecture reference models are not intended to specify individual ICT requirements, but are intended to provide an abstract representation of the broad requirements of Defence's strategic objectives to guide individual projects and business areas (IDA BRM 2011). They are not intended to drive a particular technical process.

The IDA Business Reference Model (BRM) provides a list of definitions where EA is defined as the organising logic for business processes and infrastructure reflecting the standardisation of a business operating model; the enterprise is the highest level (typically) of description of an organisation and typically covers all missions and functions.

Whole of Defence has been partitioned into three business areas:

- Warfighter,
- Intelligence, and
- Corporate.

Each business area contains multiple supporting business domains, which can be further divided into sub-domains. For example, the Warfighter business area comprises seven domains:

- Joint,
- Land,
- Air & Space,
- Maritime,
- Intelligence, Surveillance and Reconnaissance (ISR),
- Coalition, and
- Capability Integration.

Domains provide a high-level view of services and capabilities that support enterprise and operational processes and applications. Capability in the context of a domain, is a business capability, which is an ability that an organisation, person, or system possesses, and typically requires a combination of organisation, people, processes and technology to achieve. This definition is different to that used in the DCDH, with significant implications for governance, so the context of terminology is important when interpreting guidance.

The term EA used in the context of the SIE and the IDA differs from the term architecture used in the context of the AUSDAF. The SIE introduces the notion of EA as "a means for aligning Defence capabilities and outputs with Defence's strategic drivers" with the term EA defined in the IDA BRM in terms of the organising logic for processes and infrastructure. The IDA BRM seeks to provide a common structure as a basis for capability planning, and the development of consistent enterprise processes (IDA BRM 2011).

The IDA BRM goes further to suggest that EA thus establishes a linkage between EA initiatives, the Strategy Framework, and the capability development process. Reference is made to a companion document, *The NCW Integration and Implementation Strategy (NCWIIS)* (NCWIIS 2010). The NCWIIS was to provide a methodology for the development of integrated capabilities; however, it still was not implemented some three years later, thus its future applicability is uncertain.

The IDA BRM also provides a separate definition of architecture which accords with the IEEE-1471 definition of architecture from a SW engineering perspective, which is an entirely different notion of architecture from that of EA.

Significantly from an SE and DCDH perspective, the IDA BRM does not include definitions for the terms "component", "interface", "integration", or "system", despite the liberal use of these terms throughout IDA related documentation. It will therefore be difficult to obtain common understanding between practitioners of the different communities of interest if they do not appreciate the significant differences in semantics of commonly used vernacular.

Significantly, each business domain has taken a different implementation approach, with difference governance regimes applied; there is no overarching guidance on implementation of the IDA at the highest level. Notably, ICT investment that occurs outside the DCP is not subject to the capability development process nor the same EA governance.

In terms of EA governance, little specific guidance is prescribed; individual ICT projects may call for one of more meetings with invited stakeholders to review architectural artefacts, known as architecture review meetings (ARMs). ARMs are called at the discretion of each individual project across the three respective EA domains. The purpose, timing, scope and invitees for each ARM is also determined at the discretion of the respective project. Attendance is not specifically resourced. Individual projects may also prepare additional documentation to describe their respective initiatives, but these are typically crafted in the form of policy guidance rather than in engineering form.

There is, at times, significant overlap of responsibilities and acquisition activity for those DCP Projects with a significant ICT contribution, delivering new or upgraded capability into service that is deemed part of the IDA. However, it is not apparent how the IDA initiative integrates with the mandated DCDH capability development process.

The IDA concept developers stated their expectation that the IDA will be populated with data generated by the respective MCE Projects as they progress through the capability development process, supplemented by data provided by the respective Capability Managers.

However the form and format of data required, and future resourcing and management requirements to support are dependent on future policy directives and funding pertaining to the IDA. This initiative has not been explicitly resourced within the DCDH capability development process.

Key notions which are integral to a possible future data-centric EA approach include transitioning the conceptual notions of EA into real-world instantiations, and addressing numerous matters including:

- data definitions;
- provision of modelling and analysis guidance (how to and what for);
- governance responsibility;
- risk management;
- resourcing responsibility;
- configuration management responsibility;
- data integrity management (much of which is contextual in nature);
- clarification of vocabulary and semantics;
- quality assurance;
- accessibility of data repositories;
- ownership and management of data;
- articulation of technical processes;
- tool availability and suitability;
- skill and training requirements; and
- other support infrastructure.

All merit further consideration if an enterprise wide data-centric approach is to taken across whole-of-Defence. These considerations are also pertinent to MBSE.

11. MBSE Origins and Concepts

11.1 MBSE Impetus

The document-based approach described in Section 5.3.4 can provide significant rigour in supporting a SE process. However, it has limitations. Assessing the completeness, consistency, and traceability of relationships between requirements, analysis, design, construction and test can be difficult if the information is distributed across multiple documents, particularly for the larger-scaled and more complex systems. This can make it difficult to understand different aspects and to assess trade-offs and change impacts.

This in turn can lead to poor synchronisation between system-level requirements and the system design, and with lower level hardware and software detail design. This can also make it difficult to maintain or reuse the information for evolving systems and for variants of the system.

These issues with the document-based approach have spurred the development of alternate, model-based approaches, drawing from the SW engineering discipline, to facilitate improved communication between stakeholders, and improve specification and design precision, system design integration, and reuse of system artefacts. Instead of documentation, the output from a model-based SE approach is a coherent model of the system, where the emphasis is placed on evolving and refining the system model using model-based methods and tools (Friedenthal et al. 2008).

11.2 MBSE Origins

A number of MBSE methodologies have been formalised since the 1980s, coinciding with the ready availability and widespread adoption of computer workstations and desktop computing into the mainstream engineering development environment. It has also corresponded with the emergence of SW engineering as a distinct sub-discipline within the engineering discipline; SW no longer being the sole purview of the originating computing science discipline.

These advances have also spurred the development of commercially available Computer Aided Software Engineering (CASE) tools, the precursors to the most recent generation MBSE tools which are now widely available, which are themselves SW applications hosted on a main-frame or desktop computing environment.

As mentioned previously in Section 2, MBSE is a catch-all phrase encompassing a number of different methodologies supporting the SE discipline in a “model-based” or “model-driven” context[73]. A variety of computer-based SE tools can now be purchased to provide SE process support, as described in Table 8. An overview of various tool vendors supporting engineering development is provided in Appendix A.

[73] (Estefan 2008) notes that some authors use the term Model-Driven System Design (MDSD) interchangeably with Model-Based Systems Engineering (MBSE), despite subtle differences. The term MDSD is used synonymously with MBSE in the Estefan report.

Table 8. Key Systems Engineering Tools.

Type of Tool	***Key Features***
Requirements Management	End-to-end traceability of individual requirements from original source to lower tier mission, system, subsystem and component requirements.
Verification Cross-Reference Management	End-to-end traceability of verification activity against each individual requirement.
Model-based Engineering Development	Modelling of system requirements, system functionality, trade studies, target system implementation, verification and validation activity, with end-to-end traceability.
Change Management and Configuration Management	End-to-end change management and configuration management of individual items and associated documentation across an entire system, and across its life cycle stages.
Automated Documentation Generation	Generation of pre-formatted project documentation, including specifications, drawings, V&V procedures, and project management reports.
Integrated Systems and Software Engineering	End-to-end traceability and management of all tiers of engineering and project activity, including embedded software development and automated code generation and verification.

While process, method, and tools are integral concerns in the SE development environment, MBSE is differentiated by its focus on a particular method of enquiry, and the prominent role of the tool to support the method and drive the process. The process and method are therefore enabled by the capabilities of the respective system modelling tools.

Despite the prevalence of model-based design methodologies in HW and SW engineering over the extended period, the extension of these methodologies to the much broader systems level, specifically attributable within a SE context, has only occurred within the last decade.

This ostensibly appears to have been spearheaded firstly by OMG member organisations, then subsequently taken up by INCOSE, through their sponsorship and organisational support for crucial initiatives including the initial codification of UML and subsequent support for SysML development.

Vendor response in making available new generation modelling environments based on UML and SysML has been key to the wide-spread interest, both in industry and Government, of the potential of new generation MBSE tools to support systems analysis and design activities. The transition appears to have been assisted by, (and sometimes confused by), the emergence of the military and commercial EAF, which are variously used to either complement or replace formal SE process activities, particular in the earlier stages of systems conceptualisation.

In the last decade in particular, SE, SW engineering, business, and EA modelling tool vendors have expended considerable effort to expand their product suites to support a multitude of graphical modelling techniques, primarily within the object-oriented paradigm[74], promoted as supporting both EA and business practice, and SE and MBSE concepts.

Some specific "model-based" or "model-driven" methodologies relating to the SE discipline are now examined in more detail to highlight the differences between the SE and the non-SE perspectives.

11.3 MBSE Tool Capabilities

The new generation MBSE tool product suites typically comprise:

- databases to store the required set of information (i.e. model repository);
- a means to define, tag and structure the way data is stored in the database (typically a using schema);
- a means of querying the database in a structured way and presenting the query output in a specific display format; and
- a user-friendly graphical interface to allow users to input and modify their data in a structured way.

The system model is created using the MBSE tool incorporating system specification, analysis, design, and test information. The model consists of elements that represent specific information types including requirements, design elements, test cases, and design rationale, as well as their interrelationships. The primary use of the model is to design a system that satisfies system requirements and allocates the requirements to the system's components. The model elements and interrelationships are stored in the model repository where they can be separately created, edited, modified and deleted as the design evolves (Friedenthal 2008).

Automated document generation using pre-defined templates is also inherent functionality in these tools, depending on the enterprise architecture framework or SE methodology supported. This allows the model repository information to be viewed from different perspectives, either as diagrams, tables, drawings or text reports, generated by querying the model repository.

The new generation MBSE tools typically provide the ability to store and visualise the list of system requirements comprising the top-tier specification in a database as a requirements model. These can be analysed and expanded to synthesise lower-tier requirements, linked to the originating requirements, and also stored in the database forming a requirements tree. Complete traceability between the respective requirements is therefore inherent throughout the database forming the requirements model.

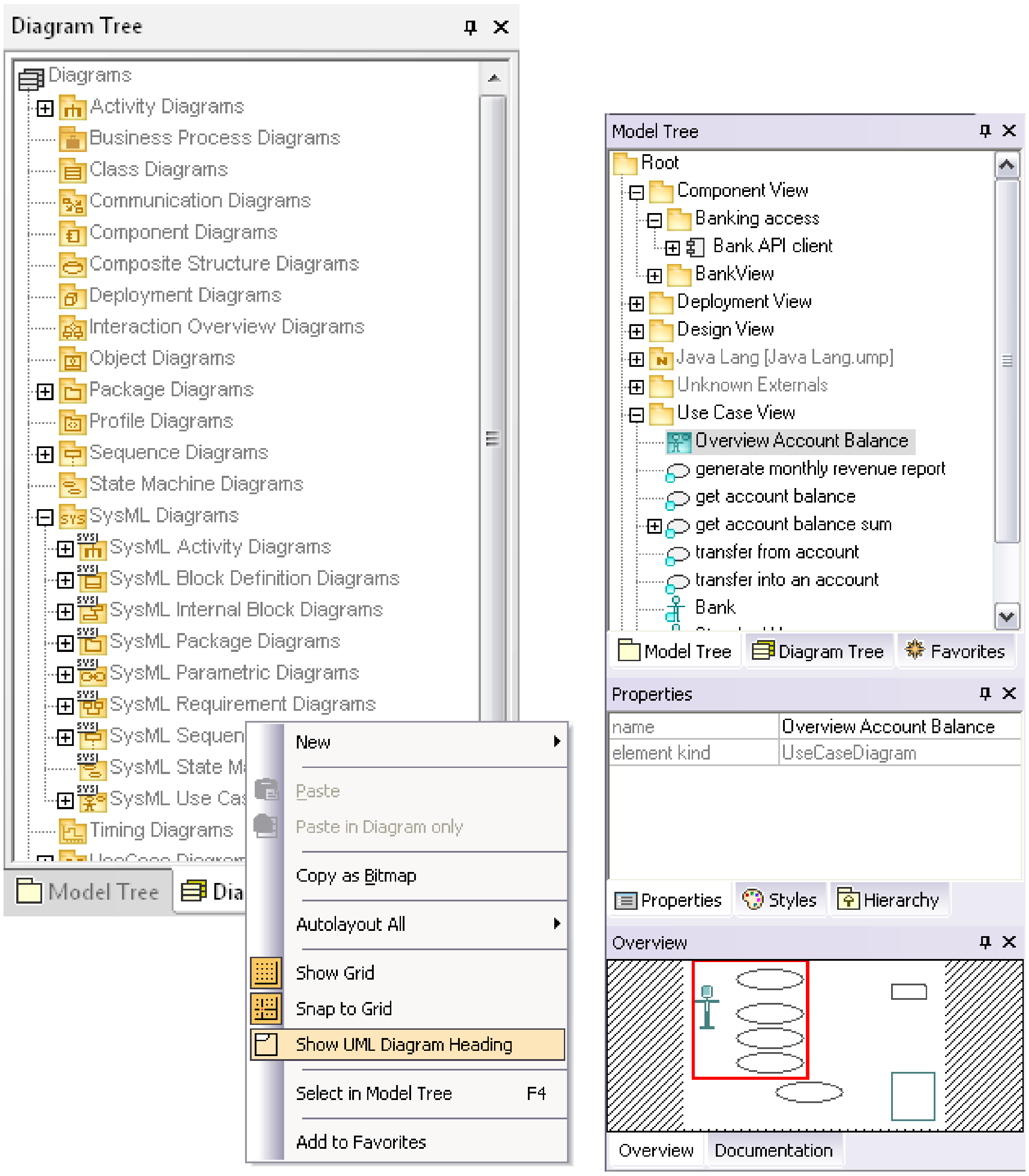
[74] The CORE SE tool from Vitech Corporation is one notable exception. Vitech offer a proprietary system design language (SDL) to support SE activities within the CORE SE tool. The CORE tool supports tailoring of its SDL to create a user-specific profile as a subset of the SDL. Vitech sell DoDAF and MODAF plug-ins for the CORE tool, which essentially comprises a tailored subset of the SDL supporting the DoDAF and MODAF graphical modelling capabilities. [online] URL: http://www.vitechcorp.com/.

The functional allocation and detail design of deeply nested subsystems, equipments and finally components are supported in the model; visualised using nested System Block Diagrams. System-to-system behaviour and intra-system dynamics are defined and elaborated using Interaction Diagrams, Activity diagrams, and State Charts to visualise the dynamic behaviour, and code can be generated to implement the functionality in a variety of languages.

A typical user interface in a SysML modelling tool illustrating toolbar access to the different diagrammatic types is shown in Figure 62.

An example of the user interface presented in the tool "Enterprise Architect" when modelling in UML to produce SW code is provided in Figure 63.

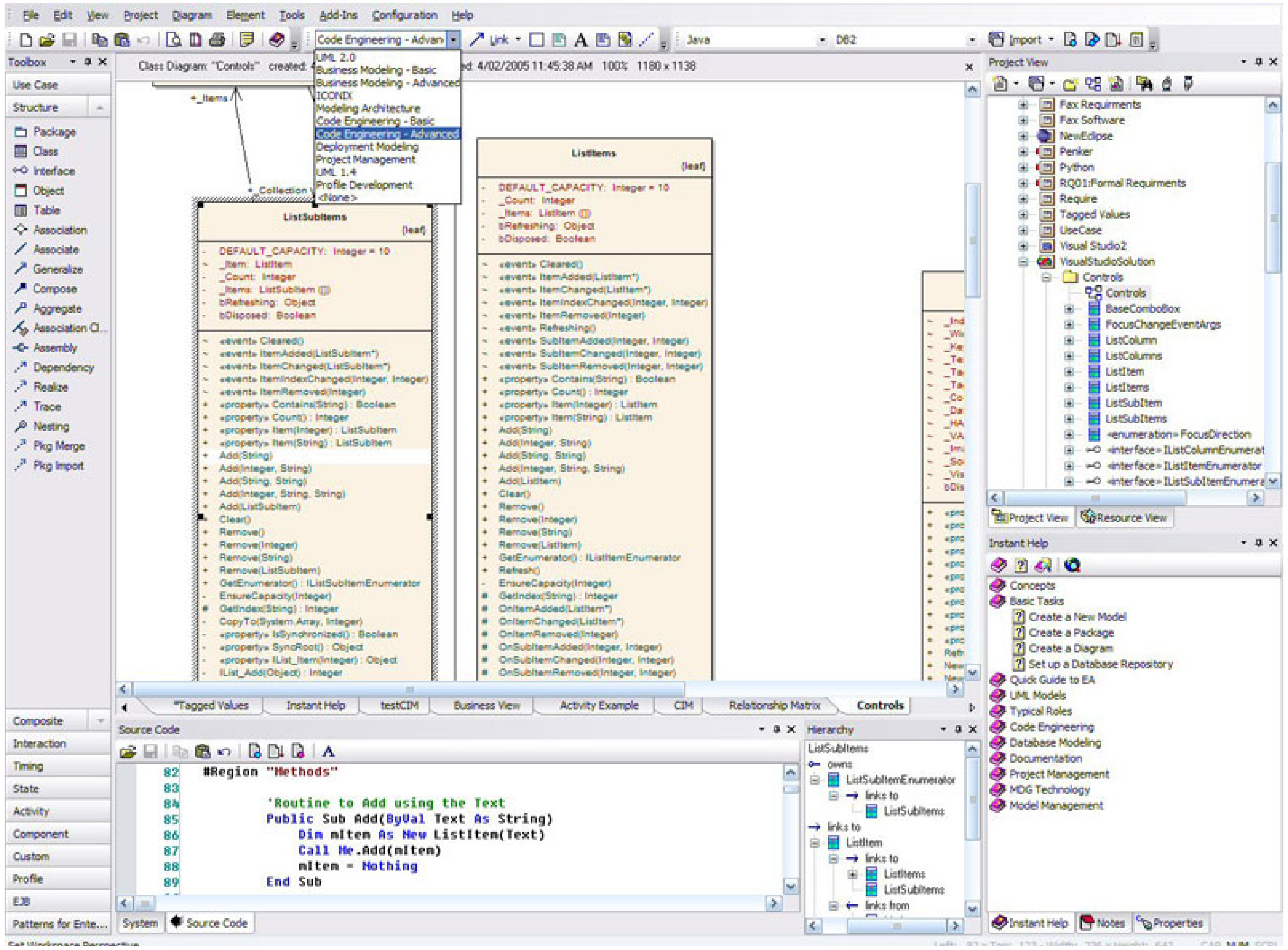
An example of documentation customisation is provided in Figure 64, where a RTF style template editor allows the creation and editing of custom RFT templates to define the required output RFT documentation. An example of report generation capabilities is illustrated in Figure 65.



name	Overview Account Balance
element kind	UseCaseDiagram

Figure 62. Altova UModel Tool SysML Graphical User Interface

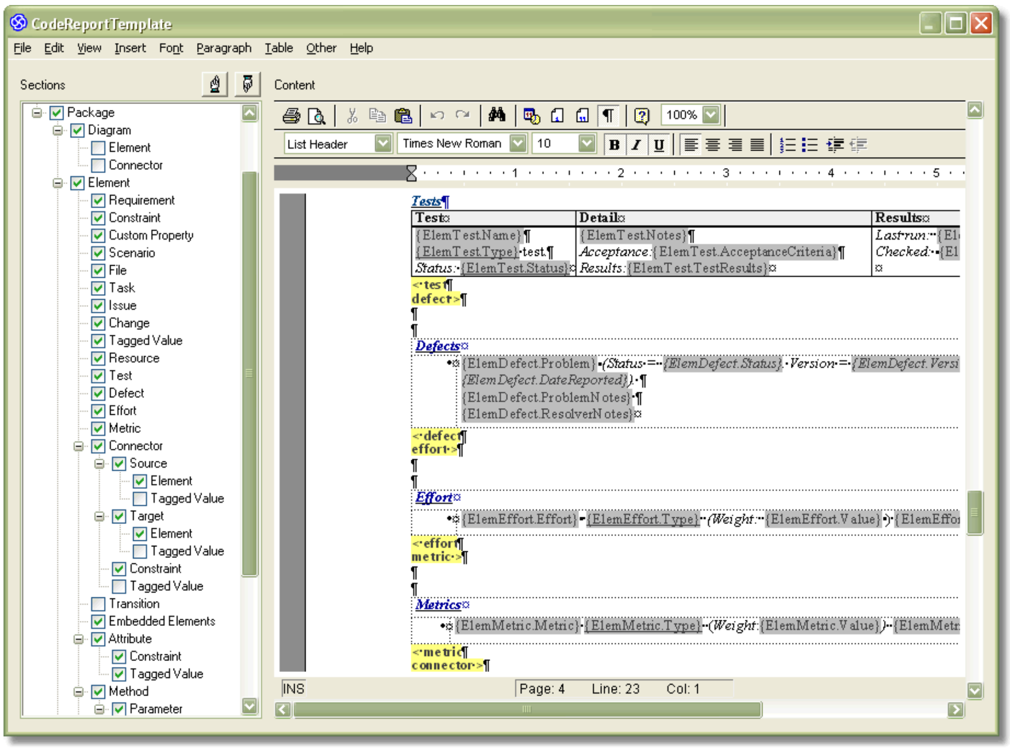
([online]: URL http://www.altova.com/umodel/sysml.html).



```
#Region "Methods"

        'Routine to Add using the Text
        Public Sub Add(ByVal Text As String)
            Dim mItem As New ListItem(Text)
            Call Me.Add(mItem)
            mItem = Nothing
        End Sub
```

Figure 63. Example of Automated Code Generation from a UML model representation using the SparxSystem's Enterprise Architect Tool ([online] URL: http://www.sparxsystems.com/platforms/software_development.html).



Sections:

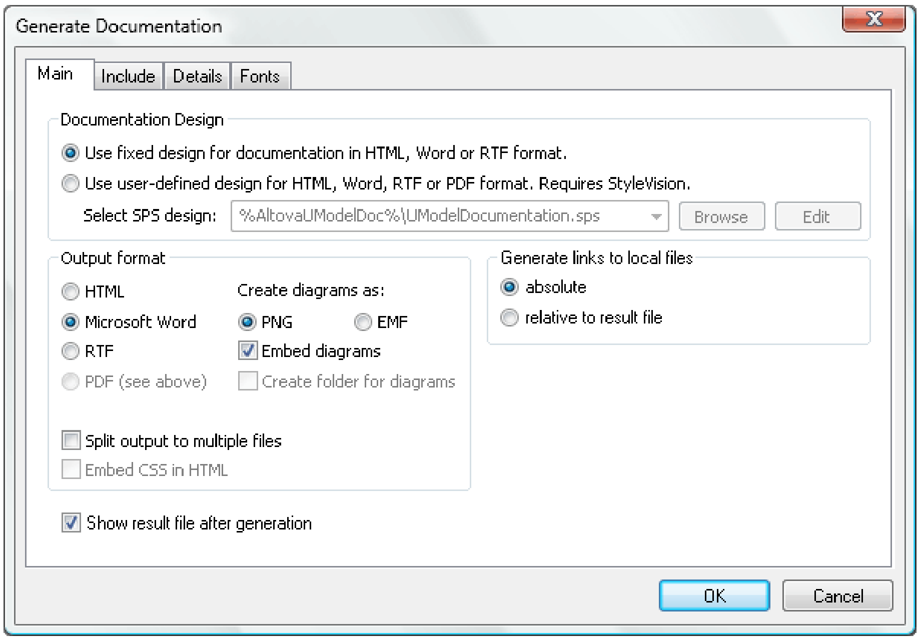
- [✓] Package
  - [✓] Diagram
    - [ ] Element
    - [ ] Connector
  - [✓] Element
    - [✓] Requirement
    - [✓] Constraint
    - [✓] Custom Property
    - [✓] Scenario
    - [✓] File
    - [✓] Task
    - [✓] Issue
    - [✓] Change
    - [✓] Tagged Value
    - [✓] Resource
    - [✓] Test
    - [✓] Defect
    - [✓] Effort
    - [✓] Metric
    - [✓] Connector
      - [✓] Source
        - [✓] Element
        - [ ] Tagged Value
      - [✓] Target
        - [✓] Element
        - [ ] Tagged Value
      - [✓] Constraint
      - [ ] Tagged Value
    - [ ] Transition
    - [✓] Embedded Elements
    - [✓] Attribute
      - [✓] Constraint
      - [✓] Tagged Value
    - [✓] Method
      - [✓] Parameter

Content:

```
Tests¶
Test¤                      | Detail¤                                          | Results¤
{ElemTest.Name}¶           | {ElemTest.Notes}¶                                | Lastrun:·{El
{ElemTest.Type}·test.¶     | Acceptance:{ElemTest.AcceptanceCriteria}¶        | Checked:··{El
Status:·{ElemTest.Status}¤ | Results:{ElemTest.TestResults}¤                  | ¤
<·test¶
defect>¶
¶
¶
Defects¤
    •¤ {ElemDefect.Problem}·(Status·=··{ElemDefect.Status}.·Version·=·{ElemDefect.Versi
       {ElemDefect.DateReported}).·¶
       {ElemDefect.ProblemNotes}·¶
       {ElemDefect.ResolverNotes}¤
<·defect¶
effort·>¶
¶
¶
Effort¤
    •¤ {ElemEffort.Effort}·{ElemEffort.Type}··(Weight:··{ElemEffort.Value}·)·{ElemEffor
<·effort¶
metric·>¶
¶
¶
Metrics¤
    •¤ {ElemMetric.Metric}·{ElemMetric.Type}··(Weight:{ElemMetric.Value})··{ElemMetr
<·metric¶
connector·>¶
```

Figure 64. Example of a Report Template in a MBSE Tool

([online] URL: http://www.sparxsystems.com.au/resources/rtf/template_editor.html).



Documentation Design

- [✓] Use fixed design for documentation in HTML, Word or RTF format.
- [ ] Use user-defined design for HTML, Word, RTF or PDF format. Requires StyleVision.

Select SPS design: %AltovaUModelDoc%\UModelDocumentation.sps

Output format

- [ ] HTML
- [✓] Microsoft Word
- [ ] RTF
- [ ] PDF (see above)

Create diagrams as: [✓] PNG [ ] EMF

- [✓] Embed diagrams
- [ ] Create folder for diagrams
- [ ] Split output to multiple files
- [ ] Embed CSS in HTML

Generate links to local files

- [✓] absolute
- [ ] relative to result file

[✓] Show result file after generation

Figure 65. Example of Automated Documentation Generation using the Altova UModel UML Modelling Tool

([online] URL: http://www.altova.com/umodel/uml-project-documentation.html).

11.4 Model-Based Design

One of the earliest examples of an MBSE approach is the Model-based Design Method, a mathematical and visual method useful to address certain classes of problems, typically to assist with the design of real-time SW associated with complex control, signal processing and communication systems.

The Model-based Design Method typically includes the following steps:

- Design and develop a functional model of the SW to meet the specification, typically expressed in mathematical form.
- Analyse and synthesise system solution building blocks from the functional model to meet the requirements.
- Simulate the solution building blocks and verify consistency with the specification.
- Integrate the solution building blocks within the system model and verify the system model meets the specification.
- Develop prototype code based on the system model (or auto-generate code).
- Verify and validate the system solution meets the specification[75].

This method is still widely used today, for example, for simulation and subsequent implementation of real-time embedded SW-based functionality such as control algorithms in mechatronic systems, utilising programmable microprocessors and Field Programmable Gate Arrays (FPGAs).

11.5 Model Driven System Design

In recent years, members from both OMG and INCOSE have strongly supported the notion of Model-Driven System Design (MDSD) to support SW system development. They have also strongly promoted the virtue of replacing hard copy documentation with the notion of holding the information in soft copy form in an information repository, in the form of a SW model (Baker et al. 2000).

The notion of MBSE is predicated on the idea that system design data can be regarded as a collection of data elements with dedicated relationships. The advent of more powerful desktop computing environments and relational database SW has enabled the development of dedicated tools to capture and more easily manipulate the huge quantities of data associated with large scale and complex system development. The tools also provide a reporting capability that can generate documentation in specific formats by using database scripts to populate documentation templates with information stored within the database.

This allows requirements allocation, functional allocation and physical allocation, for example, to be undertaken by setting up the respective links between the data elements in the database. Corresponding requirements, functions, and physical architecture details are

[75] [online] URL: http://en.wikipedia.org/wiki/Model-based_design;
URL: http://www.lhpsoftware.com/modelbaseddesign;
URL: http://machinedesign.com/article/model-based-design-for-mechatronics-systems-1121

stored as data elements within the database. Where tools allow the capture of the entire functional and physical architecture of a system, this potentially allows the entire system solution to be captured electronically in the database Thus database links can be used to establish complete traceability from specification through to implementation and V&V. (Halligan 2011).

Application of a consistent model-based methodology for SW implementation over its life cycle is seen to offer numerous benefits, including better traceability, less ambiguity, and improved rigor compared with textual approaches, particularly in relation to its ability to support continuous assessment of consistency between requirements and design implementation (Baker et al. 2000).

Determining the impact of a proposed change to one of the requirements, functions, or physical elements is also easier than using hard copy documentation by being able to navigate electronically through the database via the links to determine the affected neighbouring data elements.

MDSD can also facilitate easier re-use of standard models and models with recurring design patterns. These in turn proffer increased productivity, reduce development time, lower development costs, increase SW reliability, and improve compatibility between SW implementations, as well improve ease of governance. Using data models and graphic notation also constrains the representation of a system design to an agreed, predefined set of engineering concepts with a reduced learning curve that aids common understanding (Estefan 2008).

Baker asserts the model-driven approach to system design is fundamentally consistent with the classical notion of SE as described in IEEE 1220, with similar basic life cycle stages as shown in Figure 66 (Baker et al. 2000).

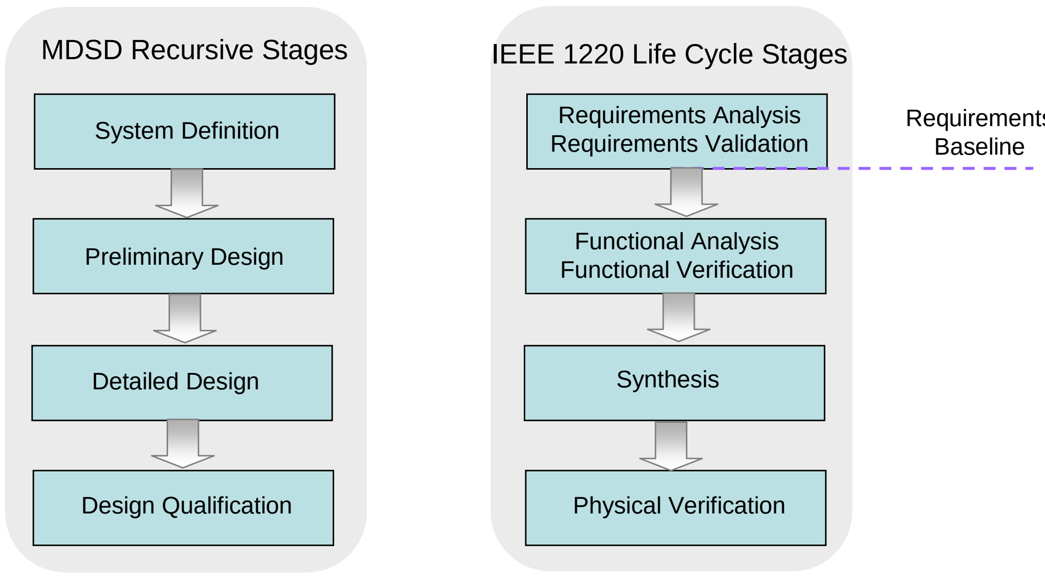


Figure 66. MDSD Development Phases of a Project.

The activities within each of the project phases are represented by Baker as basic sub-processes as shown in Figure 67, which can be repeated as many times as necessary to achieve the development objectives of a project.

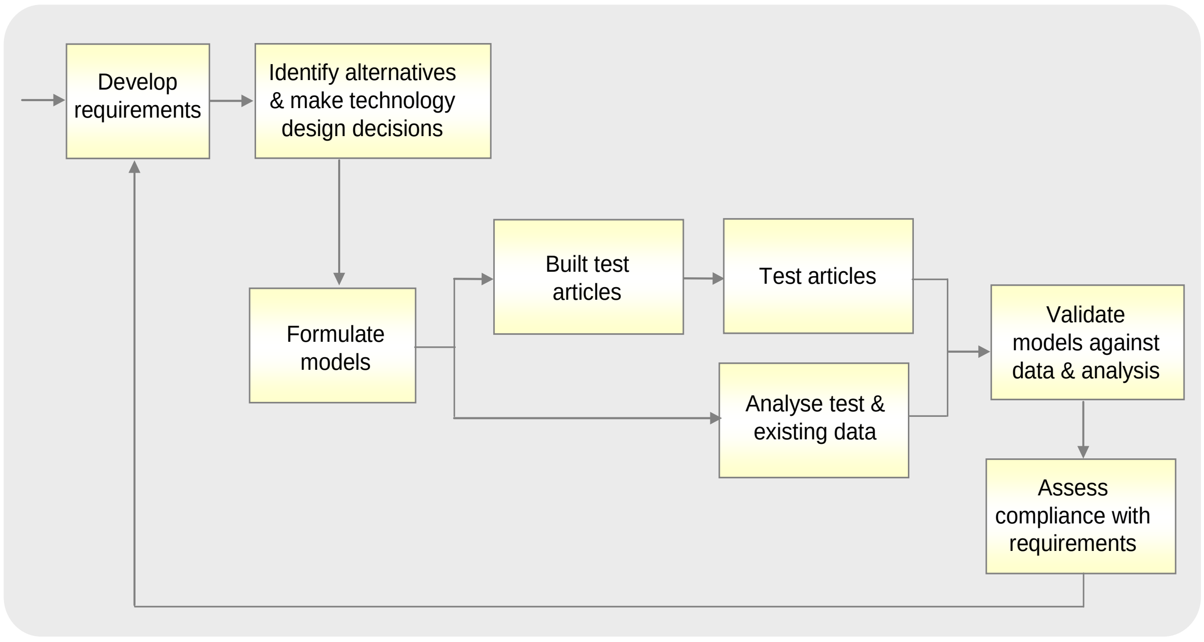

Develop requirements
Identify alternatives & make technology design decisions
Formulate models
Built test articles
Test articles
Analyse test & existing data
Validate models against data & analysis
Assess compliance with requirements

Figure 67. Typical MDSD Recursive Sub Processes within a Development Phase

In the case of MDSD, the requirements baseline, the functional architecture, and the physical architecture are revealed in increasing detail in the model as the design and implementation is progressed. These are informed by successive trade studies, synthesis, and V&V activity as necessary, towards product or system completion. Logical entities are first developed, then mapped across to physical implementations.

MDSD is characterised by the following activities where the model is the central repository for all information used to drive the system solution to its "build-to" baseline (Baker et al. 2000):

- **System Definition**
  - Undertaking of system definition activity to initiate system design, including:
    - analysis of customer needs and constraints in the context of both the external and internal environments, and
    - synthesis of the system in broad terms relating to boundaries, functions, and performance requirements (both environmental and non-environmental)

    such that the system can be adequately described in terms of a set of requirements that will drive the implementation of a suitable system solution.
  - Compilation in executable form of:
    - system, product specifications,
    - system, product and subsystem interface specifications,
    - preliminary subsystem specifications.

  - Production of:
    - machine-readable system baseline
    - machine readable preliminary "subsystem design" to baseline
  - Completion of a system performance model with sufficient detail to respond to the system requirements.
  - Completion of technical reviews, including system model validation that the system model of the "design-to" baseline is consistent in terms of cost, schedule and technical performance requirements.
- **Preliminary Design**
  - Undertaking of preliminary design activity to initiate subsystem design activity, including:
    - analysis of system requirements and constraints, and
    - synthesis or selection of subordinate subsystems.
  - Compilation in executable form of :
    - subsystem specifications,
    - preliminary lower-tier component interface specifications,
    - preliminary lower-tier component specifications.
  - Production of:
    - machine-readable subsystem baseline
    - machine-readable preliminary "component design" to baseline.
  - Completion of subsystem performance models
  - Completion of technical reviews including subsystem model validation such that the models of the "design-to" baseline show preliminary compliance with specifications.
- **Detailed Design**
  - Undertaking of detailed design activity to complete subsystem design and models down to the lowest component.
  - Compilation in executable form of:
    - component specifications
    - component interface specifications
  - Production of :
    - machine-readable "build-to" baseline for each component.
  - Completion of technical reviews including component model validation such that the models show satisfactory preliminary compliance with performance specifications and satisfactory final compliance with design constraints.
  - Completion of developmental tests and analyses to validate the performance models such that they show that production articles built to the detailed design will be compliant with the specifications,
- **Design Qualification**
  - Undertaking of design qualification activity, including validation of performance models against test data taken on test articles manufactured according to the "build-to" baseline.
  - Completion of technical reviews such that the validated performance models show satisfactory compliance with performance specifications. (Baker et al. 2000).

11.6 Model Driven Engineering

11.6.1 Graphical Modelling Techniques

The recent notion of MBSE has evolved from a SW development methodology known as Model-driven Engineering (MDE). MDE refers to a range of development approaches in SW engineering that uses graphical modelling techniques as the primary form of expression. MDE is characterised by using graphical techniques such as IDEFx diagrams to describe different aspects of the problem domain and the solution domain based around the notion of a SW architecture (Schmidt 2006), (Kent 2002)[76].

SW architecture, as described in IEEE 1471, has now become a widely accepted conceptual basis for developing non-trivial SW. The SW architecture is a depiction of a SW program or computing system within a tool, stored in a structured database in a designated format as a model, and documented in artefact form, such that it articulates the primary qualities of the system in order to design and implement the system[77].

For many classes of problems, SW models are built by constructing and documenting the SW architecture to a certain level of detail using graphical techniques. Code is then written by hand as a separate process step. However, for certain classes of problems, complete models can be built including executable actions. Thus, code can be generated from the models ranging from system skeletons to complete, deployable SW products.

Some tools also have the ability to import SW code into the tool and reverse engineer the code to convert it to diagrammatic form for ease of editing and verification, allowing the model and the code to remain synchronised throughout the life cycle of the SW.

With the emergence of object-oriented SW languages such as C++ and Java, and the formalisation of UML and SysML as international modelling standards under the auspices of OMG, MDE has become very popular today with a wide body of practitioners and supporting tools. More advanced types of MDE have flourished, leading to the development of a number of defacto industry standards supported by the respective tool vendors and/or published under the auspices of OMG.

Importantly, these methodologies encourage consistent application of methodologies and consistent interpretation of results; the continued evolution of MDE providing an increasing focus on SW architecture and process automation, rather than on code implementation, to arrive at the final solution.

11.6.2 Rational Unified Process (RUP)

The Rational Unified Process (RUP) is one such object-oriented SW engineering methodology that was instrumental in developing and using numerous graphical diagramming techniques and the accompanying vernacular that have subsequently been codified by OMG into the SW modelling language UML.

[76] [online] URL: http://en.wikipedia.org/wiki/Model-driven_engineering.

[77] Software Engineering Institute, Carnegie Mellon, [online] URL: http://www.sei.cmu.edu/architecture/?location=secondary-nav&source=1358).

The RUP was originally conceived by Rational Software[78] circa 1996 as illustrated in Figure 68. The set of diagrams formally published as UML shows the combined influences from Booch, Jacobson, who developed Objectory Object-oriented Software Engineering (OOSE), and Rumbaugh, who developed the Object Modeling Technique (OMT). All three came together at Rational Software, who spearheaded the co-development of UML in parallel with the development of the RUP to facilitate clear communication of requirements, architectures, and design of SW-intensive systems.

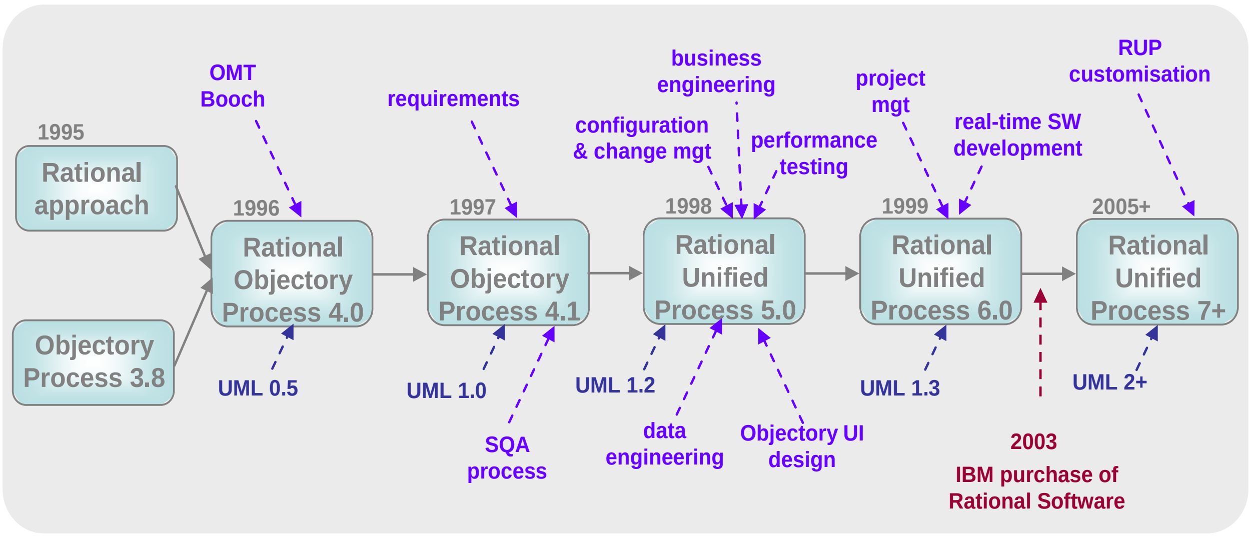


Figure 68. Rational Unified Process Evolution Timeline (adapted from Rational 2001).

The RUP was conceptualised as a project-oriented SW engineering process framework, with an accompanying tool suite to automate large sections of the process. This sought to support the efficient, cost-effective, and high quality production of large-scale bespoke SW-intensive systems to meet the needs of its intended end users, but within a predictable schedule and budget.

Team productivity was said to be enhanced by providing each team member easier access to a common knowledge base that contained guidelines, templates and tool mentors for all critical development activities. By having all team members accessing the same knowledge base, regardless of whether their focus was requirements, design, test, project management, configuration management or quality management, the RUP offered a common language, process and view of how to develop SW using best practices in SW development.

A key feature of the RUP is the focus on creation and maintenance of semantically rich representations of the SW system under development (i.e. models in the common knowledge repository), rather than producing large quantities of hard copy documentation.

[78] The RUP was originally developed by software tool developer Rational Software. Rational Software was purchased by IBM in 2003 and is now marketed as IBM Rational Software [online] URL: http://www-01.ibm.com/software/rational/.

RUP is based on the following best practice tenets:

- Develop SW iteratively
- Manage requirements
- Use component-based architectures
- Visually model the SW
- Verify the SW quality
- Control changes to SW (Rational 2001).

Process concepts in RUP are represented using four primary modelling elements as shown in Figures 69(a) (b) and (c):

- Workers – the "who" – defined in terms of the behaviour and responsibilities of an individual or group of individuals in a team;
- Activities – the "how" – an activity of a specific worker is a unit of work that an individual in that role may be asked to perform;
- Artefacts – the "what" – is a piece of information that is produced, modified, or used by a process;
- Workflows – the "when" – is a sequence of activities that produces an observable value.

These combine to describe a process in terms of who is doing what, when and how (Rational 2001).

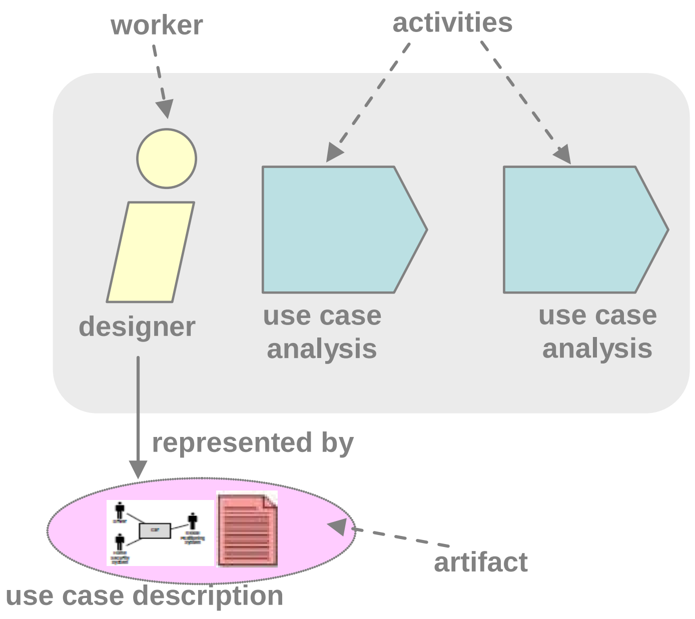


Figure 69 (a). Worker, Activity and Artefact Representations.

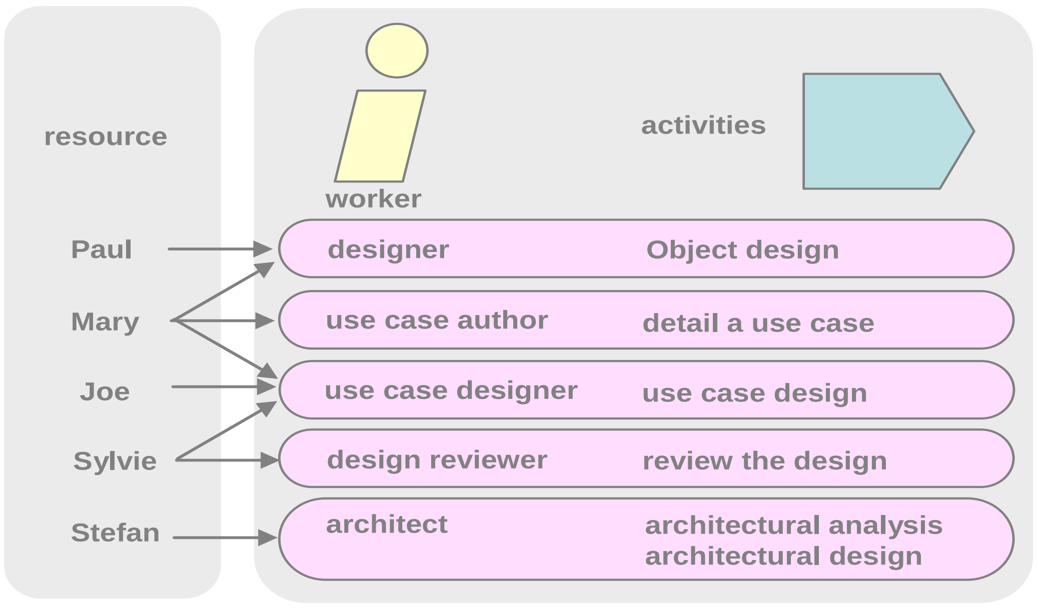
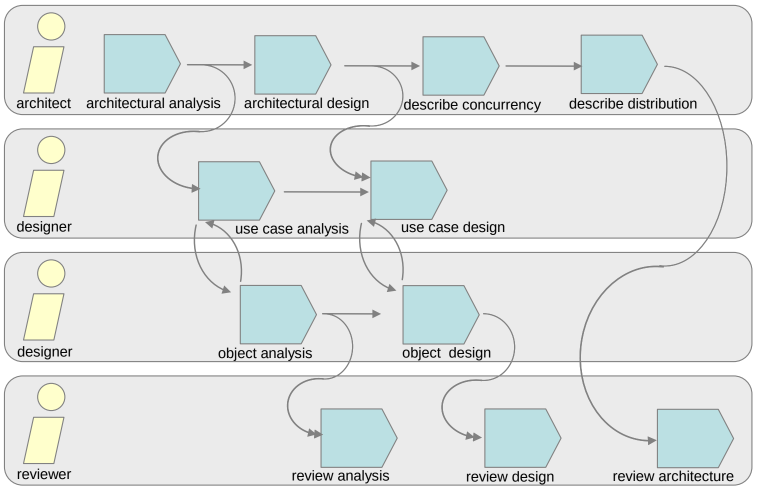


Figure 69(b). Worker and Activity Representations.



- Business modelling workflow
- Requirements workflow
- Analysis & Design workflow
- Implementation workflow
- Test workflow
- Deployment workflow
- Project management workflow
- Configuration & change management workflow
- Environment workflow

Figure 69(c). Workflow Representation.

The RUP framework essentially spans two dimensions as shown in Figure 70:

1. The *horizontal dimension*, expressed as a function of time, unveils the life cycle aspects of the process as it unfolds. This dimension represents the dynamic aspects of the process, expressed in terms of cycles, phases, iterations and milestones. The SW lifecycle is broken into cycles, each cycle working on a new generation of the product.

   The RUP divides each development cycle into four consecutive phases:

- o Inception
- o Elaboration
- o Construction
- o Transition.

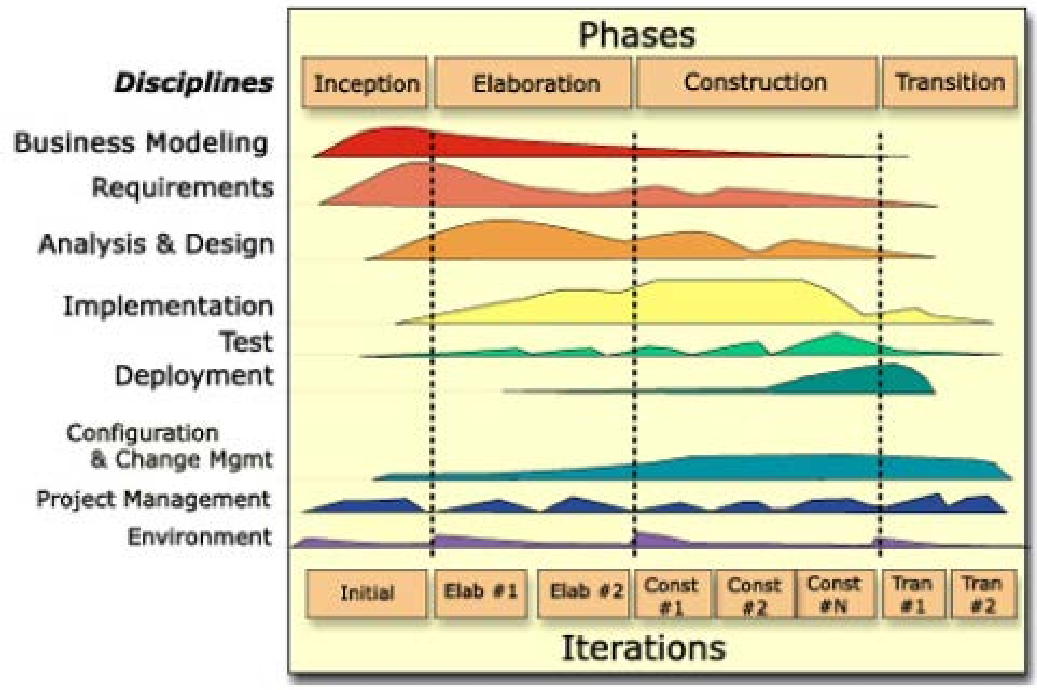


Figure 70. Two-Dimensional Rational Unified Process Framework Representation (Rational 2001).

Each phase concludes with a well-defined milestone – a point in time at which certain critical decisions must be made, and therefore indicating key goals have been achieved.

Each phase is further broken down into iterations, comprising a complete development loop resulting in a release (either internal or external) of an executable product, which is a subset of the final product under development. The product thus grows incrementally towards the final product. Compared to the traditional Waterfall or Vee process, the iterative approach is seen to offer numerous advantages, including:

- o earlier mitigation of risk
- o change is more manageable
- o higher level of reuse
- o project team's initial learning curve is reduced, and supports continuous learning
- o improved product quality.

2. The *vertical dimension* of Figure 70 represents the core process activities, which are grouped logically in terms of the nature of the SW engineering activities. This dimension represents the static aspects of the process elements, expressed in terms of activities, disciplines, artefacts and roles. Collectively across the phases, products are generated including the initial business case, vision document, project plans, risk assessments, expenditure tracking, schedule tracking, use case model, SW architecture description, executable SW architectural prototype, SW product baseline integrated on designated computing platforms, and user manual.

In contrast to the Vee or the Waterfall instantiations of the SE process described earlier in Section 5.5, the RUP supports the notion of spiral development. The designated process steps seek to support flexible, iterative SW development and integration in the presence of uncertain or volatile requirements, while explicitly managing (and hence controlling) the notion of changing requirements, changing SW configuration, and associated SW verification testing over the development cycle.

The ability to develop and maintain visual models of the system and its components under development is fundamental to the RUP. When first released, the RUP was implemented in a tool suite to automate significant portions of the process; a UML model of the RUP process was implemented in the Rational Software tool suite to support the process design and authoring activities.

The RUP places considerable importance on the early development and verification of a component-based system architecture, and in the case of SW-intensive systems, also the SW architecture.

Notably, the RUP tools provide a series of UML-based templates for describing the system and SW architectures based on the concept of multiple architectural views. The design process supports specific activities to facilitate identifying architecturally significant elements as well as architectural constraints RUP also provides guidelines on how to make architectural choices, employing object-oriented concepts of modularity and encapsulation.

Here, the component is intended to be a non-trivial piece of SW (e.g. a SW module, package or subsystem) that fulfils a clear function, has a clear boundary, and can be integrated into a well-defined architecture or structure. Component-based development then occurs through the elucidation of a modular architecture, in which well-formed SW components are identified, given a separate identity, then individually designed, coded, and tested. The components, once individually tested, are progressively brought together, and integrated and tested to form the whole system.

The SW architecture facilitates the articulation of the structure of the SW in terms of its component modules, as well as the mechanisms and patterns by which they interact. Concepts such as packages, subsystems and layers are used during analysis and design to organise components and specify their interfaces (Kruchten 2001).

The components, once developed and verified using this method, can also lend themselves suitable for reuse in other SW applications, ostensibly to reduce development and verification effort, and thus improve SW productivity and SW quality.

The prevalence of standardised COTS SW components readily available within SW frameworks such as CORBA (Common Object Request Broker Architecture), ActiveX, and JavaBeans, has transformed SW development. SW development has now shifted from programming SW one line of code at a time (i.e. requiring a top-down design approach) to composing SW by assembling COTS and MOTS SW components with open-source standardised interfaces (i.e. open architecture), then successively integrating their interfaces to form the final solution (i.e. bottom up approach).

11.6.3 The RUP SE Process

The Rational Unified Process for Systems Engineering (RUP SE) was released by IBM in 2003 as an extension of the RUP to incorporate notions of systems engineering. RUP SE sought to improve the quality outcomes of the larger scale systems in particular, as well as reducing time to implement, cost, and risk. The principal RUP SE process steps across a system life cycle are shown in Figure 71 (Cantor 2003a).

The RUP SE incorporates an architecture model framework that enables the consideration of different sets of perspectives (e.g. logical, physical, information), and addresses the concerns of multiple stakeholders to deliver the system solution. Additional detail is provided on the framework in Appendix E.

The Rational tool suite was also updated accordingly to accommodate the additional SE functionality; the tool suite providing a graphical user interface overlaid on information stored in a database to support graphical diagram creation, editing, storage and retrieval, as typically shown in Figure 72.

The RUP SE supports classical SE precepts such as the idea of recursive design and decomposition. It suggests that systems and components can be viewed from two useful perspectives:

- A "Blackbox" perspective, i.e. viewing the system as a whole, the services it provides, and the requirements that it meets;
- A "Whitebox" perspective, i.e. viewing the piece-parts or elements that make up the system.

Thus, the RUP SE provides process steps and a UML-based model framework to support teams of engineers as they determine the "Blackbox" view of the system, and synthesise an optimised "Whitebox" system design that meets all the stakeholder needs.

The RUP SE supports the project development of large-scale systems in particular, which notionally comprise SW, HW, workers and information components, interacting together to support business operations within an organisation. The project developments typically have the following characteristics:

- The projects are large enough to require multiple teams with concurrent development;
- The projects have concurrent HW and SW development;
- The projects have architecturally significant deployment issues;
- The projects include a redesign of the underlying IT infrastructure to support evolving business processes (Cantor 2003a).

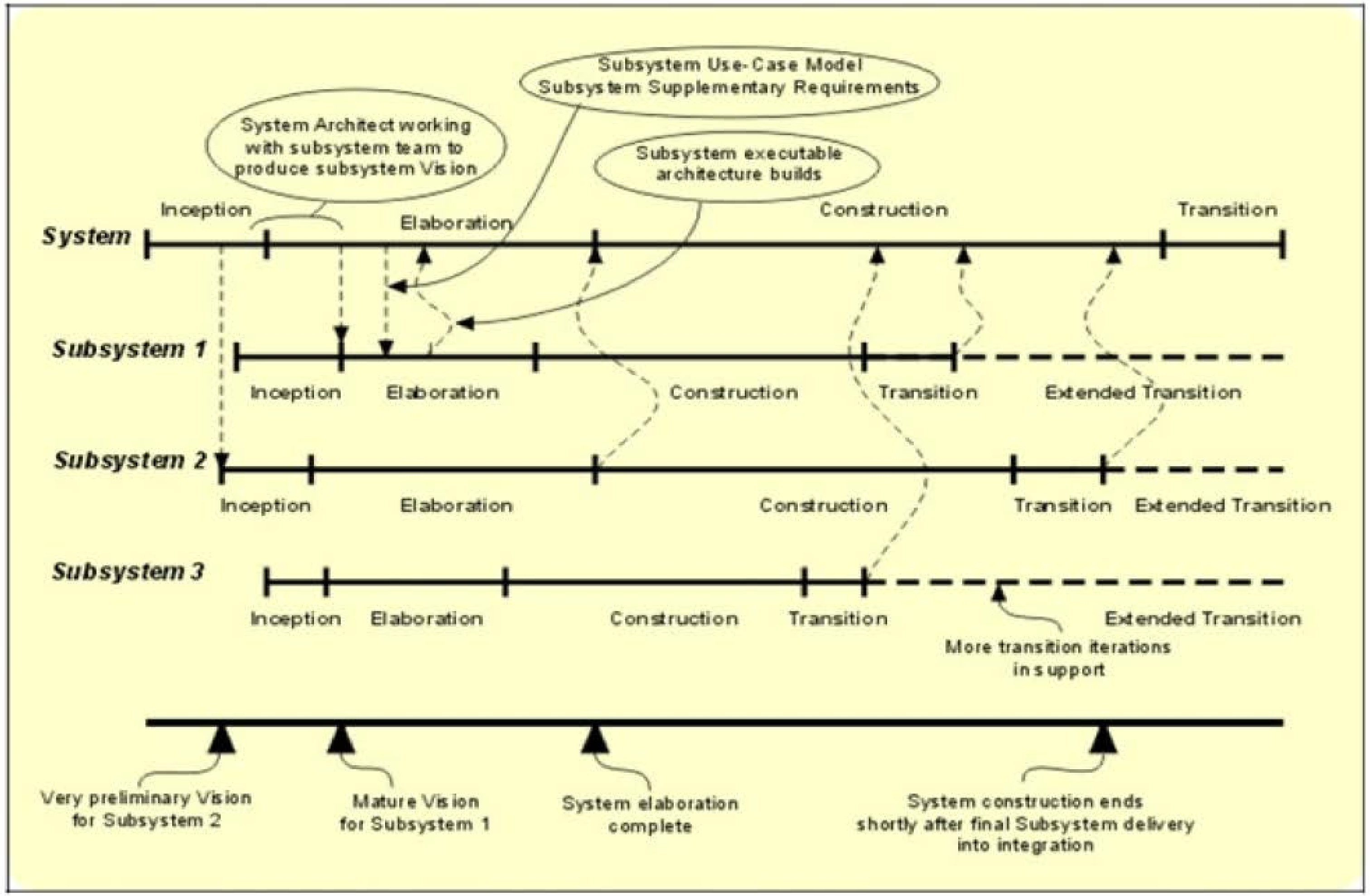

Subsystem Use-Case Model
Subsystem Supplementary Requirements
System Architect working with subsystem team to produce subsystem Vision
Subsystem executable architecture builds
Inception
Elaboration
Construction
Transition
System
Subsystem 1
Inception
Elaboration
Construction
Transition
Extended Transition
Subsystem 2
Inception
Elaboration
Construction
Transition
Extended Transition
Subsystem 3
Inception
Elaboration
Construction
Transition
Extended Transition
More transition iterations in support
Very preliminary Vision for Subsystem 2
Mature Vision for Subsystem 1
System elaboration complete
System construction ends shortly after final Subsystem delivery into integration

Figure 71. RUP SE Life Cycle Representation..

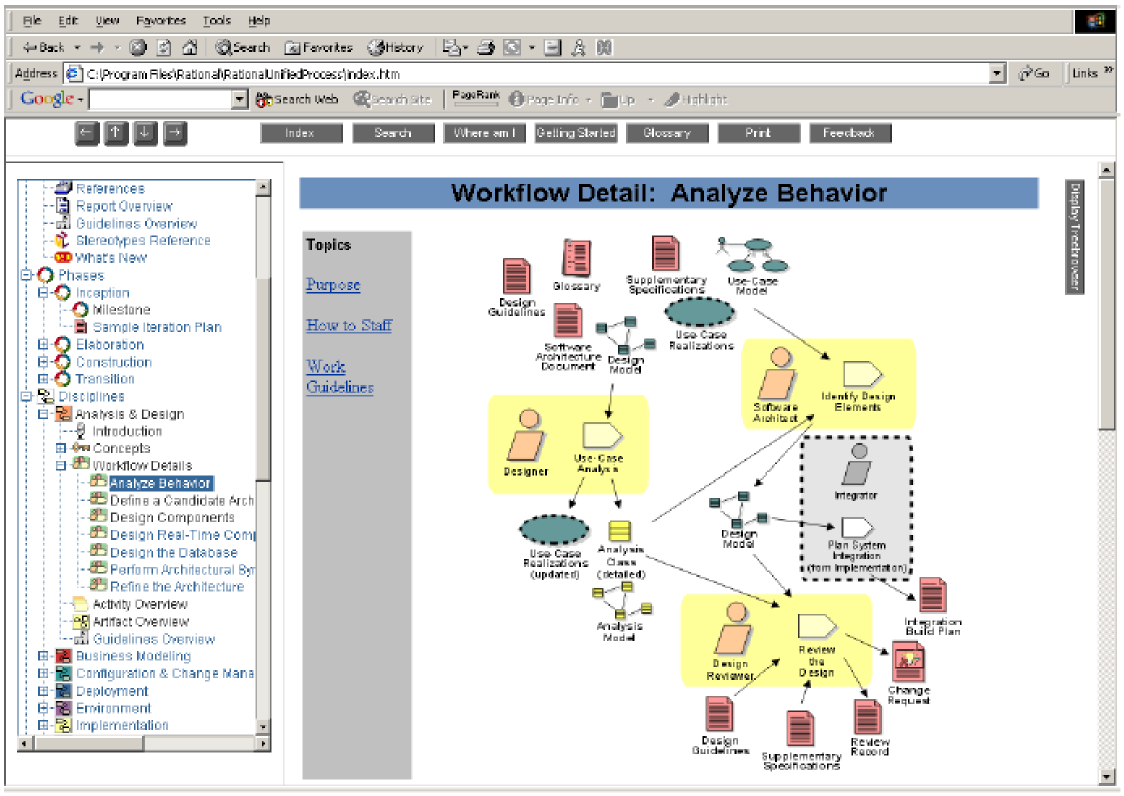


Figure 72. Example RUP Tool Graphical User Interface supporting UML Diagram Creation.

Significantly, as the RUP SE is an application of the RUP SW engineering process framework, the RUP SE retains its SW heritage; the HW component of the system equating to the IT infrastructure underpinning the SW application, which in turn supports the business processes of the organisation.

In the context of RUP SE, (Cantor 2003a) also notes that the term system can have a variety of different meanings, and offers the following clarification:

"A system is a set of resources that provide services that are used by an enterprise to carry out a business purpose or mission.
System components typically consist of hardware, software, data and workers.
Systems are specified by the services they provide, along with other non-behavioural requirements such as reliability or cost of ownership.
Designing a system consists of specifying components, their attributes and their relationships.
A system is a set or assemblage of elements that exhibit behaviour collectively" (Cantor 2003a).

The notion of system as described within the RUP SE is consistent with the notion of system as described in Section 5.2, however, it is much narrower in scope. The utility of the RUP SE is essentially constrained to the context of SW-intensive systems that can be represented as a large state machine, underpinned by simplified, generic technology infrastructure, governed by the insights of computing science and SW engineering.

In addition, the RUP SE lacks the formalisms to support design of a system as a physical entity, which is governed by the laws of physics and classical engineering disciplines such as electrical, electronic, and mechanical engineering.

11.6.4 Object-Oriented Systems Engineering Methodology (OOSEM)

The INCOSE MBSE Survey provides an overview of another systems modelling methodology that utilises SysML as the modelling language, known as the Object-Oriented Systems Engineering Methodology (OOSEM), as illustrated in Figure 73.

OOSEM is a hybrid approach that leverages object-oriented concepts overlaid on a SE foundation (Lykins et al. 2000). OOSEM was initially conceived to support the design and development of large, distributed information systems. Maturation of the approach has been largely driven by INCOSE members with a specific interest in the methodology under the auspices of the INCOSE OOSEM Working Group, established in 2000.

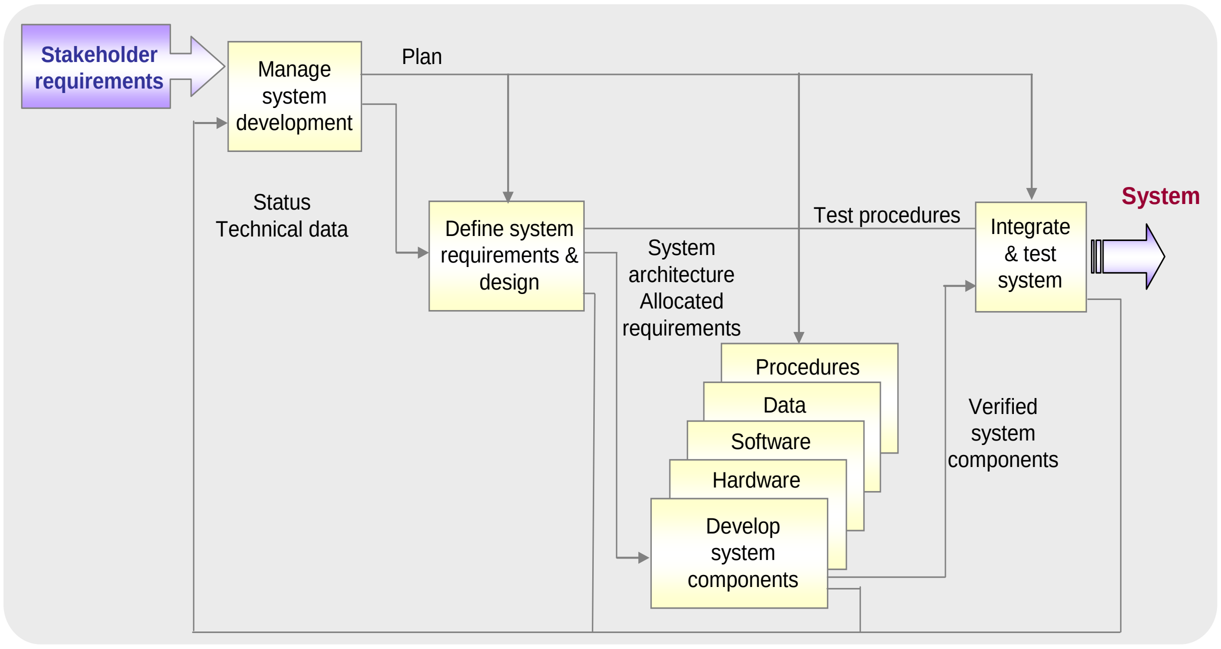


Figure 73. OOSEM Methodology Overview (Estefan 2008).

The OOSEM methodology incorporates process steps as shown in Figure 74. These closely resemble well-known SE process activities associated with system design and development illustrated previously in Figure 30, including:

- Analyse Stakeholder Needs – Concept Phase
- Define System Requirements - Concept Phase
- Define Logical Architecture – Development Phase
- Synthesise candidate solution architectures – Development Phase

- Evaluate alternative system solutions and optimise – Development Phase
- Verify and validate the system solution – Development Phase

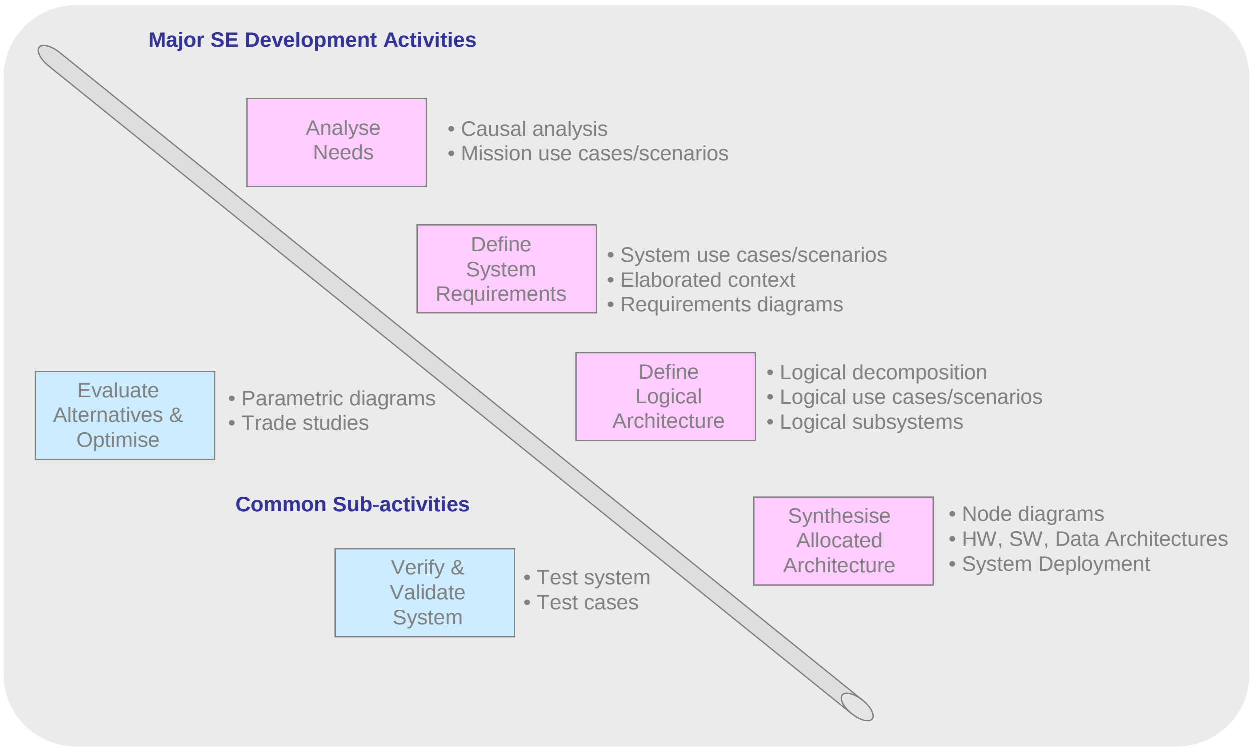

Major SE Development Activities
Analyse Needs
• Causal analysis
• Mission use cases/scenarios
Define System Requirements
• System use cases/scenarios
• Elaborated context
• Requirements diagrams
Evaluate Alternatives & Optimise
• Parametric diagrams
• Trade studies
Define Logical Architecture
• Logical decomposition
• Logical use cases/scenarios
• Logical subsystems
Common Sub-activities
Synthesise Allocated Architecture
• Node diagrams
• HW, SW, Data Architectures
• System Deployment
Verify & Validate System
• Test system
• Test cases

Figure 74. OOSEM Activities and Modelling Artefacts (Estefan 2008).

As in SE, these activities are recursively and iteratively applied at each level in the system hierarchy. Notably, the concept of production is treated markedly different in the SE context as compared to the SW context – principally due to the nature of the product. Production and preparation of a SW application for delivery can take only a few simple process steps – it can be simple as copying the required application files, installation instructions and user guide onto non-volatile media, then packaging suitable for postal or courier despatch.

Similarly, as with other design-driven systems modelling methodologies such as RUP, OOSEM is focussed on facilitating detailed system implementation, and still requires disciplined management processes to support each of the activities to be effective, including risk management, SE management, and configuration management.

The OOSEM representation is implementation-centric in that it assumes the SE process starts with the provision of stakeholder needs, and it assumes the solution will be largely implemented in SW. From the Defence context, the SE process does not start with the provision of Stakeholder Needs as represented in the diagram of Figure 73, but requires extensive effort using a variety of methods of enquiry to support requirements elicitation activity, trade studies and evaluation of alternatives as a prelude to RFT issue during the Needs and Requirements phases. The OOSEM activity is equivalent to the acquisition phase of the Capability life cycle model in Defence.

11.7 Industry Impetus for MBSE

"It takes a heck of a lot more than just brilliant engineering to create a smart product that is successful in the market-place. Research shows that a third of all produced devices do not meet performance or functionality requirements and that 24% of all projects are cancelled due to unrecoverable schedule delays. Many times, the reason for a catastrophic system failure is not related to the system's engineering design; rather it is due to failures of knowledge or communication."

(Shamieh 2011)

One of the significant challenges typically associated with large scale or particularly complex projects is that of communication and management of information. The larger or more complex the project, typically:

- the larger the learning curve,
- the more people involved,
- the greater the dispersement and disparity of teams,
- the greater the difficulty in implementation,
- the greater the cost of implementation,
- the greater the difficulty and cost of fault remediation, and hence
- the greater the associated commercial and technical risk.

Poor communication can cause numerous problems, including:

- Lack of clarity of system goals,
- Multiple interpretations of system requirements,
- Missing, overlooked, or contradictory requirements,
- Significant time wasted gathering, sorting and consolidating inconsistent information from multiple sources,
- Teams working with outdated or inconsistent versions of documents,
- Extensive rework and repeated (and hence wasted) effort,
- Personnel dissatisfaction and disharmony.

Process formalisation can assist to significantly mitigate this risk, however, the greater the governance effort, the less flexible and responsive the process, and the greater the impost on cost and schedule. Good communication and collaboration across the stakeholder community is therefore imperative to ensure engineering endeavour is consistently, cohesively, and cost-effectively applied across the entire project.

One of the keys to good collaboration is unencumbered access to design information with known integrity and ownership. When the information generation and usage is geographically distributed, good information management is critical to the success of the engineering endeavour.

As indicated in Table 7, a variety of tools supporting different functionality may be utilised on any given project; no single tool can offer "one size fits all" functionality with a universal database. Information is likely to be distributed across multiple databases. The project information architecture is therefore of critical import to the collaborative effort.

MBSE tool vendors such as IBM contend that using a unified systems development environment with a "virtual" information repository such as shown in Figure 75 can assist to streamline development and testing processes. Miscommunication is reduced by making it easier for distributed teams to integrate their work and share their knowledge, thereby increasing yields in terms of quality, time, profitability and customer satisfaction (Shamieh 2011).

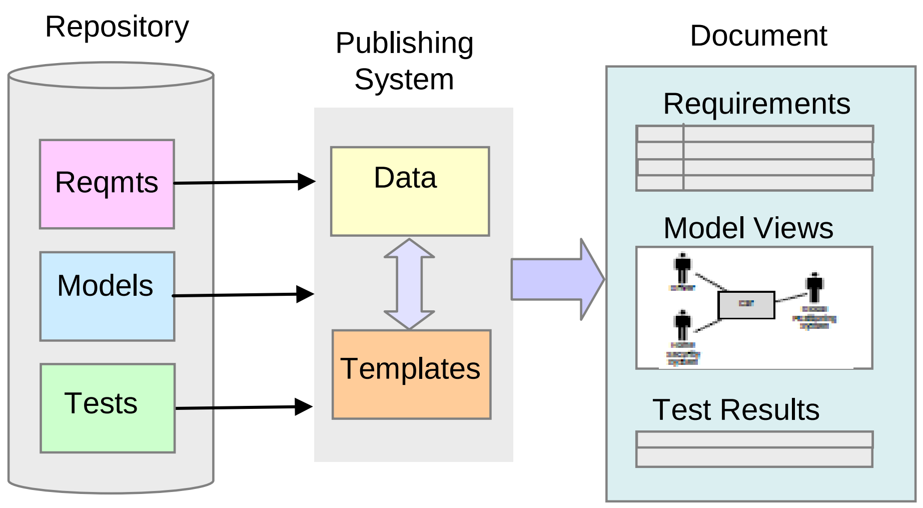


Figure 75. Project Information Architecture with Virtual Repository of Design Information to Improve Collaborative Effort (Shamieh 2011).

Automated documentation functionality to access the virtual information repository is a key feature of the MBSE-based unified development environment, to streamline the production of customised reports and ensure the integrity of the information and consistency of presentation in each report.

Significantly, not all information can remain in electronic form. Some hard copy documentation may still be required to satisfy contractual obligations, for compliance reporting, technical reviews, and for project management purposes.

11.8 INCOSE Impetus for a New MBSE Approach

The INCOSE MBSE Focus Group has actively sought to elevate the prominence of modelling in the SE process to take a central and governing role in the specification, design, integration, validation and operation of systems. This has been prompted by an appreciation of the increasing scale and complexity of new system and SoS developments, and increasing difficulty in managing consistency and maintaining currency in very large volumes of complex information, typical of military capability.

The new generation EA and MBSE tools support a variety of graphic modelling approaches specifically to implement information models to capture and manage prescribed information. Their stated objective is to provide improved decision support for capability definition, development and acquisition of both military systems and corporate ICT systems around the world.

These approaches variously utilise BPMN diagrams, IDEFx diagrams, N^2 diagrams, entity-relationship diagrams, UML diagrams, and most recently, SysML diagrams to present different views of data contained within a model or generated by a model.

Notably, new generation MBSE tools support a variety of object-oriented architecture description languages based on UML and SysML. These underpin specific modelling activity for acquisition decision-support, with the capability to generate the plethora of views defined in the US and UK military architecture frameworks DoDAF and MODAF respectively.

To improve the cost-benefit of the modelling activity and the quality of decision-support provided, a recent collaboration between OMG, US DoD and UK MoD has led to the merging of the information models or schemas[79]of the DoDAF and the MODAF into a common and simplified information model, with fewer artefacts.

Challenges have included:

- harmonising the modelling approaches,
- reducing information model complexity;
- promoting model and data reuse and model interoperability; and
- transitioning the stakeholder mindset from being view-centric to becoming more data-centric.

The rationalised schema common to both MODAF and DoDAF is an integrated data model, defined in terms of the architecture description language UPDM (Unified Profile for DoDAF and MODAF). UPDM comprises a selected subset of language constructs and diagrammatic representations from UML2 and OMG SysML. UPDM has similarly been adopted by NATO for use in conjunction with their NATO AF.

A similar collaboration has spurred the development of the tool data interchange standard AP-233 described earlier in Section 3, although this standard is still evolving (IOS 10303-233:2012).

Although each of these military architecture frameworks prescribes different graphical representations and sets of artefacts to meet local policy and governance requirements, and semantics can vary between different nations despite common use of terminology, the overall trend over the last ten years has seen significant progress in harmonisation of approaches across the international Defence community. This is despite significant differences in scale, budget, national policy, governance requirements and acquisition processes, in order to provide a more cost-effective approach with improved decision support for capability acquisition.

In Australia, Defence utilises a drawing-oriented approach to produce hard copy artefacts,

[79] Previously known as the CADM in versions up to DoDAF v1.5, recently re-released as DM2 in DoDAF v2.0.

with prescribed drawing templates and tool environment to produce the prescribed SE documentation and EA artefacts in line with policy and governance requirements in support of the capability development process. This is in contrast to the data-centric approach using an integrated data model and the UPDM modelling language as advocated in the US, UK and NATO.

Guidance in Australian Defence is provided in terms of the DCDH, available publically, together with a set of architecture principles published on the Defence internal network. As yet, guidance does not specifically articulate MBSE concepts or principles. Guidance is agnostic to a specific architecture modelling language or information meta-model (i.e. schema) ; the default being that provided by the mandated tool environment and respective tool configurations. Similarly, no specific guidance is provided on modelling methods. Australia is yet to accede to a more data-centric and vendor-neutral approach with prescribed architecture description language, schema, and modelling methods as used elsewhere amongst coalition partners.

11.9 SE Perspective on New Generation MBSE Tool Environment

Notwithstanding the increasing availability of new generation MBSE tools and methodologies, pertinent questions to ask include:

- "who is the user-base for new generation MBSE tools and methodologies in the Defence context"?
- "for what purpose(s) might this user-base be interested in using these new generation MBSE tools for"?

MBSE methodologies rely predominantly on diagrammatic techniques to convey the required information to support detailed system implementation, as opposed to current convention text-based documentation. By reducing the use of hard copy SE related documentation in favour of maintaining the information in managed databases with automated documentation generation capability, the INCOSE MBSE Working Group posits that MBSE methodologies can offer improved flexibility, consistency and traceability in documentation generation, and facilitate easier upgrade of the associated information set.

In this regard, ***the recent focus of interest expressed by the INCOSE MBSE Focus Group on information management, diagram generation, and automated documentation generation differs considerably from previous notions of systems modelling within the SE and SW engineering disciplines, which focussed on problem structuring and analysis, and solution synthesis.***

The system model in the MBSE context would appear to be a repository of information about the system, where the scope of information stored in the model is shaped by the schema or information model used to craft the model. This is more akin to an EA model representation rather than an engineering based system model representation. This is ostensibly more useful in providing acquisition decision support for governance and management purposes rather than for analysing and structuring a problem, and informing system analysis and design activity to drive towards a system solution (although the tools are capable of doing exactly this).

In other words, specifically pertaining to new generation MBSE practice, and most importantly, ***the modelling focus has shifted from incorporating information on the system in the model to compiling an information set about the system; defined by the prescribed architecture framework schemas, architecture descriptions, and SE documentation required to support specific policy and governance requirements.***

The recent emergence of SysML, with expanded language support for SE-based concepts such as requirements management and parametrics, and the subsequent codification of UPDM, coincide with the recent re-emergence of MBSE to the fore to support the new notions of information management. Notably, this pertains in particular to architectural information generated within the confines of the respective military EAFs, which are used to provide capability acquisition decision-support.

11.10 EAF Perspective on New Generation MBSE Tool Environment

As explained previously, most of the EAFs use UML and SysML like graphical diagrams as artefacts to document the framework outputs. However, EAFs are typically agnostic to both the underlying process to generate the artefacts, and the purpose for which the artefacts are to be used for.

EAFs typically do not acknowledge any particular relationship between:

- the originating modelling environment,
- a particular analytical process,
- the artefacts generated, and
- the context for which the artefacts are to be used for (i.e. the purpose of the modelling activity, e.g. if they are to be used to provide decision support within a SE process, an IT acquisition process, or an enterprise business process).

In the absence of an overarching engineering process, particular weaknesses of this artefact or view-centric approach from an engineering perspective include:

- the emphasis on the artefact rather than the analytical method (i.e. model construction) to generate the data contained within the artefact;
- a corresponding lack of understanding of the integrity of the data; and
- lack of visibility of intended use of the data.

All these are crucial considerations before undertaking any modelling activity.

Similarly, within the Defence community, operations researchers can also use the same modelling environment, EAFs, and/or UML/SysML-like artefacts to frame and pursue their respective lines of enquiry. However, similar to the situation using EAFs, it is agnostic to the SE context, and the intended use can differ substantially from the intent of similar artefacts generated within a SE or other engineering context. It can therefore be difficult to reuse artefacts from one modelling application to the next without knowing the heritage and integrity of the information set represented in each respective artefact.

Notably, tool vendors encourage usage of their tool environments to the widest market accessible. It is contrary to their commercial interest to constrain their market to one particular user segment. Over the last decade in particular, this has precipitated the emergence of a number of EAF tool vendors with backgrounds in either corporate business or commercial IT, who do not have a specific historic association supporting engineering activity in the Defence military or engineering communities. However, some of these vendors have been instrumental in forming the OMG and the subsequent codification of UML.

One such prominent commercial vendor is Microsoft™, who market their Microsoft Office™ tool suite, comprising a word processing tool, spreadsheet tool and a drawing tool. This tool suite is one of the preferred tools recommended by Defence to prepare AUSDAF artefacts in support of the Defence capability development process.

Another example is the commercial EAF, TOGAF. TOGAF is marketed to both the commercial sector and the Government sector, including Defence, along with a companion suite of business analysis tools. Since the tools can be used to generate military EAF compliant artefacts, but are agnostic to SE, these tools fall in the category of EA tools rather than SE tools.

Notwithstanding, the EA tools may also be suitable for use to support certain types of engineering activity, particularly for systems analysis and design, provided they are used by appropriately skilled tool-users, and in the appropriate context.

Notably, in its definition of MBSE provided in Section 2, INCOSE has apparently sought to differentiate between the broader set of potential users of the same modelling environment in either a business, IT, OR, or EA context, constraining the application of modelling techniques and tools to that specifically occurring within a deliberate SE setting. A more comprehensive survey of MBSE methodologies is provided in (Estefan 2008).

11.11 Clarifying MBSE Perspectives

It is apparent there are multiple MBSE perspectives, somewhat similar, but with different problem foci and different problem solving approaches:

- A general class of computer-based SE engineering methodologies and modelling tool environments supporting SE principles and processes, which offer improved process efficiency with reduced cost, schedule and/or technical risk supporting bespoke engineering development;
- A general class of computer-based SW engineering methodologies and graphical modelling tool environments, which offer improved SW engineering development efficiency supporting bespoke SW development; typically with auto-code generation capability, and typically hosted on an OTS commodity HW host platform.
- A unified engineering development environment with a "virtual" information repository (i.e. database), offering automated documentation or artefact generation (replacing hard copy documentation);
- A computer-based graphical modelling approach for developing views or artefacts as defined in military enterprise architecture frameworks such as MODAF and DoDAF;
- A computer-based modelling tool for developing views for commercial enterprise architecture frameworks to assist in ICT investment-related and other corporate business process analyses;
- A computer-based graphical modelling tool to assist in business process analysis;

- A computer-based graphical modelling tool used to undertake operations research and general systems analysis;
- A specific SW engineering development methodology known as Model Driven System Design, promoted by the MBSE Working Group of INCOSE;
- Specific system modelling methodologies utilising the SysML™ graphical modelling language.

Notably, in the majority of these computer-based MBSE approaches, the system is assumed to be software-intensive. It is therefore crucial to understand the context of the usage of the term MBSE when referring to MBSE concepts and utilising different MBSE resources.

12. Mindsets and Perspectives

From this report, it is evident that the various aspects of the capability development process can be applicable to a very large number of Defence stakeholders, potentially spanning numerous organisations, either providing inputs to the process, partaking in the process, managing part of the process, or interested in the output, outcomes, and consequences of the process in action.

Two decades ago Brian Mar portended that if the basic concepts of SE were not adequately understood by the workforce, then the practice of SE would struggle to cope with the envisaged challenges in the 21st Century, including rapidly changing technologies, modification rather than replacement of existing systems, and fragmentation of engineering disciplines (Mar 1992).

The INCOSE Handbook goes further to suggest that SE is a profession as much as it is a process, where exponents engage expansively in systems thinking.

These challenges foretold by Mar have come to bear; but underlying challenges in the Defence context include resolving:

- who in Defence comprises the SE workforce;
- what level of SE expertise is required to take on the challenge of developing the Networked Force 2030;
- what level of SE expertise does Defence has;
- where does this SE expertise reside; and
- where should the SE expertise reside?

As previously discussed, in Defence, multiple skills and perspectives are brought to bear in support of the capability development process as illustrated in Figure 76.

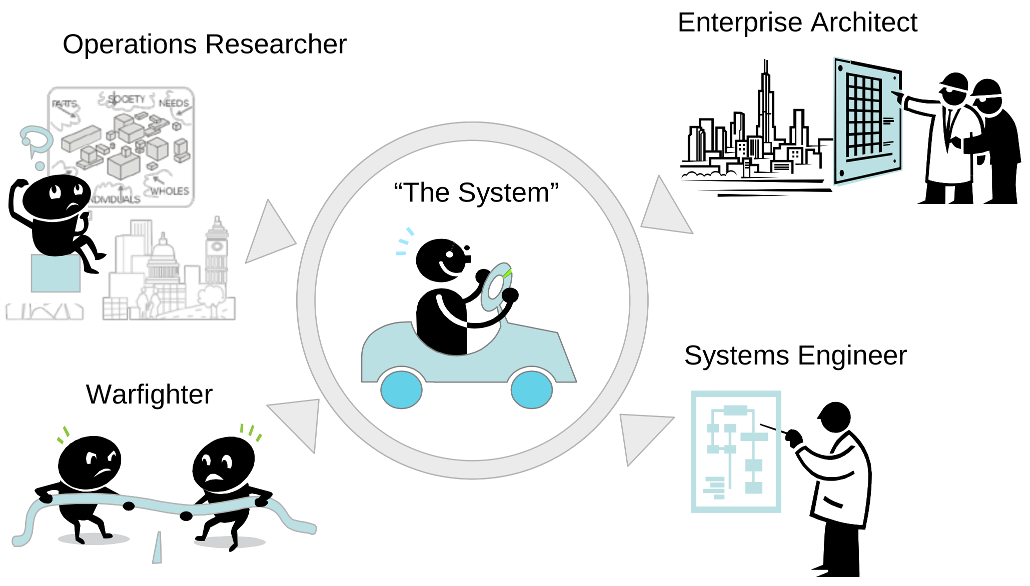


Figure 76. Different Mindsets and Perspectives in Defence.

During the Needs phase, extensive input is provided by the warfighter, complemented by significant contributions from the Defence operations research (OR) community. Systems analysis is therefore typically undertaken from an OR perspective rather than a SE perspective. Although there is significant overlap between the professions, different questions are normally posed by operations researchers compared with systems engineers.

OR encompasses a wide range of problem solving techniques and methods, drawing extensively from the field of mathematical sciences. OR is typically used to improve decision-making quality and efficiency. Operations researchers can similarly utilise systems thinking in their approach to complex problem solving. Scientific modelling is used extensively, particularly with regard to analysing human-technology interactions and influences in complex socio-technical systems, including measurement of factors such as chance and risk with which to predict and compare the outcomes of alternative decisions, strategies and controls (Taha 2002).

One notable difference between systems analysis in the OR context compared with the SE context is one of system definition. The notion of "system" in OR can be quite abstract or conceptual; often deliberately agnostic to any physical instantiation. This is in contrast to the requirement in SE to be explicit in terms of identifying the specific system of interest, system boundary locations, system element composition, and required functionality in a real-world context, in order to achieve specific engineering outcomes.

Similar input is provided by the warfighter and the OR community during the Requirements phase. SE expertise for preparation of capability development documentation (CDD) during the Requirements phase is primarily sourced from personnel contracted from industry to assist in CDD writing. Input may also be provided from DMO participation in Integrated Project Teams (IPTs), which may be stood up for a specific MCE project phase. As such, the warfighter has primary SE responsibility for system definition and CDD articulation during the formative stages of the system life cycle.

The primary focus of the Defence acquisition organisation is on project managing cost and schedule after contract award to achieve the required contract outcomes. To achieve the desired engineering outcome, it remains the responsibility of the warfighter to manage the specific materiel system requirements during the acquisition activity, and it is the responsibility of the capability manager to manage the requirements of the capability system over the life of the capability.

This means SE responsibility during the Needs and Requirements phases of the capability development process is vested with the warfighter. During the acquisition phase, DMO has primary responsibility for contract management, with the Supplier responsible for system implementation and verification. After acceptance into operational service, SE responsibility associated with provision of on-going support is vested with DMO, with management of the capability system normally vested with the capability manager responsible for operating the capability.

Thus, it is evident that the majority of contribution to the SE process in Defence, particularly during the early stages in the life cycle, is undertaken by Defence personnel outside the SE profession, and without the detailed knowledge base embodied in the SE Body of

Knowledge (SEBoK 2012)[80]. This places particularly strong reliance on the process articulated in the DCDH, and the DMO acquisition and support processes for successfully managing the aggregate capability outcomes sought by Defence.

[80] The SEBOK has been developed by INCOSE to encapsulate the internationally accepted body of knowledge relating to SE (SEBOK 2012). The SEBOK is available in Wikipedia form at [online] URL: http://www.sebokwiki.org/.

13. MBSE Utility to Defence

13.1 Utility Considerations

SE and MBSE utility to Defence is examined from a variety of perspectives:

- Nature of the problem,
- Nature of the outcomes required,
- Relevance or suitability of the methodologies,
- Ease or difficulty of application of the methodologies,
- Suitability of the MBSE development environment,
- Implications for stakeholders, and
- Implications for governance.

13.2 Defence Problem Space Considerations

Many problems faced by the military cannot be solved by just employing technology-based solutions. After analysing and determining the nature of the problem, many can be solved by other means. These can include application of :

- process change;
- organisational change;
- change in personnel capability;
- change in service delivery; and
- improved training and/or resource allocation.

The fundamental utility of the SE process lies in its ability to provide a formal method to systematise systems analysis and systems synthesis in a repeatable manner, with defined decision and quality criteria, to solve a problem that, by its nature, can be solved by arriving at a technology-based solution. This solution must satisfy the original intention and be fit for the purpose intended, with its development managed in a deliberate manner to achieve explicit technical outcomes within budget and schedule expectations.

Spawned from the US Defence and Aerospace industry under the auspices of MIL-STD-499, the notion of the SE process has specifically evolved to drive technology-centric solutions to military problems of the ilk faced by Defence, as previously shown in Figure 33 drawn from international standard IEEE-1220.

Although instantiations of the SE process are many and varied, the fundamental concepts of system identity, system hierarchy, system life cycle, and the need to establish a requirements baseline, with recursive analysis and synthesis activity followed by verification and validation and support activities remains unchanged. The basic tenets of SE, to ensure “the right solution is built”, and “the solution is built right”, are thus both robust and enduring, and remain relevant to the engineering of Defence capability systems of the 21st Century.

Military capability can manifest in many forms, but it is self-evident that technology-based military systems must undergo an engineering process of some form in order to achieve the intended technical outcomes. This engineering process must contain sufficient formalisms and rigour to address the entire scope of the problem at hand, whilst bounding the scope of the problem to the minimum necessary to ensure a feasible, timely, quality and cost-effective technical solution can be achieved. This is to occur in the circumstances where the solution is to be effected as an actual physical technical system which can be used by people to achieve intended outcomes, rather than by other means.

By definition, an engineered product or technology-based system is at the heart of a Defence capability system, being one of the FIC elements; military capability being defined in socio-technical terms as the integrated composite of people, products and process. The remaining socio-technical FIC elements help shape both the requirements space and the constraint space associated with the capability system.

The scope of technical considerations then spans properties and performance of all HW, SW, facilities, materials, data, services and techniques combined, and requires application of commensurate management considerations as shown in Figure 35 drawn from international standard ISO/IEC 15288 to achieve the intended outcomes.

Most importantly, it must be possible to explicitly establish the system boundary of the engineered capability system, along with the specific configuration items that are components attributed as belonging to the system, and occurring within the system boundary, so that the nature of the interactions can be appropriately accounted for, and the interdependencies suitably managed.

It is therefore of critical significance that the Defence capability development activity continues to remain agnostic to notions of specific system identity and system boundaries in the SE context. Use of Project boundaries in lieu of SoS or system boundaries, belie the ability to manage key system interactions and key system interfaces over time, across multiple projects and project phases.

Also importantly, but not always apparent, those remaining elements of FIC associated with a particular Defence Capability System are not shaped by engineering processes, despite the fact that they are interdependent. Organisation, personnel, training, and command and management result from the application of other military planning and organisational management processes within the enterprise and are therefore external to the Defence instantiation of the SE process.

13.3 MBSE Process Considerations

As suggested above, technical systems can manifest in many different forms with many different properties, spanning for example:

- mechanical and electro-mechanical;
- electrical and electronics;
- optical and electro-optical;

- microwave;
- acoustic;
- magnetic;
- firmware;
- middleware; and
- software.

Each engineering discipline requires application of different specialist techniques to contribute to the overall progression of the SE activity. However, all engineering activity undertaken within the respective specialist disciplines still falls under the general umbrella of the SE process. It is therefore crucial to understand the nature of the problem first, and the type of analysis and synthesis required. It is also essential to factor the likely nature of the technology solution, the SE development environment needed, and associated risk, when first planning and resourcing the engineering activity. This also includes being able to assemble an appropriately staffed and skilled team to undertake the activity.

It is evident that Defence and industry have markedly differing perspectives: one seeking to explore the problem space from an acquirer's perspective, and hence elicit requirements whilst refraining from forming a specific solution; the other focussing on exploring the solution space from a supplier's perspective, with specific intent to elicit potential solutions. The methods of enquiry for Defence and industry will therefore differ markedly between the two mindsets and the differing responsibilities.

This difference in stance is a crucial differentiator between Defence and industry, having profound impact on many different considerations including:

- the utility and relevance of methods
- the utility and relevance of tools;
- what analytical techniques are employed;
- what skillsets are brought to bear;
- what information is required;
- what outcomes are sought;
- what governance is applied;
- what information is generated;
- who owns the information;
- how the information is managed;
- in what form is the information required; and,
- what decisions are sought.

A critical dichotomy between the Defence approach to capability acquisition and classical SE is with regard to the mindset applied towards shaping the solution space. An essential part of the SE process is the effort directed towards trade-off analysis in shaping the final solution; considering the range of possibilities, and optimising decisions to obtain the best outcome after consideration of cost, risk, schedule, performance and quality factors and/or constraints.

Defence has a mandate to remain largely conceptual in the RFT documentation for capability acquisition, primarily constrained to describing solution classes rather than a specific system solution. The reliance on tenderers to offer different system design solutions to meet the abstract concepts described in the RFT, limits Defence to choosing a specific system solution offered within the constraints of the RFT scope, regardless of whether any of the solutions offered by the tenderers provides the best possible outcome for Defence as a whole.

The inability of a Defence Project to reconsider the ramifications of a proposed techncial solution and refine the proposed system design based on the additional information brought to hand is a fundamental break point in the SE process, particularly as it may contain unconsidered ramifications on the remaining elements of FIC. As such, any system development methodologies that rely on articulating an initial concept for system implementation, then refining the concept with more and more detail as the design progresses through successive decomposition, are outside the remit of the Defence contribution to capability development.

The responsibility for successful implementation of the Defence materiel system within the auspices of a Defence MCE Project falls on the winning tenderer, despite interdependencies with the other elements of FIC, and despite the tenderer's inability to reshape the remain elements of FIC. A feedback mechanism is implied, although it is actually decoupled, between the successful tenderer's system design activity and other Defence management activity to accommodate unanticipated ramifications of the system design for the remaining FIC elements, and to factor in other Defence Project and Defence capability interdependencies.

The successful tenderer in industry is also constrained to determining a system solution within the approved boundaries set by the DCP Project and approved government funding. This mindset also exacerbates the potential of a tendered solution being incompatible with existing legacy systems. Ensuring the right solution is built is thus out of scope of the actual system design activity. Capability shortfalls of note between the contracted implementation and the actual requirement, which are outside the auspices of the authorised Defence Project activity, are typically relegated for separate consideration by Government at a later date.

Ultimately, the focus of MBSE methodologies such as RUP SE and OOSEM is on arriving at a good quality SW system solution to the problem at hand, through a process of supporting feedback, design refinement and decomposition. They are thus are solution-centric, and fall within the purview of technically-oriented SW engineering specialists.

Defence attention, on the other hand, as the acquirer, is directed towards understanding and articulating the nature of problem to be solved, typically expressed in terms of customer needs and requirements. Defence's perspective is therefore problem-centric, which falls within the purview of (non-technical) strategic analysts and operations researchers. Any solution classes suggested by Defence are typically generic in nature so as not to prescribe a specific implementation.

The problem-centric perspective of Defence, together with the lack of impetus in Defence to construct and manage models of actual system implementations; a paucity of SE know-how; and the inability to provide feedback, design refinement and influence decomposition, renders notions of a recursive MBSE process ineffectual under these conditions of tendering. Closer collaboration between Defence as the acquirer, and industry as the supplier, is a necessary precursor to accommodate notions of recursive feedback and design refinement across the acquirer/supplier boundary to realise better acquisition outcomes.

The use of Integrated Product Teams (IPT) comprising both the acquirers and suppliers is one such collaborative approach that has proven successful in the US for defence capability acquisition. This has been codified by DoD through the publication of the Integrated Product and Process Development (IPPD) Handbook (DoD IPPD Handbook 1998). One of the key tenets of IPPD is multidisciplinary teamwork through IPTs. IPPD uses three core principles of integrated teaming, shared vision, and concurrent engineering, where:

- integrated processes emphasise parallel rather than serial development;
- these processes for developing the product and for developing product-related life cycle processes, such as the manufacturing process and support process, are integrated and conducted concurrently;
- these processes accommodate the information provided by stakeholders representing all phases of the product lifecycle from both business and technical perspectives; and
- processes for effective teamwork are effected (Dickinson et al. 2008).

Closer collaboration between Defence and industry also provides a valuable learning opportunity for all parties concerned to expand their knowledge bases and better understand opposing perspectives.

13.4 MBSE Tool Considerations – Analytic Capability

Notwithstanding Defence capability acquisition process guidance, commercial MBSE tools support a range of analytical techniques that can be usefully applied conceptually, without requiring actual physical instantiations in the real world. Similar to trade-off analyses, the ability to create system models of various notional systems, as well as real-world implementations, allows an analyst to explore the manifestation of a client's problem space, the boundaries of the feasible solution space; and to articulate the constraint space. The modelling environment provides the means to describe problems in detail in a form amenable to support systems analysis, with the ability to compare function and performance, and potential fitness for purpose of various notional system configurations to resolve the original problem posed.

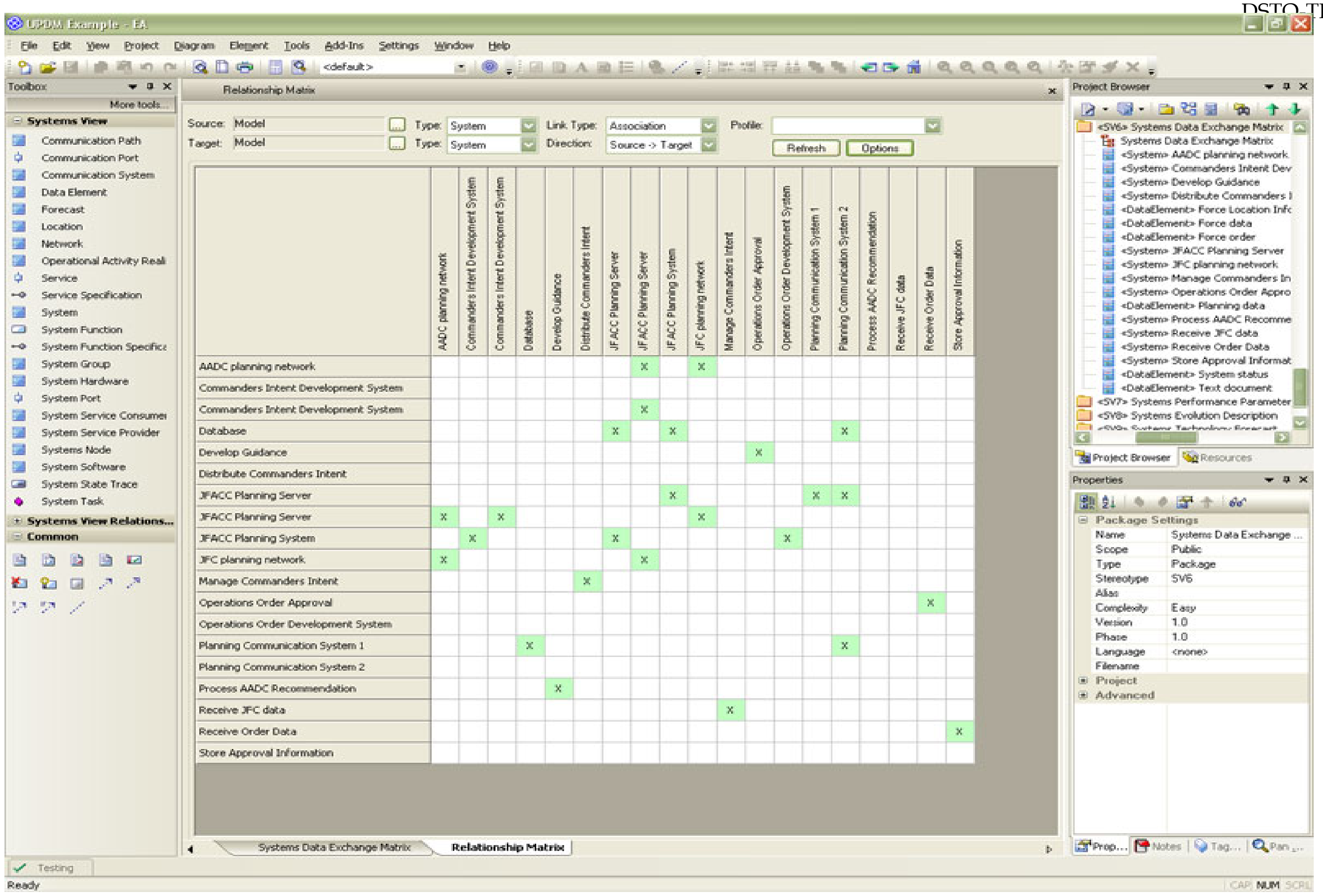
The MBSE tool capabilities allow important analytical insights to be distilled including:

- who are the users of the system;
- what are the users of the system doing;
- what are the objects in the real world;
- what are the associations between the objects in the real world;
- what use cases are relevant;
- what use cases are sufficient;

- what objects are relevant for each use case;
- how do the objects behave for each use case;
- how can objects within a use case collaborate with each other;
- how well does the system perform;
- how can real-time control be implemented;
- how can the system be built (i.e. trade options); and
- how is the system built.

These objects can represent a huge variety of "things" that are relevant to the manifestation of the system, both concrete and abstract. These can include individual FIC elements, resources, information, process steps, functions, metrics, and states of existence – depending on the vocabulary of the SDL supported by the tool, and the selective tailoring of the SDL in the form of a published and managed database schema or meta-model, scoping the information set of interest.

Similarly, MBSE tools can typically report the results of the analyses in a range of suitable engineering style artefacts, in both textual and graphical form, including textual notation, tabular notion, N^2 diagrams, IDEFx diagrams, and Entity-Relationship diagrams. A UPDM based Relationship Matrix from the Sparxsystem EA tool *Enterprise Architect* showing cross-referencing capabilities is illustrated in Figure 77.



Source: Model Type: System Link Type: Association Profile:
Target: Model Type: System Direction: Source -> Target

	AADC planning network	Commanders Intent Development System	Commanders Intent Development System	Database	Develop Guidance	Distribute Commanders Intent	JFACC Planning Server	JFACC Planning Server	JFACC Planning System	JFC planning network	Manage Commanders Intent	Operations Order Approval	Operations Order Development System	Planning Communication System 1	Planning Communication System 2	Process AADC Recommendation	Receive JFC data	Receive Order Data	Store Approval Information
AADC planning network								X		X									
Commanders Intent Development System																			
Commanders Intent Development System								X											
Database							X		X						X				
Develop Guidance												X							
Distribute Commanders Intent																			
JFACC Planning Server									X					X	X				
JFACC Planning Server	X		X							X									
JFACC Planning System		X					X						X						
JFC planning network	X							X											
Manage Commanders Intent						X													
Operations Order Approval																		X	
Operations Order Development System																			
Planning Communication System 1				X											X				
Planning Communication System 2																			
Process AADC Recommendation					X														
Receive JFC data											X								
Receive Order Data																			X
Store Approval Information																			

Name	Systems Data Exchange ...
Scope	Public
Type	Package
Stereotype	SV6
Alias	
Complexity	Easy
Version	1.0
Phase	1.0
Language	<none>
Filename	

Figure 77. *UPDM-based Relationship Matrix showing Traceability Links of Interest*

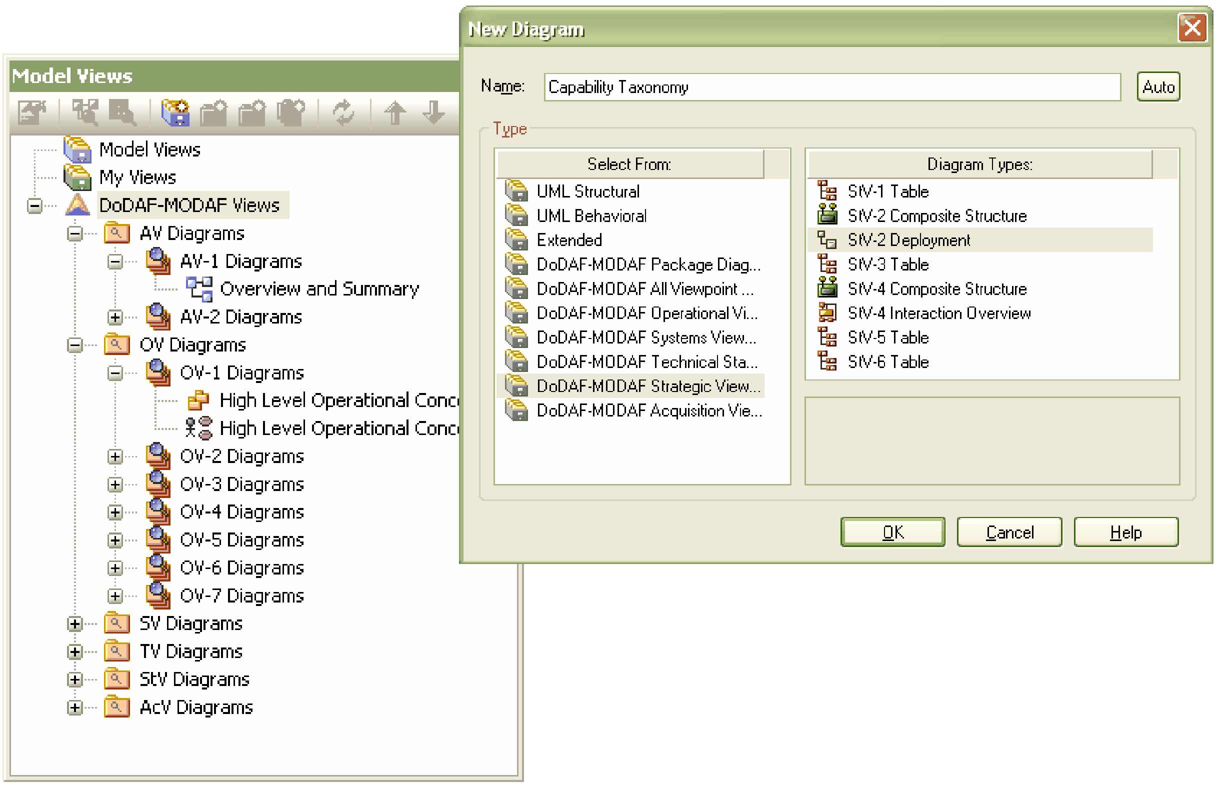
[online] URL: *http://www.sparxsystems.com/products/mdg/tech/dodaf-modaf/index.html*.

As the database is successively populated with information (architectural elements) as defined by the schema, an information repository is built up of the entities and their relationships, spanning both source data and analytic output data, within the defined scope of problem under consideration, and within the ascribed solution space (can be actual or conceptual; feasible or non-feasible).

The ability of the MBSE tools to articulate the various relationships between the information describing both the problem space and the solution space in a structured and repeatable way is therefore of particular utility to systems analysis.

Depending on the knowledge representation techniques supported by the MBSE tools, the information in the repository can be drawn from to support further analysis or to populate numerous templates for display as artefacts, for example, as framed by the various enterprise architecture frameworks.

An example tool user interface to access the different diagram types and presentation formats in MODAF and DoDAF is shown in Figure 78.


Model Views

- Model Views
- My Views
- DoDAF-MODAF Views
  - AV Diagrams
    - AV-1 Diagrams
      - Overview and Summary
    - AV-2 Diagrams
  - OV Diagrams
    - OV-1 Diagrams
      - High Level Operational Conc
      - High Level Operational Conc
    - OV-2 Diagrams
    - OV-3 Diagrams
    - OV-4 Diagrams
    - OV-5 Diagrams
    - OV-6 Diagrams
    - OV-7 Diagrams
  - SV Diagrams
  - TV Diagrams
  - StV Diagrams
  - AcV Diagrams

New Diagram

Name: Capability Taxonomy

Select From:	Diagram Types:
UML Structural	StV-1 Table
UML Behavioral	StV-2 Composite Structure
Extended	StV-2 Deployment
DoDAF-MODAF Package Diag...	StV-3 Table
DoDAF-MODAF All Viewpoint ...	StV-4 Composite Structure
DoDAF-MODAF Operational Vi...	StV-4 Interaction Overview
DoDAF-MODAF Systems View...	StV-5 Table
DoDAF-MODAF Technical Sta...	StV-6 Table
DoDAF-MODAF Strategic View...	
DoDAF-MODAF Acquisition Vie...	

Figure 78. MBSE Tool Sparx System Enterprise Architect Tool User Interface to Access DoDAF-MODAF Diagram types ([online] URL: http://www.sparxsystems.com/downloads/pdf/DoDAF-MODAF.pdf).

An example of the DoDAF 1.0 CADM integrated data model is shown in Figure 79, depicted using the architectural style of IEEE-1471. This database schema prescribes both the entities and the relationships of interest on which information is sought, for example, pertaining to specific US DoD Program acquisition activity. The aggregate set of data (i.e. architectural elements) is stored in an information repository (i.e. a database) for subsequent retrieval and display in the form of DoDAF architecture products or artefacts (OASD 2006).

The data set, in the example of a US DoD Program acquisition activity will typically include:

- a set of users of the system;
- a set of objects within the operating environment;
- a set of associations between the objects (including behaviours);
- a set of use cases and their descriptions;
- a set of components comprising the system elements; and
- a set of performance metrics.

Most importantly, in the case of the US DoD, information garnered for storage in the MBSE tool information repository is derived from concurrent Program SE-based acquisition activity, and is used to provide process decision support to shape the US DoD capability acquisition outcomes[81].

The example of Figure 80 provides an indication of the types of knowledge representation techniques and display formats that can be supported within an MBSE tool to generate different DoDAF views of the data in the database. Descriptions of corresponding key CADM entities in the CADM Meta-model are provided in Figure 81. The corresponding mapping of these key CADM entities to the respective DoDAF diagrams is shown in Figure 82 (OASD 2006).

A list of UML diagrams supporting knowledge representation formats conforming to DoDAF architecture product requirements is provided in Appendix F of this report.

[81] U.S. Defense capability acquisition is carried out under the auspices of US DoD Directive 5000.01 and DoD Instruction 5000.02, (DoDD 5000.01 2007), (DoDI 5000.02 2008). Briefs on the underlying US DoD SE-oriented method known as Joint Capabilities Integration and Development System (JCIDS) is provided in (Ryder & Flanigan 2005) and (Dickerson & Mavris (2010).(Dickerson & Mavris 2010) also provided a brief on the UK MOD Acquisition Operating Framework, which uses similar capability-based acquisition principles to the US JCIDS.

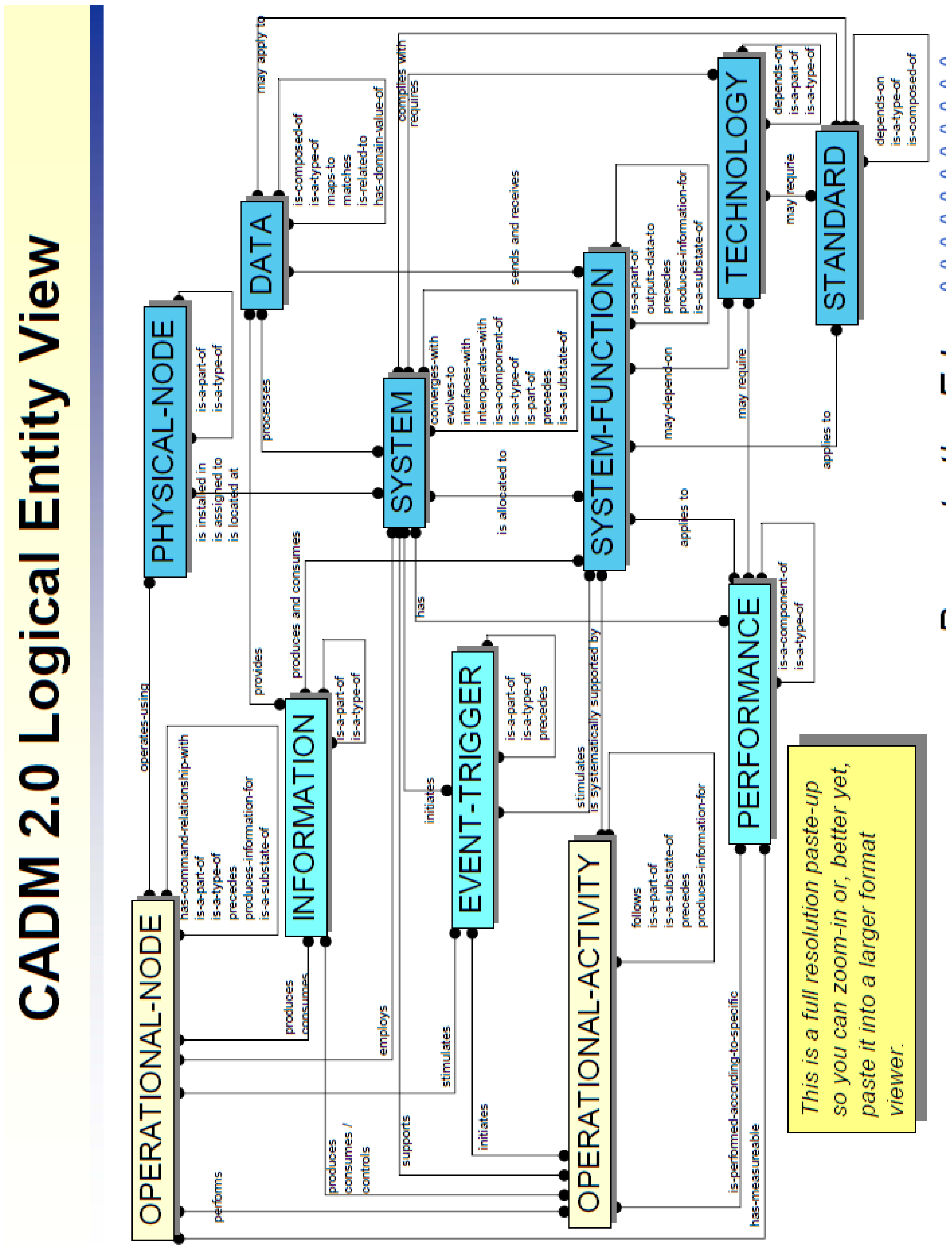
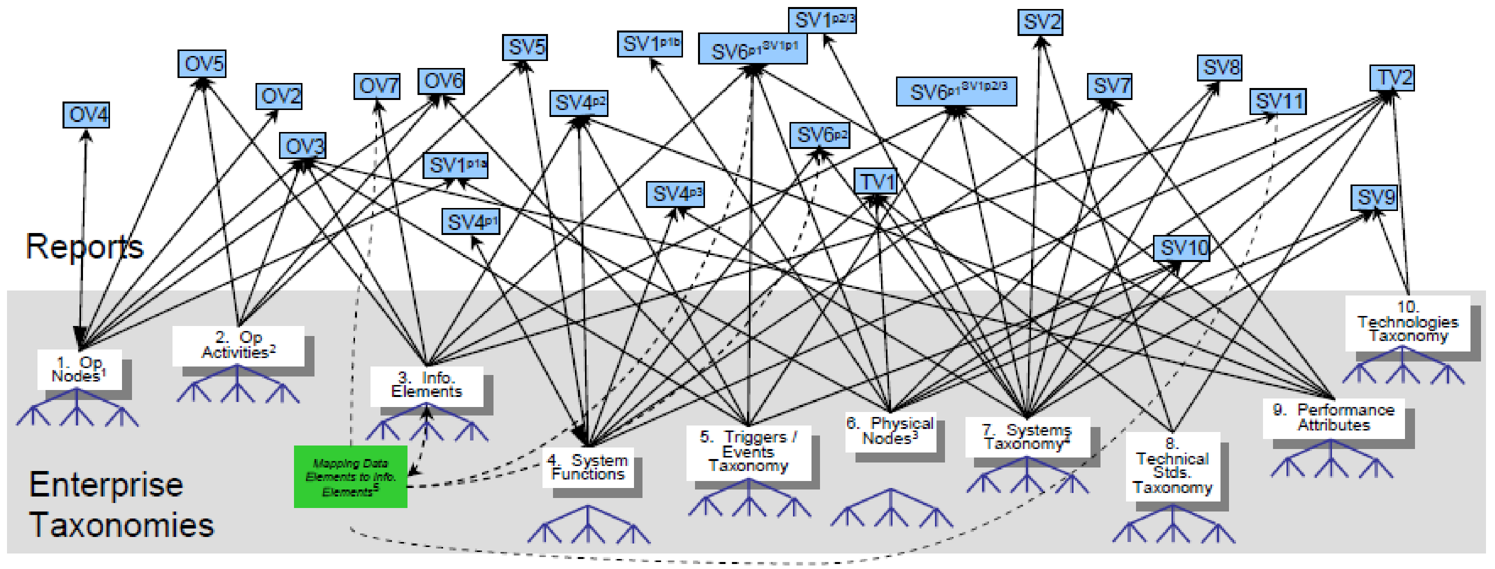


Figure 79. DoDAF 1.0 Core Architecture Data Model Example (OASD 2006).

Taxonomies Mapping to Architecture Reports



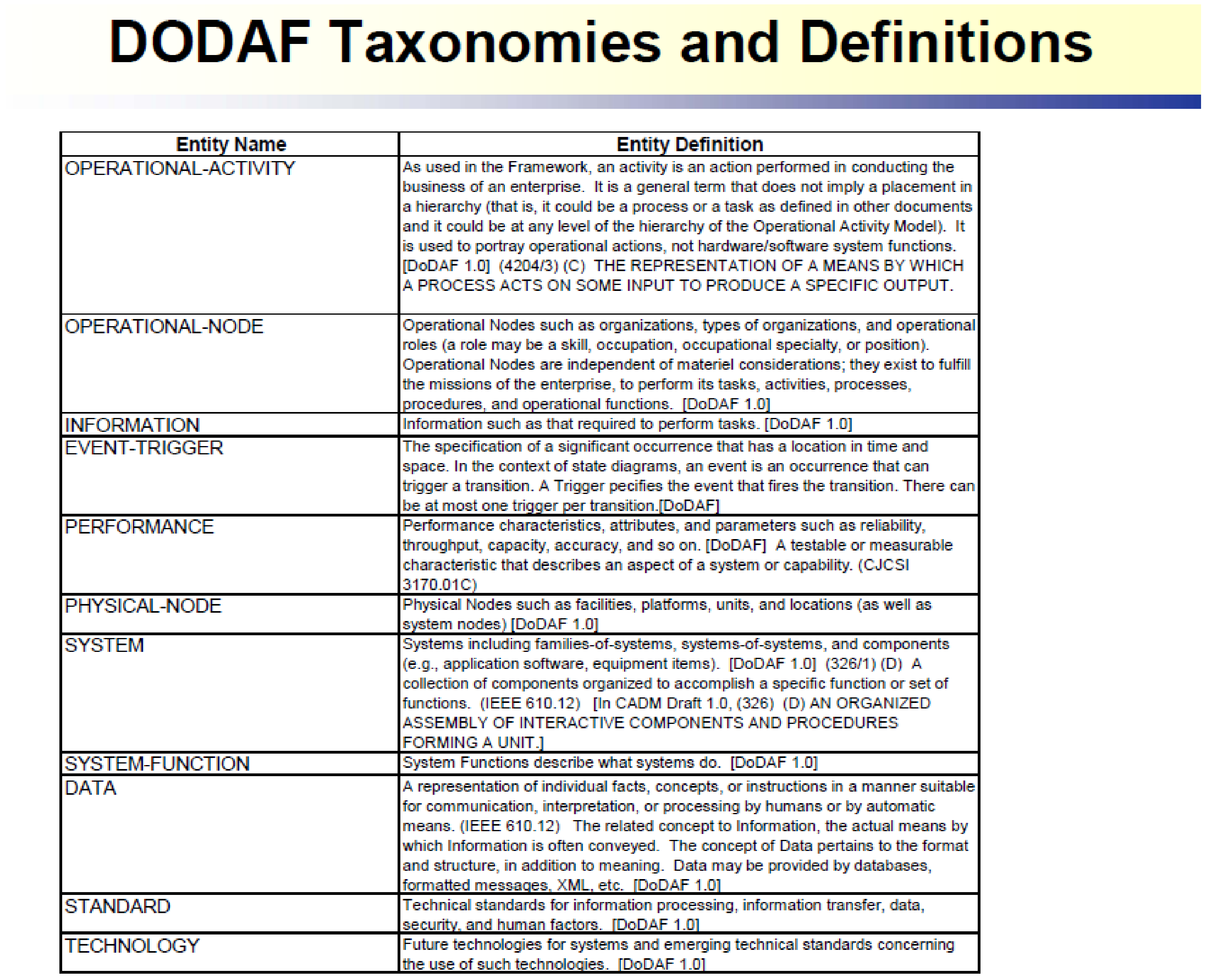
Footnotes:
1. Op Nodes = Organizations, Org Types, or Occupational Specialties
2. Includes Tasks
3. Physical Nodes = Platforms, Facilities, Units, and Locations
4. Includes SoS, FoS, individual Systems, equipment, applications, and software
5. Data Elements including unstructured

Reports Subparts Legend:

SV4^{p1} = System Functions Taxonomy
SV4^{p2} = Function Information Flow
SV4^{p3} = Functional Allocation
SV1^{p1a} = Operational Node – Physical Node Relation
SV1^{p1b} = System Interface Description, Internodal
SV1$^{p2/3}$ = System Interface Description, Intranodal and Intrasystem
SV6$_{p1}^{SV1p1}$ = System Information Exchange Description, Internodal
SV6$_{p1}^{SV1p2/3}$ = System Information Exchange Description, Intranodal and Intrasystem

Taxonomies = { The dictionary of words or terms used in the architecture
The Enterprise common terms of reference
The Enterprise essential elements of architecture
The categories and composition of those objects and elements }

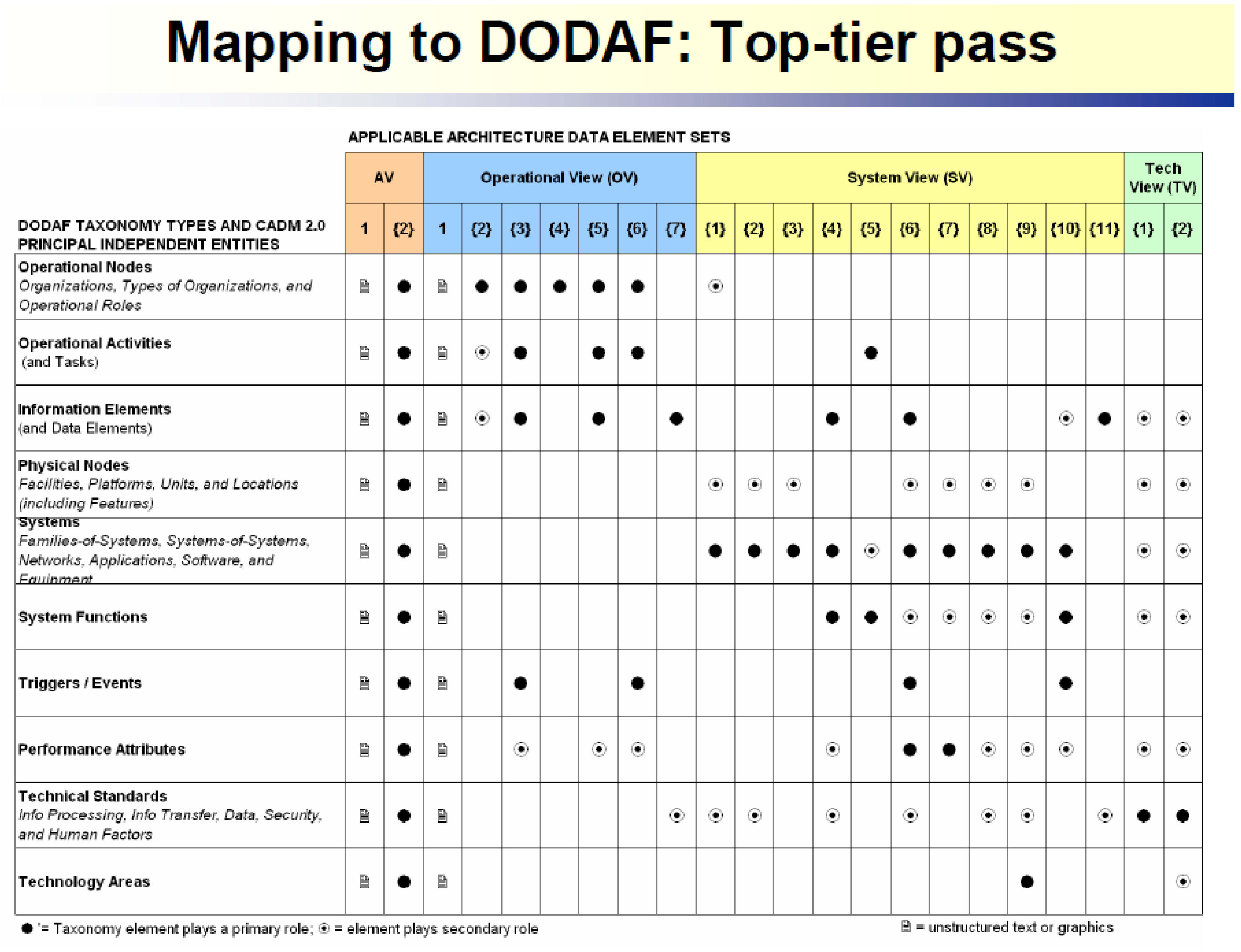
Figure 80. MBSE Tool Graphical Output Example Supporting DoDAF Knowledge Representations (OASD 2006).



DODAF Taxonomies and Definitions

Entity Name	Entity Definition
OPERATIONAL-ACTIVITY	As used in the Framework, an activity is an action performed in conducting the business of an enterprise. It is a general term that does not imply a placement in a hierarchy (that is, it could be a process or a task as defined in other documents and it could be at any level of the hierarchy of the Operational Activity Model). It is used to portray operational actions, not hardware/software system functions. [DoDAF 1.0] (4204/3) (C) THE REPRESENTATION OF A MEANS BY WHICH A PROCESS ACTS ON SOME INPUT TO PRODUCE A SPECIFIC OUTPUT.
OPERATIONAL-NODE	Operational Nodes such as organizations, types of organizations, and operational roles (a role may be a skill, occupation, occupational specialty, or position). Operational Nodes are independent of materiel considerations; they exist to fulfill the missions of the enterprise, to perform its tasks, activities, processes, procedures, and operational functions. [DoDAF 1.0]
INFORMATION	Information such as that required to perform tasks. [DoDAF 1.0]
EVENT-TRIGGER	The specification of a significant occurrence that has a location in time and space. In the context of state diagrams, an event is an occurrence that can trigger a transition. A Trigger pecifies the event that fires the transition. There can be at most one trigger per transition.[DoDAF]
PERFORMANCE	Performance characteristics, attributes, and parameters such as reliability, throughput, capacity, accuracy, and so on. [DoDAF] A testable or measurable characteristic that describes an aspect of a system or capability. (CJCSI 3170.01C)
PHYSICAL-NODE	Physical Nodes such as facilities, platforms, units, and locations (as well as system nodes) [DoDAF 1.0]
SYSTEM	Systems including families-of-systems, systems-of-systems, and components (e.g., application software, equipment items). [DoDAF 1.0] (326/1) (D) A collection of components organized to accomplish a specific function or set of functions. (IEEE 610.12) [In CADM Draft 1.0, (326) (D) AN ORGANIZED ASSEMBLY OF INTERACTIVE COMPONENTS AND PROCEDURES FORMING A UNIT.]
SYSTEM-FUNCTION	System Functions describe what systems do. [DoDAF 1.0]
DATA	A representation of individual facts, concepts, or instructions in a manner suitable for communication, interpretation, or processing by humans or by automatic means. (IEEE 610.12) The related concept to Information, the actual means by which Information is often conveyed. The concept of Data pertains to the format and structure, in addition to meaning. Data may be provided by databases, formatted messages, XML, etc. [DoDAF 1.0]
STANDARD	Technical standards for information processing, information transfer, data, security, and human factors. [DoDAF 1.0]
TECHNOLOGY	Future technologies for systems and emerging technical standards concerning the use of such technologies. [DoDAF 1.0]

Figure 81. DoDAF Taxonomies and Entity Definitions (OASD 2006).



Mapping to DODAF: Top-tier pass

APPLICABLE ARCHITECTURE DATA ELEMENT SETS

DODAF TAXONOMY TYPES AND CADM 2.0 PRINCIPAL INDEPENDENT ENTITIES	AV		Operational View (OV)							System View (SV)											Tech View (TV)	
	1	{2}	1	{2}	{3}	{4}	{5}	{6}	{7}	{1}	{2}	{3}	{4}	{5}	{6}	{7}	{8}	{9}	{10}	{11}	{1}	{2}
Operational Nodes *Organizations, Types of Organizations, and Operational Roles*	🗎	●	🗎	●	●	●	●	●		⊙												
Operational Activities (and Tasks)	🗎	●	🗎	⊙	●		●	●						●								
Information Elements (and Data Elements)	🗎	●	🗎	⊙	●		●		●				●		●				⊙	●	⊙	⊙
Physical Nodes *Facilities, Platforms, Units, and Locations (including Features)*	🗎	●	🗎							⊙	⊙	⊙			⊙	⊙	⊙	⊙			⊙	⊙
Systems *Families-of-Systems, Systems-of-Systems, Networks, Applications, Software, and Equipment*	🗎	●	🗎							●	●	●	●	⊙	●	●	●	●	●		⊙	⊙
System Functions	🗎	●	🗎										●	●	⊙	⊙	⊙	⊙	●		⊙	⊙
Triggers / Events	🗎	●	🗎		●			●							●				●			
Performance Attributes	🗎	●	🗎		⊙		⊙	⊙					⊙		●	●	⊙	⊙	⊙		⊙	⊙
Technical Standards *Info Processing, Info Transfer, Data, Security, and Human Factors*	🗎	●	🗎						⊙	⊙	⊙		⊙		⊙		⊙	⊙		⊙	●	●
Technology Areas	🗎	●	🗎															●				⊙

● '= Taxonomy element plays a primary role; ⊙ = element plays secondary role

🗎 = unstructured text or graphics

Figure 82. *Mapping of DoDAF Taxonomies and Entity Definitions to Framework Views (OASD 2006).*

13.5 MBSE Tool Implications

Similar to the case of the US DoD, the data set in the context of a Defence MCE Project should also contain, for example:

- the set of users of the system;
- a set of objects within the operating environment;
- a set of associations between the objects;
- a set of use case descriptions;
- a set of components comprising notional or abstract system elements; and
- a set of performance metrics.

Data definitions are therefore required to elucidate without ambiguity each data item within the data model (both data elements and relationships between data elements), as well as to elucidate the entire data set populating the data model. This is essential to facilitate shared understanding of the analytic artefacts produced by interrogating the model.

It is requisite that all data entered into the same model be consistent with the data definitions, and that they are agreed and understood by both the modellers entering the data, and the respective users of the artefacts. It is also requisite that the integrity of each data item is known; the data is sufficient; it is in the correct format; and that it is suitable for use in the context intended. These requisites are no different to the employment of any other modelling or analytical techniques to inform decisions relating to capability acquisition.

This raises a number of important questions with respect to such an information repository from a Defence perspective:

- who are the tool-users?
- why are they using the tool?
- what capabilities in the tool are being used?
- what are the tool outputs?
- what methodologies are being employed?
- what are the perceived benefits of using the tool?
- what is the scope of information to be stored in the repository?
- how is the information to be stored – i.e. as data or as artefacts?
- what format is the information to be stored in?
- can the information be stored in the right format?
- what is the data to be used for ?
- can specific useful information sought be searched for and easily found?
- can the information be retrieved and displayed in the right knowledge representation format?

- who is responsible for data entry?
- what skills are required to enter in data?
- who is responsible for interrogating the database and performing systems analysis?
- what skills are required to interrogate and interpret the data?
- how does this information relate to the capability acquisition activity?
- who is responsible for the creation and maintenance of the meta-model or schema?
- how extensible is the meta-model or schema?
- where does the data come from to populate the meta-model or schema?
- how scalable the information repositories are ?
- how shareable is the data in the information repositories?
- how re-usable is the data in the information repositories?
- how does this relate to future MBSE development directions?

13.6 MBSE Tool Considerations - Capability Development Process

The INCOSE MBSE Working Group impetus for MBSE has been to raise the prominence of models in the engineering process to a central and governing role in the specification, design, integration and validation. As described earlier in Section 7, Defence effort during the earliest stages of a capability life cycle is focussed on:

- scoping the extent of the problem to be addressed during a specific Project phase;
- scoping the extent of the solution space in terms of affordability, workforce implications, and perceived risk;
- crafting an RFT documentation package to solicit an industry provided materiel solution; and
- obtaining Government approval to proceed with the tendering to industry for a materiel solution against the approved CDD.

Currently, the approval process is documentation-centric; key documentation products summarising the capability within the CDD being the OCD, the FPS, and the TCD pertinent to the particular MCE Project phase. Together, these three documents provide the basis of the capability development proposals for different options for progression through Defence committees and presentation to Government for approval to proceed to contract acquisition.

The operational concept described in the OCD is deliberately abstract in nature, and is primarily textual, supplemented with some illustrative diagrams. Its principle purpose is to provide an acquirer's point of view. Information is typically presented and collated using corporate word processing and drawing tools provided as part of the Defence Desktop computing Common Operating Environment (COE).

As such, the information contained in the resultant document does not lend itself suitable for representation in a relational database, neither as a list of requirements that needs to be explicitly managed on an ongoing basis, nor as a set of entities and relationships as might be defined in a data model that can usefully portray linkages in meaningful way, for example, to specific requirements in the FPS or other project information for auditing or compliance purposes.

The test concept described in the TCD is also abstract in nature; its purpose to outline the general principles to be applied and the scope of the testing sought for the contracted project activity. Again, this information is primarily textual in nature, and is typically prepared using the same tools as the OCD within the Desktop computing COE.

Similar to the OCD, the information does not lend itself suitable for representation in a relational database, nor does it contain information that needs to be explicitly managed on an ongoing basis, or that can be usefully linked in meaningful ways to other contract information, for example, to support improved traceability or compliance.

The FPS, on the other hand, comprises the head specification for the contract placed on industry for acquisition of the offered industry solution. Because the FPS specifies requirements in list form, the information is suitable for entry into a relational database, either part of a dedicated requirements management tool, or an MBSE tool with requirements management capability. Defence has utilised the Telelogic DOORS requirements management tool for many years to manage the configuration of each FPS, providing a strong track record of demonstrated utility for use of a requirements management tool.[82]

Traceability from the FPS to subservient specifications and to corresponding V&V activity is essential, both to ensure the right solution is built, and ensure the solution is built right. The Verification Assurance Cross-Reference Matrix (VACRM), linking individual requirements to specific verification activity, is a crucial artefact listed within the CDRL, to be provided by industry to support compliance monitoring and contract governance.

Any changes in FPS requirements after contract award must be within Government approved scope, and accompanied by a contract change proposal. This in turn precipitates change control procedures within the industry-based Prime Contractor to ensure the changes are promulgated appropriately throughout the subsequent SE process activity, and they are traceable such that the changes are reflected in the delivered solution, and can be confirmed during the V&V activity.

A tool with the ability to automate specification and V&V traceability could ostensibly be a useful management and governance aid, provided the cost of the tool license and the complexity of use did not mitigate against its utility. However, responsibility for the VACRM lies with industry. As such, it is apparent that there is no requirement for Defence to duplicate this capability for compliance or governance purposes.

Similarly, there are no formalised contractual or management linkages between the contents of the OCD nor the TCD, and subsequent SE management activity or V&V activity undertaken, either by industry or Defence. Since there is no Defence governance requirement for formal verification activity to be performed against the OCD, either by Defence or industry, and the configuration of the OCD is not maintained after contract award, there is no impetus to inform whether the original OCD intent was actually met by

[82] Specific evaluation of the merits or limitation of individual tools currently used by Defence, such as the Telelogic DOORS™ requirements management tool, the Vitech CORE™ SE tool, the IBM System Architect™ tool, Microsoft Visio™, and Microsoft Office™, is out of scope of this report.

the delivered solution. This negates the benefit of using a requirements management tool or a specialist MBSE tool to facilitate improved traceability and configuration management, linking the OCD and TCD through the design and V&V activity. The current approach using a general-purpose word processor to prepare the text-based documentation within the OCD and the TCD would appear to suffice.

The current document-centred approach for Defence project governance, both up to contract award and during the acquisition phase offers benefits over a database approach with regard to:

- it is much simpler to use,
- is easier to distribute for review to the multitude of stakeholders contributing to the governance process,
- it does not require specialist training and support, and
- does not require the additional expense of a specialised tool license to use outside the scope of the Defence-wide, standardised Desktop computing COE.

Defence has expressed interest in exploring the potential of MBSE tools and their underlying databases to improve the analytical rigour supporting CDD development. MBSE tools, as opposed to EA tools, have been used to generate some AUSDAF artefacts from a common database for inclusion in some CDD as an initial step. Another initiative has also trialled the use of MBSE tool scripts to generate project-specific CDD inclusive of the requisite AUSDAF artefacts. However, the broader potential benefit of MBSE will be constrained in a project-centric, document-centric governance regime.

Revisiting the initial notion of a methodology as described in Figure 1, realising the benefits of a MBSE methodology by Defence will require some fundamental conditions to be satisfied, spanning:

- the capability development processes, methods and formalisms employed,
- the environment in which the MBSE tools are deployed, and
- the knowledge base and skills of the process participants.

Notwithstanding, a document-centred approach by Defence prior to RFT issue does not preclude the use of MBSE tools by Defence to provide the requisite analysis and decision support within the capability development process, when progressing from initial beginnings to implementation and support across the various lifecycle stages of the capability.

13.7 MBSE Tool Considerations – AUSDAF

As suggested in Section 13.4, a number of MBSE tools provide SDL vocabulary support for the database schema or integrated data model underpinning EAFs such as the US DoDAF and the UK MODAF. They also provide the ability to generate DoDAF and/or MODAF compliant artefacts from the populated databases.

As described in Section 10, the notion of architecture adopted for the AUSDAF differs significantly from that of the DoDAF and the MODAF. The AUSDAF does not prescribe a specific database meta-model, nor does it provide specific definitions for vocabulary represented in AUSDAF artefacts. Users of the AUSDAF are given flexibility to choose from either or both the DoDAF and MODAF artefacts, and therefore are not constrained to a particular EAF SDL vocabulary or EAF semantics, nor a specific EAF set of artefacts. Finally, the AUSDAF architecture repository comprises an archive of aggregated project artefacts rather than a repository of data attributed to the architecture, so the information contained within the artefacts remains decoupled and independent of the existence of the AUSDAF artefacts.

In the absence of a defined SDL or specific methodology, any tool output will essentially mimic the data entered into the tool; the information may be presented in alternate ways or it might conform to particular textual or graphical formats consistent with the DoDAF or MODAF artefacts. This essentially means that the tool is being used as a drawing and/or information formatting tool rather than being used as an analytical tool to produce specific and/or cross-correlated analytic outputs.

Not all AUSDAF artefacts may necessarily be produced using the same tool and with the same semantics; it is therefore not be possible to discern which information may be cross-correlated between the different AUSDAF artefacts. In the absence of an integrated data model, problems with lack of data cohesion are compounded if the specific project context is not evident. As both context and semantics will differ from project to project, from one tool to another, and from one tool-user to another, the database content in any of the tools will not necessarily be consistent over time or be readily re-usable. Re-usability will depend on the information source, whether data ownership and integrity can be established, and whether the information is being actively managed, and by whom.

Differences between the AUSDAF constructs and MBSE tool capabilities will be compounded if any future updates to AUSDAF propose to introduce additional artefacts over and above that of the MODAF and the DoDAF. If MBSE tool capabilities are not taken into account, and any new AUSDAF artefacts diverge significantly from the MODAF and DoDAF, the new generation MBSE tools may not necessarily be able to provide adequate SDL support, nor be capable of supporting the new presentation formats, which will further limit the potential utility of the tools to generate AUSDAF-compliant artefacts.

As previously suggested in Section 10, it is apparent that the current AUSDAF and proposed future developments do not embrace data-oriented system modelling concepts and notions of information management. The need for specialised licenses and training, and associated additional cost, along with restricted usage as a drawing tool, also casts question on the potential utility and value of new generation MBSE tools to support future AUSDAF artefact preparation.

13.8 MBSE Tool Considerations - IDA

Similar to the AUSDAF, the notion of architecture in the IDA context differs significantly from that of the DoDAF and MODAF. In particular, the IDA has focussed specifically on the notion of a business architecture rather than physical infrastructure, with particular emphasis on business process activity modelling. The activity is entirely agnostic to notions of SE or SW engineering, and formal process linkages to infrastructure acquisition activity and governance requirements are not readily apparent.

Since the IDA utilises reference models in lieu of an integrated data model or meta-model, it has no association with any SDL or ADL, and does not align semantically, neither within domains within the IDA, nor with capability development process vocabulary and concepts. Thus, there is no correlation between information contained in IDA specific artefacts and MCE Project generated artefacts.

Again similar to the AUSDAF instantiation, in the absence of a SDL or methodology for the IDA, any tool output will essentially mimic the data entered in the tool. This means the tool is again being used as a drawing and/or information formatting tool rather than an analysis tool to produce analytic outputs relating to the IDA. The intrinsic value of using modelling tools of the ilk of MBSE in the context of the IDA is not readily apparent in the current instantiation of the IDA.

13.9 System Modelling Challenges

As evident in the discussion thus far, system modelling using modelling languages such as UML and SysML, and applying MBSE analytical precepts, is not a panacea for quickly and efficiently describing the Defence problem space, and investigating solution space possibilities. How the model is to be constructed and populated with data, by whom, and why, are of paramount importance in the systems modelling environment.

The nature of the problem being structured has direct bearing on the suitability and relevance of individual data elements in a data model, and the significance of any particular subset of relationships. It can affect the class structure of objects in the database, and inheritance properties of note. The availability, suitability and quality of source data to populate the data model, and alignment of semantics, are also of critical import.

When setting up a model for a particular MCE Project, it is important to establish the scope of the model, and to understand any interdependencies between different Project models, both in terms of common data elements in the data model, and/or common source data to populate the models.

If modelling is only undertaken within the confines of a specific project, then there can be considerable latitude in tailoring data elements in the model to suit the specific problem at hand. However, for the most part, projects cannot be considered in isolation due to the presence of extensive cross-project and capability interdependencies; pervasive right across the DCP, across all three Services and Joint, and across all elements of FIC. Project-specific models may also be at a different stage of development, affecting availability and quality of information available for analysis at any particular point in time.

This raises many questions regarding managing the exchange and reuse of data between models, including consideration of:

- model understandability,
- data format,
- data searchability and retrievability,
- data suitability
- data semantics,
- data structures,

- data integrity,
- data ownership,
- data configuration management,
- data quality,
- data validation, and
- data archiving.

Pan-project process support is therefore requisite to manage both the modelling environment, and the aggregate data set (i.e. information repository) to leverage benefit from a systems modelling environment in support of improved capability acquisition outcomes. The ability of the MBSE tools to handle the very large scale and complexity of the multi-project data set must also be established.

13.10 MBSE Possibilities for Defence

Defence already provides process and graphical tool support, including training and tool licenses, for both capability development and acquisition and for EA purposes. One of the prescribed tools is UML or SysML capable[83], however, the focus of training is centred around using the tool to draw DoDAF or MODAF artefacts rather than on the underlying modelling language or analytical capabilities. The question of MBSE tool utility therefore centres around the perceived benefits for Defence of transitioning from a general purpose document and drawing production working environment to storing individual data items as objects or blocks in a managed database in a UML or SysML tool-based distributed modelling environment.

It is apparent that the capabilities of the new-generation SysML-based MBSE tools are essentially similar to their UML-based predecessors; the scope of application being extended beyond the traditional notion of a SW system to a more generalised notion of a system, albeit still SW-centric.

As previously discussed in Section 3, SysML is intended to provide superior language support compared with UML to cater for an extended range of SE process steps, and provides improved robustness to cater for other types of systems other than just SW-intensive systems. The recent MBSE tool upgrades to support the UPDM data constructs indicate that tool support to generate the variety of MODAF and DoDAF artefacts will continue to be provided into the future in support of the organisations respective acquisition processes, and the focus will continue to be data-centric.

An AUSDAF meta-model of key architectural elements is a mandatory precursor to realising the analytical benefit inherent in a multi-project distributed MBSE modelling environment. This would pave the way for supporting notions of pan-organisational data management and knowledge management of key information (i.e. data items or architectural elements) - facilitating common data definitions, common semantics, data consistency, data ownership, data configuration control, and data integrity management within the capability development process.

[83] The native ADL within the CORE SE™ tool is proprietary. The IBM Rhapsody System Architect™ tool supports both UML and SysML. Visio and Microsoft Office™ do not support an ADL.

This ostensibly would allow key architectural data to be correlated and re-used in different SoS context, across multiple DCP project boundaries, whilst preserving contextual integrity, and allowing the same data to be promulgated to whichever SoS(s) that might be applicable.

An information repository in the form of an enterprise-wide virtual database populated with key project independent and pervasive architectural elements (i.e. in a pan-organisation, configuration controlled, distributed database of entities and relationships) is also prerequisite.

A tiered approach of abstraction based on types of decisions supported would assist to mitigate the scope and complexity of such an undertaking, where different information (i.e. architectural elements) may be relevant to support different types of decisions, depending on the perspective of organisational responsibility (e.g. system or SoS perspective).

A SoS or Project architecture, in such a setting, would comprise the relevant subset of pan-organisation data or architectural elements at the appropriate level of abstraction, spanning the people, processes, systems, and/or organisations and inter-relationships applicable from the specific SoS or Project perspective.

MBSE tools have the potential to address a wide range of capability acquisition concerns, ranging from articulation of issues, investigation of problems, evaluation of different organisational structures and processes, evaluation of technical system alternatives for form and fitness for purpose, checking for logical consistency, examining the merits of different functional to physical allocations, examining environmental constraints, examining parameter sensitivities to system performance, and distilling key architectural requirements, constraints and interdependencies.

MBSE tools have the potential to provide improved decision support by providing a more structured and formalised approach to systems analysis, which is inherently traceable and repeatable. By providing a Project with the ability to examine the ramifications of various solution alternatives, particularly taking into account other Project and Capability interdependencies, can provide valuable feedback to improve the robustness of the FPS issued to industry in the RFT package for capability acquisition. The tools can also be used to examine the relative merits of alternative FIC element proposals to assist shaping other Defence planning processes to optimise the capabilities outcomes.

14. Conclusions

The underpinning fundamentals of SE and MBSE have been scrutinised in this report in the context of the current Defence capability development process and enterprise architecture initiatives. The capabilities, relevance, and utility of next generation MBSE tools and methodologies have then been examined, contrasting Defence and industry perspectives to reveal potential implications for Defence.

MBSE is proffered by modelling tool vendors to provide improved ability to cope with the more onerous demands of engineering the larger scale and more complex systems as aspired in Defence. MBSE tool vendors posit that MBSE methodologies can offer improved flexibility, consistency and traceability, and facilitate easier upgrade of the associated information set.

However, it is evident there are multiple overlapping MBSE perspectives, somewhat similar, but with different problem foci and different problem solving approaches. These range from a general class of computer-based system and software engineering methodologies supporting bespoke engineering development; to computer-based modelling and analysis supporting operations research; to generating artefacts in accordance with military and commercial enterprise architecture frameworks. If no common agreement can be reached, these differences in perspective can introduce considerable ambiguity within the Defence stakeholder community; this can potentially exacerbate rather than resolve the problems at hand.

It is also evident there is a major divide between Defence, as the customer, and industry as the supplier, in terms of mindsets, skill sets, scale of endeavour, process requirements, constraints and responsibilities. The methods of enquiry and utility of MBSE tools for Defence and industry will therefore differ markedly between the two mindsets and the differing responsibilities.

The differing utility of SE expertise as perceived by Defence and industry is a major differentiator. Due to the distributed responsibilities within the overall Capability Lifecycle, Defence does not have a unified SE approach, with the potential to decouple the capability development process from the traditional systems engineering approach. This in turn introduces additional challenges, and can negate other efforts towards achieving the desired decision outcomes.

Defence faces a number of challenges in developing and applying sufficient SE knowledge and experience both at the high-end platform and the System of Systems engineering levels to effect any major improvement to capability acquisition outcomes. The current approach to capability development does not explicitly define the role of SE, instead, relying on process description in the Defence Capability Development Handbook to drive the capability development and acquisition process. Process governance relies on extensive scrutiny by numerous stakeholders from many perspectives, however, there is no independent scrutiny from a SE perspective to ensure the SE precepts are preserved.

The need to undertake systems analysis is inherent but not explicitly acknowledged within Defence. Of particular import, the capability development and acquisition process is document-centric and governance-oriented. Early capability definition activities are centred on development of the documentation and supporting governance requirements rather than following a traditional SE approach.

Critically, systems analysis in the context of SE cannot be undertaken without SE knowhow. A paucity of SE expertise to garner the multitude of inputs and couch them in the appropriate SE perspectives during the early stages of capability definition can lead to major weakness in the application of the capability development and acquisition process within Defence.

The desire to remain largely conceptual, to allow Defence industry flexibility to offer different competitive solution proposals, is a fundamental constraint in the application of the SE process within Defence. This problem-centric mindset belies the underlying need for system synthesis on a larger scale, with feedback and refinement, taking into account the multitude of project and capability interdependencies and constraints, to ensure the resultant RFT package tendered to Defence industry to deliver each new or modified Defence system is robust in its requirements, and can be mapped to a feasible solution space.

Defence industry on the other hand, has responsibility for developing and delivering a system solution back to Defence. It must therefore have a system solution focus, and provide an appropriate SE development environment to support synthesis, construction, integration and verification of the system solution as specified under contract. The imperative for Defence industry is therefore very different to that of Defence, with markedly different decision support and governance requirements.

The scale of endeavour also has significant implications in terms of the complexity of the problem space and the type of decision support required. Defence has responsibility for managing concurrent development of a multitude of projects, all at differing stages of progress. The degree of abstraction and applicability of relationships and interdependencies differ markedly in the DCP capability planning environment compared with an individual Project contracting environment. The architectural elements and relationships relevant to DCP planning will therefore also differ markedly from those of particular relevance to Defence industry under contract during capability acquisition.

The process duality of the capability development process and enterprise architecture activity, with separate vocabulary and semantics, also creates the potential for a schism in terms of information traceability and management. The inability to tag individual information pertaining to each architectural element as to its origin and original intent, whether engineering derived or EA derived, and the absence of a common information model or meta-model can introduce ambiguity in the data set, rendering information unsuitable for reuse or sharing beyond its original purpose for creation.

Finally, it is important to distinguish between the concept of a methodology that is facilitated by a tool environment and the analytical capability of a tool modelling environment. The established MBSE methodologies such as RUP SE and OOSEM are modelling language dependent and implementation focussed, and thus may offer potential cost savings and efficiencies in industry. However, they do not address the problem space posed to Defence. These established methodologies are therefore not necessarily suited for adoption in the Defence context.

Notwithstanding, MBSE tools can provide a powerful analytic capability to cope with significant design complexity, particularly to investigate capability and project interdependencies and propagation of capability system properties and behaviour. This is contingent on the system models being set up correctly, used by knowledgeable practitioners, and the results are used in the correct context.

MBSE tools supporting object-oriented analysis are particularly useful as they support a 1:1 correlation between objects in the modelling world and real world items. This facilitates ease of aggregation and decomposition of systems and system piece-parts whilst preserving the properties and behaviour of the individual piece-parts, irrespective of the acquirer/supplier perspective, and irrespective of project boundaries. Object-oriented analysis can also provide a powerful means of predicting possible emergent behaviours arising from different interactions of the piece-parts in a complex, multidisciplinary environment.

From a Defence enterprise architecture perspective, the new generation MBSE tools provide a useful means to create Defence Architecture Framework (DAF) artefacts using templates. However, these artefacts are not intended to form a specification, and lack many of the formalisms to drive real-world system implementation. Version 2 of AUSDAF incorporates the artefacts from Version 2 of the US military enterprise architecture framework DoDAF, although underpinning formalisms such the underlying DoDAF data model have not been incorporated.

The MBSE tools are evolving to support future developments of the MODAF and DoDAF towards a common Unified Architecture Framework, embracing data-centric system modelling concepts. The AUSDAF2 view-based orientation does not provide a pathway towards supporting MBSE data-oriented constructs, nor the Unified Architecture Framework as proposed by collaborating partners including U.S. DoD, UK MOD, Swedish DOD, Canadian DND and NATO.

A separate study is recommended to investigate these issues further, including:

- the implications to Australian Defence capability development and acquisition, and Defence enterprise architecture initiatives, of the Unified Architecture Framework proposed developments;
- the feasibility and selection criteria of different information elements for incorporation in an enterprise-wide repository, and associated knowledge management process support requirements, to achieve improved Defence networked Force integration outcomes; and
- provision of formal methodology guidance to leverage the potential of new-generation MBSE tools to achieve improved Defence capability acquisition and integration outcomes.

15. References

AGAF 2009	Australian Government Information Management Office, *Australian Government Architecture Framework*, Version 3.0, Commonwealth of Australia, Canberra, Australia, 2009. [online] URL: www.finance.gov.au/e-government/strategy-and-governance/australian-government-architecture.html
AGIMO 2011	*Australian Government Architecture Reference Models*, Version 3, Australian Government Information Management Office (AGIMO), Australian Government Department of Finance and Deregulation, Canberra, ACT, August 2011. [online] URL: http://agimo.gov.au/files/2012/04/AGA_RM_v3_0.pdf
ANSI/EIA 632:2009	Standard, *Processes for Engineering a System*, Electronic Industries Alliance, Virginia, USA, 2009.
Arsanjani 2004	Arsanjani, Ali, *Service-oriented modeling and architecture*, IBM developerWorks e-zine, IBM, Software Group, 9 November 2004. [online] URL: http://www.ibm.com/developerworks/webservices/library/ws-soa-design1/
Baker et al. 2000	Baker, Loyd, Clemente, Paul, Cohen, Bob, Permenter, Larry, Purves, Bryon, and Salmon, Pete; *Foundational Concepts for Model Driven System Design*, White Paper, INCOSE Model Driven System Design Interest Group, International Council on Systems Engineering, 15 July 2000.
Bell 2008	Bell, Michael, *Service-Oriented Modeling (SOA): Service Analysis, Design, and Architecture*, John Wiley & Sons, 2008.
Bittler & Kreizman 2005	Bittler, R. Scott and Kreizman, Gregg, *Gartner Enterprise Architecture Framework: Evolution, 2005*, Gartner Inc., 21 October 2005. [online] URL: http://www.gartner.com/DisplayDocument?docCode=130849&ref=g_fromdoc
Bittner & Spence 2002	Bittner, Kurt and Spence, Ian, *Use Case Modeling*, Addison-Wesley Professional, 2002.
Blanchard & Fabrycky 1998	Blanchard B.S. and Fabrycky W.W., *Systems Engineering and Analysis*, Third Edition, Prentice Hall International Series in Industrial and Systems Engineering, New Jersey, 1998.
Boehm 1988	Boehm, Barry W., *A Spiral Model of Software Development and Enhancement*, Computer, pp. 61-72, May 1988.

Booch et al. 1999	Booch, Grady, Rumbaugh, James, and Jacobson, Ivar, *The Unified Modeling Language User Guide*, Rational Software Corporation, Addison Wesley Longman Inc., 1999.
Booch et al. 2007	Booch, Grady, Maksimchuk, Robert A., Engel, Michael W., Young, Bobbi J., Conallen, Jim, and Houston, Kelli A., *Object-Oriented Analysis and Design with Applications* (Third Edition), Addison-Wesley, Massachusetts, 2007.
Booher 2003	Booher, H., *Handbook of Human Systems Integration*, Wiley, 2003.
Boucher & Kelly-Rand 2011	Boucher, Michelle and Kelly-Rand, Colin, *System Design Get it Right the First Time*, White Paper, Aberdeen Group Group, August 2011. [online] URL: http://www.aberdeen.com/Aberdeen-Library/7121/RA-system-design-engineering.aspx
Bowler 2010	Bowler, Mark, *Introduction to NCOIC Net-Centric Patterns*, NCOIC, October 27, 2010. [online] URL: http://www.dtic.mil/ndia/2010systemengr/WednesdayTrack3_10955Bowler.pdf
Box 1979	Box, G.E.P., *Robustness in the strategy of scientific model building,* Robustness in Statistics, R.L. Launer and G.N. Wilkinson, Editors., Academic Press, New York, 1979.
BPMN 2011	Standard, *Business Process Notation Modeling*, version 2.0, formal 2011-01-03, Object Management Group, January 2011. [online] URL: http://www.omg.org/spec/BPMN/2.0/PDF
Buede 2000	Buede, Dennis M., *The Engineering Design of Systems: Models and Methods*, First Edition, Wiley Series in Systems Engineering and Management, John Wiley & Sons Inc., 2000.
Buede 2009	Buede, Dennis M., *The Engineering Design of Systems: Models and Methods*, Second Edition, Wiley Series in Systems Engineering and Management, John Wiley & Sons, Inc., 2009.
C4ISRAF 1997	C4ISR Architectures Working Group, *C4ISR Architecture Framework Version 2.0*, U.S. Department of Defense, 18 December 1997.
Cantor 2003a	Cantor, Murray, *Rational Unified Process for Systems Engineering Part 1: Introducing RUP SE Version 2.0*, The Rational Edge e-zine, IBM Software Group, August 2003.
Cantor 2003b	Cantor, Murray, *Rational Unified Process for Systems Engineering Part I1: System Architecture*, The Rational Edge e-zine, IBM Software Group, September 2003.
Cantor 2003c	Cantor, Murray, *Rational Unified Process for Systems Engineering*

	Part II1: Requirements Analysis and Design, The Rational Edge e-zine, IBM Software Group, October 2003.
Carson et al. 2009	Carson, Chris, Fitzgerald, Mike and Hallen, Sue, *Model Driven Development with SysML,* tutorial presentation slides, INCOSE 2009, International Council on Systems Engineering, June 2009.
Chang 1990	Chang, Chen Chung and Keisler, H. Jerome, *Model Theory*. Studies in Logic and the Foundations of Mathematics (3rd ed.). Elsevier, 1990.
Chen 1976	Chen, Peter Pin-Shan, *The Entity-Relationship Model: Towards a Unified View of Data,* ACM Transactions on Database Systems, 1976.
Coopers & Lybrand 1991	*SQA 2000 Methodology Overview,* SQA Version 1.3, Coopers and Lybrand, 1991.
Coopers & Lybrand 1995	*System Development Life Cycle – Overview Reviews and Audits Schedule*, , SQA 2000 Version 1.3, Coopers and Lybrand, 1995.
Croll 2002	Crollo, Paul R.[84], *Interoperability of Systems Engineering Standards – Harmonizing World and National Perspectives*, presentation slides, NDIA Systems Engineering Conference 2002, 24 October 2002.
Dandashi et al. 2006	Dandashi, Fatma, Siegers, Rolf, Jones, Judith, and Blevins, Terry, *The Open Group Architecture Framework (TOGAF) and the US Department of Defense Architecture Framework (DoDAF)*, W061, The Open Group, November 2006.
DCA 2011	*Defence Corporate Architecture (DCA) Part 1 – A Business View*, Version 1.1 (draft), Chief Information Officer Group, Australian Government Department of Defence, Canberra, ACT, 20 June 2011.
DCDH 2012	*Defence Capability Development Handbook 2012*, Version 1.0, Australian Government Department of Defence, Defence Publishing Service, Canberra, ACT, 2012.
DCP 2012	*Defence Capability Plan Public Version 2012*, Capability Development Group, Australian Government Department of Defence, Defence Publishing Service, Canberra, ACT, 15 May 2012. [online] URL: http://www.defence.gov.au/publications/CapabilityPlan2012.pdf
de Marco 1979	de Marco, Tom, *Structured Analysis and System Specification*, Yourdan Inc., 1979.
Densmore &	Densmore, James and Bohn, Tim, *An engineering paradigm for*

[84] Presented by Croll in 2003 in his capacity as Chair, IEEE Software Engineering Standards Committee, and Vice Chair, ISO/IEC JTC/SC7 U.S. TAG.

Bohn 2007	*Service Oriented Architecture*, IBM developerWorks e-zine, IBM Software Group, 15 May 2007.
de Villiers 2001	de Villiers, DJ, *Using the Zachman Framework to Assess the Rational Unified Process*, The Rational Edge e-zine, Rational Software, March 2001. [online] URL: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/mar01/UsingtheZachmanFrameworktoAssesstheRUPMar01.pdf
Dickerson & Mavris 2010	Dickerson, C.E. and Mavris D.N., *Architecture and Principles of Systems Engineering*, Complex and Enterprise Systems Engineering Series, CRC Press, 2010.
Dickinson et al.	Dickinson, David, Vandeville, Joe, and McDonough, Mike, *Making IPPD Real*, Northrup Grumman presentation, Proceedings of CMMI Conference 2008, November 2008. [online] URL: http://www.dtic.mil/ndia/2008cmmi/Track2/Thursday/AM/7152VANDEVILLE.pdf
DNDAF 2012	*DND/CF Architecture Framework (DNDAF) version 2.04*, Canadian Department of National Defence, 2012. [online] URL: http://www.img-ggi.forces.gc.ca/pub/af-ca/index-eng.asp
DoDAF 2009a	DoDAF Architecture Framework Working Group, *The Department of Defense Architecture Framework (DoDAF) Version 2.0 Volume 1: Introduction, Overview, and Concepts, Manager's Guide*, U.S. Department of Defense, 28 May 2009. [online] URL: https://www.us.army.mil/suite/page/454707
DoDAF 2009b	DoDAF Architecture Framework Working Group, *The Department of Defense Architecture Framework (DoDAF) Version 2.0 Volume 2: Architectural Data and Models Architect's Guide*, U.S. Department of Defense, 28 May 2009. [online] URL: https://www.us.army.mil/suite/page/454707
DoDAF 2009c	DoDAF Architecture Framework Working Group, *The Department of Defense Architecture Framework (DoDAF) Version 2.0 Volume 3: DoDAF Meta-model Physical Exchange Specification Developer's Guide*, U.S. Department of Defense, 28 May 2009. [online] URL: https://www.us.army.mil/suite/page/454707
DoDAF 2010	Deputy Chief Information Officer, *The DoDAF Architecture Framework Version 2.02*, U.S. Department of Defense, August 2010. [online] URL: http://dodcio.defense.gov/dodaf20.aspx

DoDAF TWG 2012	*Department of Defense Architecture Framework (DoDAF) v2.0 Technical Working Group (TWG) Overview*, presentation slides, Department of Defense Human Resources Management, 2012. [online] URL: http://www.dodenterprisearchitecture.org/program/Documents/UPDM%20Tutorial%20DoD%20EA%20M%20Hause%20[Compatibility%20Mode].pdf
DoDD 5000.01 2007	Department of Defense Directive (DoDD) Number 5000.01, *The Defense Acquisition System*, Undersecretary of Defense for Acquisition and Technology, U.S. Department of Defense, 20 November 2007.
DoDI 5000.02 2008	Department of Defense Instruction (DoDI) 5000.02, *Operation of the Defense Acquisition System*, U.S. Department of Defense, 8 December 2008.
DoD IPPD Handbook 1998	*DoD Integrated Product and Process Development Handbook*, Office of the Undersecretary of Defense (Acquisition and Technology), US Department of Defense, August 1998.
Doran 2006	Doran, Teresa, *IEEE 1220: for Practical Systems Engineering*, IEEE Computer, 39:92, January 2006.
Doran 2008	Doran, Teresa, *Systems and Software Life Cycle Process Standards: Foundation for Integrated Systems and Software Engineering*, National Defense Industrial Association, 11th Annual Systems Engineering Conference, 23 October, 2008.
DWP 2009	*Defence White Paper 2009 Defending Australia in the Asia Pacific Century: Force 2030*, Australian Government Department of Defence, Defence Publishing Service, Canberra, ACT, 2009. [online] URL: http://www.defence.gov.au/whitepaper
Eeles 2006a	Eeles, Peter, *What is a software architecture?*, IBM developerWorks e-zine, IBM Software Group, 15 February, 2006. [online] URL: http://www.ibm.com/developerworks/rational/library/feb06/eeles/
Eeles 2006b	Eeles, Peter, *Characteristics of a software architect*, IBM developerWorks e-zine, IBM Software Group, 15 March, 2006. [online] URL: http://www.ibm.com/developerworks/rational/library/mar06/eeles/index.html
Eeles 2006c	Eeles, Peter, *The benefits of software architecting*, IBM developerWorks e-zine, IBM Software Group, 15 May, 2006. [online] URL: http://www.ibm.com/developerworks/rational/library/may06/eeles/index.html

Eeles & Cripps 2009	Eeles, Peter and Cripps, Peter, *The Process of Software Architecting*, Addison-Wesley Professional, 2009. [online] URL: http://www.processofsoftwarearchitecting.com/index.php
EIA/IS 731.1:2009	Standard, *Systems Engineering Capability Model*, Electronic Industries Alliance, 2009.
Endrei et al. 2004	Endrei, Mark, Ang, Jenny, Arsanjani, Ali, Chua, Sook, Comte, Phillippe, Krogdahl, Poei, Luo, Min, Newling, Tony, *Patterns: Service-oriented Architecture and Web Services*, IBM International Technical Support Organisation, ibm.com/redbooks, 2004. [online] URL: http://www.redbooks.ibm.com/redbooks/SG246303/wwhelp/wwhimpl/js/html/wwhelp.htm
Estefan 2008	Estefan, Jeff A., *Survey of Candidate Model-Based Engineering (MBSE) Methodologies Rev. B,* INCOSE MBSE Initiative, International Council on Systems Engineering (INCOSE), 23 May 2008. [online] URL: http://www.omgsysml.org/MBSE_Methodology_Survey_RevB.pdf
Eva 1994	Eva, Malcom, *SSADM Version 4: A User's Guide* (2nd Ed), McGraw-Hill, 1994.
Fatolahi & Shams 2006	Fatolahi, Ali & Shams, Fereidoon, *An investigation into applying UML to the Zachman framework*, DOI 10.1007/s 10796-006-7977-8, Inf Syst Front 8:133-143, Springer, 2006. [online] URL: http://isa.sbu.ac.ir/sources/papers/003-fatolahi.pdf
FEA 2012	*The Common Approach to Federal Enterprise Architecture*, U.S. Department of Treasury, 2 May 2012.
FEAF 2001	Federal Architecture Working Group, *A Practical Guide to Federal Enterprise Architecture Version 1.0*, Federal Chief Information Officer Council, U.S. Department of the Treasury, February 2001.
FORCEnet 2004a	*FORCEnet Architecture & Standards Volume I Operational & Systems View Version 1.4*, Office of the Chief Engineer SPAWAR 05, U.S. Department of Defense, 30 April 2004.
FORCEnet 2004b	*FORCEnet Architecture & Standards Volume II Technical View*, Office of the Chief Engineer SPAWAR 05, U.S. Department of Defense, 30 April 2004.
Forrestor 1968	Forrestor, Jay W., *Principles of Systems*, Wright-Allen Press Inc., 1968.

Forsberg & Mooz 1992 Forsberg, Kevin and Mooz, Harold, *The Relationship of Systems Engineering to the Project Cycle*, Engineering Management Journal, Vol. 4, No. 3, pp. 36-43, September 1992.

Frakes et al. 2005 Frakes, William B. and Kang, Kyo, *Software Reuse Research: Status and Future*, IEEE Transactions on Software Engineering, Vol. 31, No. 7, July, 2005.

Frankel et al. 2003 Frankel, David S., Harmon, Paul, Mukerji, Jishnu, Odell, James, Owen, Mark, Rivitt, Pete, Rosen, Mike and Soley, Richard Mark, *The Zachman Framework and the OMG's Model Driven Architecture*, Business Process Trends White Paper, Object Management Group, September 2003.

[online] URL: http://www.omg.org/mda/mda_files/09-03-WP_Mapping_MDA_to_Zachman_Framework1.pdf

Friedenthal et al. 2006 Friedenthal, Sanford, Moore, Alan and Steiner, Rick, *OMG Systems Modeling Language (OMG SysML™) Tutorial*, tutorial presentation slides, INCOSE 2006, International Council of Systems Engineering, 11 July 2006.

[online] URL: http://www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf

Friedenthal et al. 2008 Friedenthal, Sanford, Moore, Alan and Steiner, Rick, *A Practical Guide to SysML* The *Systems Modeling Language*, The Morgan Kaufman OMG Press, 2008.

Gamma et al. 1994 Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John, *Design Patterns Elements of Reusable Object-oriented Software*, Addison-Wesley Professional Computing Series, USA, 1994.

Gershon 2008 Gershon, Sir Peter, *Review of the Australian Government's Use of Information and Communication Technology, Department of Finance and Deregulation*, The Australian Government Information Management Office, August 2008.

[online] URL: http://www.finance.gov.au/publications/ict-review/docs/Review-of-the-Australian-Governments-Use-of-Information-and-Communication-Technology.pdf

Gomma 2011 Gomma, Hassan, *Software Modeling & Design UML, Use Cases, Patterns, & Software Architectures*, Cambridge University Press, 2011.

Gregory 1993 Gregory, Frank Hutson, *Cause, Effect, Efficiency and Soft Systems Models*, Journal of the Operational Research Society 44 (4), pp 149-168, 1993.

Halligan 2011 Halligan, Robert, *Model Based Systems Engineering – A New Methodology or an Old One in a New Jacket*, SysEN #031, Project Performance International Newsletter, May 2, 2011.

Hause 2006 Hause, Matthew, *The SysML Modelling Language*, Fifth European

	Systems Engineering Conference, September, 2006.
Hause 2010	Hause, Matthew, *Model-Based Systems of Systems Engineering with UPDM*, White Paper, Atego, 2010. [online] URL: http://www.omg.org/ocsmp/Model-Based_System_of_Systems_Engineering_with_UPDM.pdf
Hause 2012	Hause, Matthew, *Introducing Artisan Studio*, tutorial presentation slides, Systems Engineering Test & Evaluation Conference, Brisbane, Australia, 2012.
Hause et al. 2012	Hause, Matthew, Brookshier, Daniel and Bleakley, Graham, *UPDM – Unified Profile for DoDAF/MODAF*, presentation slides, UPDM Group, Object Management Group, April 2012. [online] URL: http://www.dodenterprisearchitecture.org/program/Documents/UPDM%20Tutorial%20DoD%20EA%20M%20Hause%20[Compatibility%20Mode].pdf
Hawryszkiewycz 1988	Hawryszkiewycz, I.T., *Introduction to Systems Analysis and Design*, University of Technology, Sydney, Prentice Hall of Australia Pty. Ltd., 1988.
Hilliard 2000	Hilliard, Rich, *Impact Assessment of IEEE 1471 on The Open Group Architecture Framework*, discussion paper, March 30, 2000[85].
Hitchens 2007	Hitchens, Derek K., *Systems Engineering A 21st Century Systems Methodology*, Wiley Series in Systems Engineering and Management, John Wiley & Sons Ltd., 2007.
Hoban & Lawbaugh 1993a	Hoban, Francis T. and Lawbaugh, William M. (editors), *Readings in Systems Engineering, What is a System? NASA's Phased Project Description*, SP-6102,National Aeronautic and Space Administration Scientific and Technical Information Program, Washington D.C., pp. 24-34, 1993.
Hoban & Lawbaugh 1993b	Hoban, Francis T. and Lawbaugh, William M. (editors), *Readings in Systems Engineering, Management Issues in Systems Engineering*, SP-6102, National Aeronautic and Space Administration Scientific and Technical Information Program, Washington D.C., pp. 35-77, 1993.
Hoban & Lawbaugh 1993c	Hoban, Francis T. and Lawbaugh, William M. (editors), *Readings in Systems Engineering, The Systems Engineering Overview and Process,* SP-6102, National Aeronautic and Space Administration Scientific and Technical Information Program, Washington D.C., pp. 8-22, 1993.
Hue 2008	Hue, M.A., *Architecture Practice in Defence – Realising the Seamless NCW Force*, System Engineering Society of Australia, Newsletter No. 47, October 2008, pp 34-46.

[85] This discussion paper was prepared on behalf of The Open Group. Hilliard was a member of the IEEE Architecture Working Group and editor for IEEE 1471 at the time.

Hue 2011	Hue, M.A., *Enterprise Architecture Practice and Systems Engineering – Grappling with the Void*, Proceedings of the Systems Engineering and Test Evaluation Symposium, SESA, 2011.
ICT 2009	*Defence Information and Communications Strategy 2009*, Chief Information Officer Group, Australian Government Department of Defence, Defence Publishing Service, Canberra, ACT, July 2009.
IBM UPDM 2012	*UPDM DoDAF 2.0 Tutorial Overview*, presentation slides, IBM Corporation, 6 February 2012. [online] URL: http://www.ibm.com/developerworks/wikis/display/Rhapsody/UDDM+DoDAF+2.0+Tutorial+Overview
IDA BRM 2011	Directorate of Business Architecture, *Integrated Defence Architecture Business Reference Model v1.0*, Chief Information Officer Group, Australian Government Department of Defence, Canberra, ACT, 2011.
IDEF1X 1993	Standard, *Integration Definition for Information Modelling (IDEF1X)*, Federal Information Processing Standards Publication 184, National Institute of Standards and Technology, 21 December 1993. [online] URL: http://www.idef.com/pdf/Idef1x.pdf
IEEE 610.12-1990	Standard, *IEEE Standard Glossary of Software Engineering Terminology*, Computer Society of the IEEE, 28 September 1990.
IEEE 1220-2005	Standard, *IEEE Standard for Application and Management of the Systems Engineering Process*, Computer Society of the IEEE, New York, 9 September 2005.
INCOSE 2012	*Systems Engineering Handbook – A Guide for System Life Cycle Processes and Activities*, v.3.2, INCOSE-TP-2003-002-03.2.2, International Council on Systems Engineering (INCOSE), San Diego, CA, 14 November 2012.
ISO 10303-233:2012	Standard, *Industrial automation systems and integration – Product data representation and exchange Part 233: Systems engineering*, International Organization for Standardization, Geneva, 2012.
ISO/IEC 15288:2008	Standard, *Systems and software engineering – System life cycle processes*, International Organization for Standardization, Geneva, 2008.
ISO/IEC 1471:2000	Standard, *Recommended Practice for Architectural Description of Software-intensive Systems*, International Organization for Standardization, Geneva, 2000.
ISO/IEC 19501-1:2012(E)	Standard, *Information technology – Object Management Group Unified Modelling Language (OMG UML) Infrastructure*, International Organization for Standardization, Geneva, April

2012.

[online] URL: http://www.omg.org/spec/UML/ISO/19505-1/PDF/

ISO/IEC 19501-2:2012(E) — Standard, *Information Technology – Object Management Group Unified Modelling Language (OMG UML) Superstructure,* International Organization for Standardization, Geneva, April 2012.

[online] URL: http://www.omg.org/spec/UML/ISO/19505-2/PDF/

ISO/IEC/IEEE 42010-2011 — Standard, *Systems and Software Engineering – Architecture Description*, International Organization for Standardization, Geneva, 2011.

ISO/IEC JTC1/SC7/WG7 2002 — ISO/IEC JTC 1/SC 7/WG 7 N0560, *Systems Engineering Study Report 2002*, International Organization for Standardization, Geneva, 2002.

Jansma & Jones 2006 — Jansma, Patti A. and Jones, Ross M., *Advancing the Practice of Systems Engineering at JPL*, IEEE Aerospace Conference, Big Sky, Montana, 4-11 March 2006.

[online] URL: http://trs-new.jpl.nasa.gov/dspace/handle/2014/40111

Jensen et al. 2011 — Jensen, Jeff C., Chang, Danica H. and Lee, Edward A., *A Model-Based Design Methodology for Cyber-Physical Systems*, Proceedings of 7th International Wireless Communications and Mobile Computing Conference (IWCMC), IEEE, pp. 1666-1671, July 2011.

JIA 2010 — *Joint Intelligence Architecture - Architecture Reference Book*, v1.0 (draft for discussion), Chief Information Officer Group, Australian Government Department of Defence, Canberra, ACT, 7 July, 2010.

Jones et al. 2011 — Jones, N. A., Ross, H., Lynam, T., Perez, P., and Leitch, A., *Mental models: an interdisciplinary synthesis of theory and methods*. Ecology and Society 16(1), p 46, 2011.

[online] URL: http://www.ecologyandsociety.org/vol16/iss1/art46/

Josey 2009 — Josey, Andrew, *TOGAF™ : A Comprehensive Overview*, The Open Group, 2 March 2009.

Kent 2002 — Kent, Stuart, *Model Driven Engineering*, Proceedings of the Third International Conference: Integrated Formal Methods, 2002.

Knight et al. 2006 — Knight, Michele, Vencel, Les, and Moon Terry, *A Network Centric Warfare (NCW) Compliance Process for Australian Defence*, DSTO-TR-1928, DSTO Edinburgh, SA, August 2006.

Kobryn & — Kobryn, Cris and Sibbald, Chris, *Modeling DoDAF Compliant*

Sibbald 2004	*Architectures*, White Paper, Telelogic, 25 October 2004. [online]: URL: http://www.incose.org/mdwest/presentations/DoDAF050525.ppt
Kruchten 1995	Kruchten, P., *The 4+1 view model of architecture*, IEEE Software, 12 (6), pp. 42-50, November 1995.
Landherr 1997	Landherr, Stefan, *Software Engineering Issues*, Proceedings of Defence Seminar on Indigenous Software Support, Australian Government Department of Defence, Canberra, 21 November, 1997.
Lapkin 2005	Lapkin, Anne, *Gartner's Enterprise Architecture Process and Framework Help Meet 21st Century Challenges*, 8 November 2005. [online] URL: http://www.gartner.com/resources/133100/133132/gartners_enterprise_architec_133132.pdf
Levis 2000	Levis, Alexander H., *C4ISR Architecture Framework and Implementation*, DSTO Salisbury Course Notes, 25 February 2000.
Lieberman 2003a	Lieberman, Ben, *The art of modeling Part I: Constructing an analytical framework*, Rational Edge e-zine, IBM Rational Software, August 2003. [online] URL: http://www.docin.com/p-71440586.html
Lieberman 2003b	Lieberman, Ben, *The art of modeling Part II: Model organisation and construction*, Rational Edge e-zine, IBM Rational Software, November 2003. [online] URL: http://www.ibm.com/developerworks/rational/library/content/RationalEdge/nov03/f_art_bl.pdf
Lieberman 2004	Lieberman, Ben, *The art of modeling Part III: Visual composition*, Rational Edge e-zine, IBM Software Group, January 2004. [online] URL: http://www.biologicsoftwareconsulting.com/articles/Art%20of%20Modeling%20III.pdf
Long 2010	Long, David, *A Model-Based SE Roadmap for Developing DoDAF 2.0 Architectures*, Tutorial Presentation, Systems Engineering and Evaluation Conference, 2010.
Long & Scott 2011	Long, David, and Scott, Zane, *A Primer for Model-Based Systems Engineering*, 2nd Edition, Vitech Corporation, October 2011. [online] URL: http://www.vitechcorp.com/resources/MBSE.shtml
Lykins et al. 2000	Lykins, Howard, Friedenthal, Sanford and Meilich, Abraham,

	Adapting UML for an Object Oriented Systems Engineering Method (OOSEM), Proceedings of the Tenth Annual International Symposium of the International Council on systems Engineering (INCOSE), 2000.
Maier 1998	Maier, M. W., *Architecting Principles for Systems-of-Systems*, Systems Engineering, 1:4, pp 267-284, 1998.
Maier & Rectin 2002	Maier, Mark W., & Rechtin, Eberhardt, *The Art of Systems Architecting*, Second Edition, CRC Press, 2002.
Mar 1992	Mar, B.W., *Systems Engineering Basics*, Proceedings of the National Council of Systems Engineering, 1992.
Mar 1997	Mar, B.W., *Back to Basics Again – A Scientific Definition of Systems Engineering*, Proceedings of the Seventh Annual International Symposium of The International Council on Systems Engineering (INCOSE), 1997.
Mar & Morais 2002	Mar, Brian W., and Morais, Bernard G., *FRAT – A Basic Framework for Systems Engineering*, Proceedings of the Twelfth Annual International Symposium of The International Council on Systems Engineering (INCOSE), August 2002.
Marca et al. 1987	Marca, D., & McGowan, C., *Structured Analysis and Design Technique*, McGraw-Hill, 1987.
Martin 1998	Martin, James N., *Overview of the EIA 632 Standard – "Processes for Engineering a System" Proceedings of INCOSE*, 30 September 1998.
McDaniel 2012	McDaniel, David, *History of the DoDAF to 2.02*, Proceedings of Workshop for ACT-IAC EA SIG, Architecture & Infrastructure Directorate, Office of the Chief Information Officer, U.S. Department of Defense, 20 July 2012. [online] URL: http://www.actgov.org/knowledgebank/newknowledgebank/Events%20Programs%20and%20Initiatives/OSD%20DoDAF%20History%20-%20David%20McDaniel-OSD%2007-20-12.pdf
MIL-STD-499B	Military Standard, *Systems Engineering Management (draft)*, U.S. Department of Defense, 24 August 1993.
Muchandi 2007	Muchandi, Veer, *Applying 4 + 1 View Architecture with UML 2*, white paper, FCG Software Services, 2007. [online] URL: http://www.sparxsystems.com/downloads/whitepapers/FCGSS_US_WP_Applying_4+1_w_UML2.pdf
MODAF 2010	*MOD Architecture Framework v1.2*, UK Ministry of Defence, May 2010. [online] URL: https://www.gov.uk/mod-architecture-framework

NAF 2007	*NATO Architecture Framework v3,* Annex 1 to AC/322-D(2007)0048, NATO Consultation, Command and Control Board, 2007.
NASA 2007	*NASA Systems Engineering Handbook*, NASA/SP-2007-6105 Rev 1, National Aeronautics and Space Administration, NASA Centre for AeroSpace Information, 31 December 2007. [online] URL: http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20080008301_2008008500.pdf
NBA2020+ 2011	Directorate of Business Architecture, *Networked Battlespace Architecture 2020+ Architecture Reference Book*, Version 1.1, Chief Information Officer Group, Australian Government Department of Defence, Canberra, ACT, 15 April 2011.
NCOIC 2008	*NCOIC Interoperability Framework (NIF™) and NCOIC Patterns Overview*, presentation slides, NCOIC, August 2008. [online] URL: https://www.ncoic.org/technology/deliverables/nif/NIF.ppt#488
NCWIIS 2010	Director Network Centric Warfare Development, *Network Centric Warfare Integration and Implementation Strategy 2010*, Capability Development Group, Australian Government Department of Defence, Defence Publishing Service, Canberra, ACT, 2010.
OASD 2006	*DOD's Core Architecture Data Model (CADM) Version 2.0 Overview*, presentation slides, Silver Bullet Solutions Inc., 25 April 2006. (prepared for OASD U.S. Department of Defense under contract DAAB07-03-D-B009 (CISA)
Okon 2012	Okon, Walt, *Unified Architecture Framework DoDAF Strategic Direction*, Architecture & Interoperability Directorate, Office of the Chief Information Officer, U.S. Department of Defense, 20 July 2012. [online] URL: http://www.actgov.org/knowledgebank/newknowledgebank/Events%20Programs%20and%20Initiatives/OSD%20DoDAF%20Unified%20Architecture%20Framework%20-%20Walt%20Okon-OSD%2007-20-12.pdf
Peraire et al. 2007	Peraire, Cecile, Edwards, Mike, Fernandes, Angelo, Mancin, Enrico, and Carroll, Kathey, *The IBM Rational Unified Process for System z*, IBM International Technical Support Organisation, ibm.com/redbooks, July 2007.
Peters 1987	Peters, Lawrence, *Advanced Structured Analysis and Design*, Prentice-Hall Series in Software Engineering, New Jersey, 1987.

Pidd 2004	Pidd, Michael, Editor, *Systems Modelling Theory and Practice*, John Wiley and & Sons Ltd., 2004
PMBOK 2009	*A Guide to the Project Management Body of Knowledge (PMBOK®) – Fourth Edition*, Project Management Institute, January 2009. [online] URL: http://www.pmi.org/PMBOK-Guide-and-Standards.aspx
Purcell 2009	Purcell, CDRE Mark, RAN, *Delivery of a Single Enterprise Architecture*, presentation slides, MilCIS, 2009. [online] URL: http://www.milcis.com.au/milcis2009pdf/presentations/3.4b%20-%20Mark%20Purcell.pdf
Quatrani & Palistrat 2006	Quatrani, Terry and Palistrat, Jim, *Visual Modeling with IBM Rational Software Architect and UML*, developerWork Series, IBM Press, 26 May 2006.
Rational 2001	*Rational Unified Process Best Practices for Software Development Teams*, Rational Software White Paper, TP026B, Rev 11/01, Rational Software, 2001.
Rechtin 1991	Rechtin, Eberhardt, *Systems Architecting Creating & Building Complex Systems*, Prentice Hall Inc., New Jersey, 1991.
Ritchey, 1991	Ritchey, Dr. Tom, *Analysis and Synthesis On Scientific Method – Based on a Study by Bernhard Riemann*, Systems Research, Vol. 8, No. 4, pp 21-41, Thesis Publishers, 1991.
Rosenblueth & Norbet (1945)	Rosenblueth, Arturo and Wiener, Norbert, *The Role of Models in Science*, Philosophy of Science, Vol. 12, No. 4, pp 316-321, University of Chicago Press, October 1945.
Royce 1970	Royce, W., *Managing the Development of Large Software Systems*, Proceedings of IEEE WESCON 26, August 1970, Institute of Electrical and Electronic Engineers Inc., pp 1-9.
Ryder & Flanigan 2005	Ryder, Chris and Flanigan, Dave, *Applying the Systems Engineering Method for the Joint Capabilities Integration and Development System (JCIDS)*, presentation slides, 8th Annual Systems Engineering Conference, National Defence Industry Association, 27 October 2005. [online] URL: http://www.dtic.mil/ndia/2005systems/thursday/ryder.pdf
Sage 1992	Sage, Andrew P., *Systems Engineering*, Wiley Series in Systems Engineering, John Wiley & Sons Inc., 1992.
Sage & Rouse 2009	Sage, Andrew P. and Rouse, William B., *Handbook of Systems Engineering and Management*, Second Edition, John Wiley & Sons Inc., NJ, 2009.
Sayles 2003	Sayles, Allen., *Development of Federal Enterprise Architecture*

Framework using the IBM Rational Unified Process and the Unified Modeling Language, The Rational Edge e-zine, Rational Software Corporation, January 2003.

[online] URL: http://www.ibm.com/developerworks/rational/library/content/03July/2500/2787/2787_arch_framework.pdf

Schmidt 2006 — Schmidt, Douglas C., *Model-Driven Engineering*, IEEE Computer, Vol. 39, No. 2, pp. 25-31, February 2006.

SEBoK 2012 — Pyster, A., D., Olwell, N., Hutchison, S., Enck, J., Anthony, D. Henry and A. Squires (eds.). *Guide to the Systems Engineering Body of Knowledge (SEBoK) version 1.0.1.* Hoboken, NJ: The Trustees of the Stevens Institute of Technology ©2012, 2012.

[online] URL: http://www.sebokwiki.org/1.0.1/index.php?title=Main_Page

SEF 2001 — Systems Management College, *System Engineering Fundamentals*, Department of Defense, Defense Acquisition University Press, Fort Belvoir, Virginia, 2001.

Sessions 2007 — Sessions, Roger, *A Comparison of the Top Four Enterprise Architecture Methodologies*, Object Watch, May 2007.

[online] URL: http://msdn.microsoft.com/en-us/library/bb466232.aspx

SF 2010 — Strategy Policy Division, *Strategy Framework 2010*, Australian Government Department of Defence, Defence Publishing Service, Canberra, ACT, 2010.

Shamieh 2011 — Shamieh, Cathleen, *Systems Engineering for Dummies*, IBM Limited Edition, Wiley Publishing Inc., Indianapolis, 2011.

Sheard & Lake 1998 — Sheard, Sarah A. and Lake, Jerome G., *Systems Engineering Standards and Models Compared*, Proceedings of the Eighth International Symposium on Systems Engineering, Vancouver, Canada, pp. 589-6051998.

SIE 2010 — *Single Information Environment (SIE) Architectural Intent 2010 Integrated Defence Architecture*, Chief Information Officer Group, Australian Government Department of Defence, Defence Publishing Service, Canberra ACT, 2010.

Sparks 2012 — Sparks, E., *Course Notes for Human Systems Integration*, Defence Academy of the United Kingdom, Defence College of Management and Technology, 2012.

Sparx Systems 2007a — *Using UML Part One – Structural Modeling Diagrams*, UML Tutorials white paper, Sparx Systems, 2007.

[online] URL:

http://www.sparxsystems.com/downloads/whitepapers/UML_Tutorial_Part_1_Introduction.pdf

Sparx Systems 2007b	*Using UML Part Two – Behavioural Modeling Diagrams*, UML Tutorials white paper, Sparx Systems, 2007. [online] URL: http://www.sparxsystems.com/downloads/whitepapers/UML_Tutorial_Part_2_Introduction.pdf
Sowa et al. 1992	Sowa, J.F., and Zachman, J.A., *Updated version of Zachman EAF. Extending and Formalising the Framework for Information Systems Architecture*, IBM Systems Journal: Volume 31, Number 3, Page 590, 1992.
Stojanovic 2005	Stojanovic, Zoran, *A Method for Component-Based and Service-Oriented Software Systems Engineering*, Doctoral Dissertation, Delft University of Technology, The Netherlands, 2005.
SWEBOK 2004	Abran, Alain and Moore, James W. (Executive Editors), *SWEBOK Guide to the Software Engineering Body of Knowledge*, IEEE Computer Society, USA, 2004. [online] URL: http://www.computer.org/portal/web/swebok/
SysML 2006	Standard, *OMG Systems Modeling Language (OMG SysML™) Specification*, ptc/06-05-04, Object Management Group, May 2006. [online] URL: http://www.sysml.org/docs/specs/OMGSysML-FAS-06-05-04.pdf
SysML 2012	Standard, *OMG Systems Modeling Language (OMG SysML™) Version 1.3*, formal/2012-06-01, Object Management Group, June 2012. [online] URL: http://www.omg.org/spec/SysML/1.3/PDF/
Taha 2002	Taha, Hamdy A., *Operations Research: An Introduction*, Seventh Edition, Prentice Hall Inc., New Jersey, 2002.
TOGAF 2009	*The Open Group Architecture Framework Version 9 "Enterprise Edition"*, The Open Group, 2009. [online] URL: http://www.opengroup.org/architecture/togaf/
UML 2011a	Standard, *OMG Unified Modeling Language™ (OMG UML), Infrastructure Version 2.4.1*, formal/2011-08-05, Object Management Group, August 2011. [online] URL: http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/
UML 2011b	Standard, *OMG Unified Modeling Language™ (OMG UML), Superstructure Version 2.4.1*, formal/2011-08-06, Object Management Group, August 2011.

	[online] URL: http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/
UPDM 2012	Standard, *Unified Profile for DoDAF and MODAF (UPDM) Version 2.0*, Object Management Group, January 2012.
	[online] URL: http://www.omg.org/spec/UPDM/2.0/PDF/
VA 1992	*A Tutorial on the Zachman Framework for Enterprise Architecture*, presentation slides, US Department of Veterans Affairs, 1992.
Weilkiens 2006	Weilkiens, Tim, *Systems Engineering with SysML/UML Modeling, Analysis, Design,* The OMG Press, 2006.
White 2004	White, Stephen A., *Process Modeling Notations and Workflow Patterns*, White Paper, IBM Corporation, January 2004.
	[online] URL: http://www.omg.org/bpmn/Documents/Notations_and_Workflow_Patterns.pdf
Wymore 1993	Wymore, A. Wayne, *Model-Based Systems Engineering: an introduction to the mathematical theory of discrete systems and to the tricotyledon theory of systems design*, CRC Press Inc. Florida, 1993.
Yannopoulos et. al. 2010	Yannopoulos, Matt and King, Graham, *The Integrated Defence Architecture, The Models and their Value*, presentation slides, MilCIS, 2010.
	[online]: URL
	http://www.milcis.com.au/milcis2010pdf/MilCIS2010presentations/1.4b%20-%20Graham%20King.pdf
Yourdan 1989	Yourdan, Edward, *Modern Structured Analysis*, Yourdan Press Computing Series, 1989.
Yourdan & Constantine 1978	Yourdan, Edward & Constantine, Larry L., *Structured Design Fundamentals of a Discipline of Computer Program and Systems Design*, Yourdan Press, New York, 1978.
Zachman 1987	Zachman, J.A., *A framework for information systems architecture*, IBM Systems Journal , 26(3), pp 276-292, 1987.
Zachman 2003	Zachman, John A., *The Zachman Framework™: A Primer for Enterprise Engineering and Manufacturing*, e-book, Zachman International, March 2003.

Appendix A: Tools in the SE Development Environment

A.1. Introduction

Numerous tools are available in the form of SW applications, hosted on general-purpose desktop computers, to support the various activities within the SE process. Some simple tools are available as freeware, downloadable from the Internet; other more specialised tools require SW license purchase to use, some of which can be very expensive[86].

These tools offer a wide variety of functionality, typically tailored to specific domain disciplines to support the methods of enquiry, synthesis, and process responsibilities associated with the particular work in progress.

EA tools can typically support generation of a number of diagrams which can describe numerous aspects of an organisation, from the operation of business processes, organisational structure, information flows, IT systems and technical infrastructure. They can be useful to describe, analyse and visualise relationships between business domains, which can allow stakeholders to assess and communicate the impact of organisational decisions or changes within or between domains. The tools typically support different viewpoints to address different stakeholder concerns, to make comparisons and to highlight cause and effect relationships.

In contrast, SE, SW engineering, and MBSE tools typically provide significant engineering process support, from initial articulation of requirements, to system synthesis, implementation and V&V. Information is typically stored in a database with an integrated data model schema, where it can be viewed from various viewpoints as required for decision support. An example of the Rational range of SE management tools provided by IBM is shown in Figure A.1. An example of the Atego Artisan Studio tool interoperability support in the engineering development environment is provided in Figure A.2.

Some examples of commercial tools are provided in the following Table A.1. Additional information on vendors and tools specifically supporting SysML can be found at the OMG web-site located at URL: http://www.omgsysml.org/.

[86] For example, a SW license priced at $50,000 for a single seat can cost an organisation several million dollars for a Corporate license.

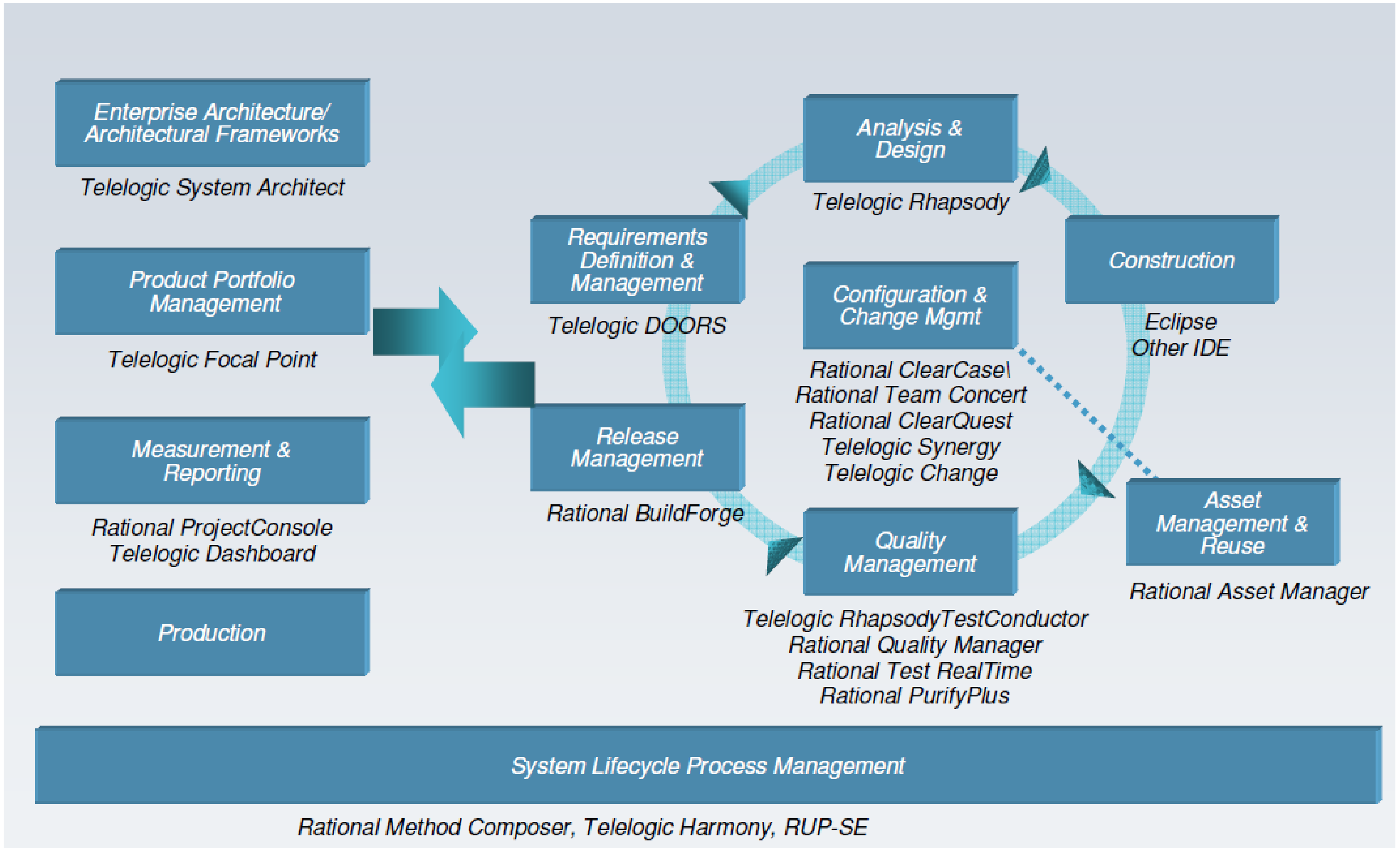


Figure A.1. Range of IBM Tools Supporting Collaborative System Life Cycle Process Management (Carson et al. 2009).

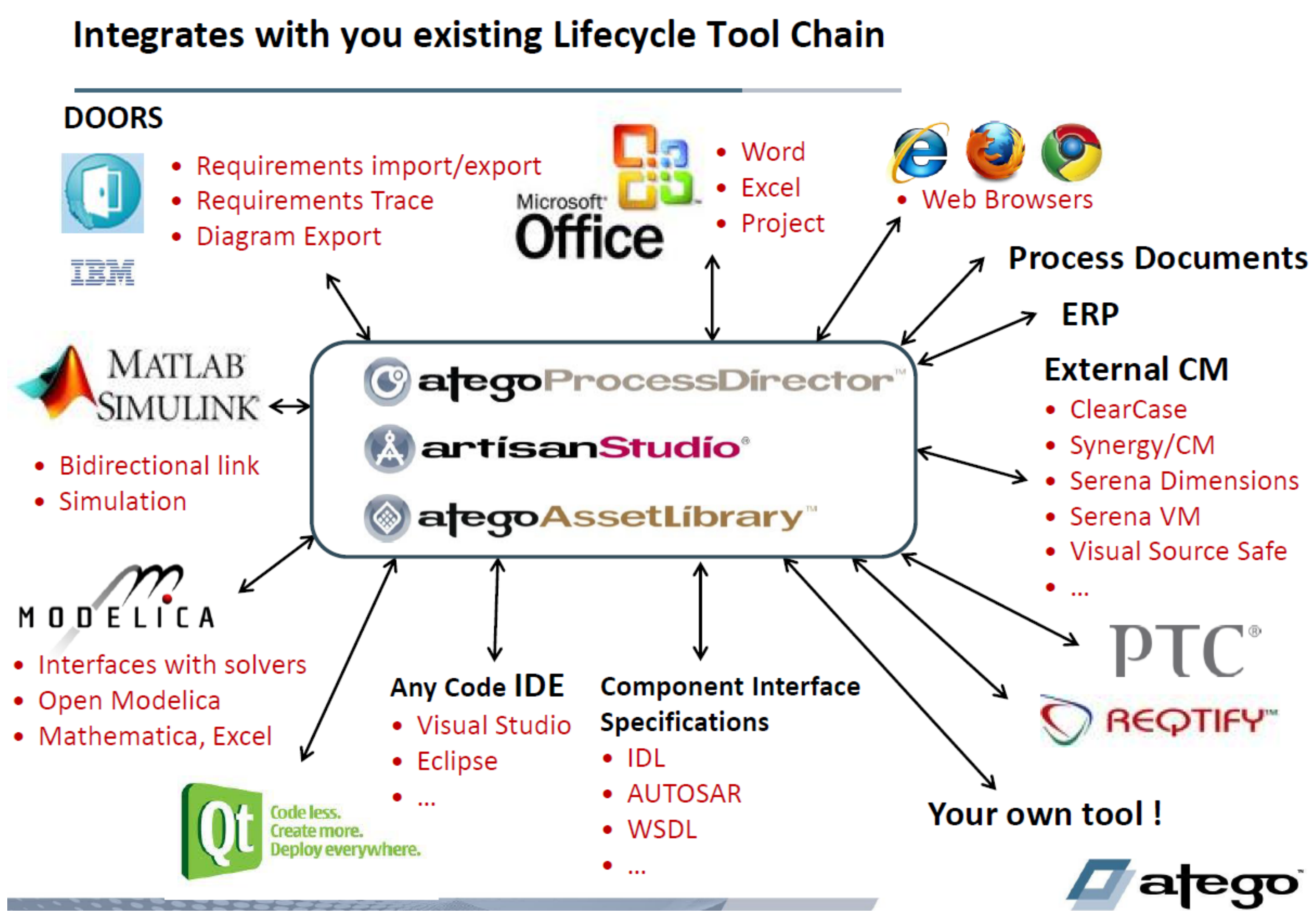


Figure A.2 Example of Artego Artisan Studio Tool Interoperability in the SE Environment (Hause 2012).

Table A.1 Snapshot of Tool Vendors and Products.

Tool Name	Vendor	Information [online] URL:
Enterprise Architecture Tools		
Artisan Studio	Atego	http://www.atego.com/products/artisan-studio/ Tool supports UML, SysML, UPDM modelling languages. Tool interfaces with IBM Rational DOORS® tool.
ABACUS	Avolution	http://www.avolution.com.au/releases/0809_archimate.html Drawing tool. Tool supports Archimate® modelling language.
IBM® Rational® System Architect®	IBM Corporation	http://www-01.ibm.com/software/awdtools/systemarchitect/ Modelling tool supports UML.
Office®	Microsoft	http://office.microsoft.com/en-au/?CTT=97 Drawing tool. Method uses Word, EXCEL and PowerPoint templates. Modelling language agnostic.
Visio®	Microsoft	http://office.microsoft.com/en-au/visio/ Drawing tool. Method uses stencils and templates. Modelling language agnostic.
BPMN®	OMG	http://www.omg.org/spec/BPMN/ Modelling language standard.
iServer Process Modeller	Orbus Software	http://www.orbussoftware.com/business-process-analysis/iserver/process-modeler Drawing tool supports BPMN modelling language standard. Method uses MS Visio stencils and templates.
SmartDraw	SmartDraw	http://www.smartdraw.com/ Drawing tool. Modelling language agnostic.
Enterprise Architect	Sparx Systems	http://www.sparxsystems.com.au/products/mdg/tech/archimate/archimate.html Modelling tool supports Archimate®, BPMN, UML, SysML, and UPDM modelling languages. Tool interfaces with IBM Rational DOORS®.
Archimate®	The Open Group	http://www.opengroup.org/subjectareas/enterprise/archimate Modelling language standard.
Engineering Tools		
IBM Rational® Requirements Composer®	IBM Corporation	http://www-01.ibm.com/software/awdtools/rrc/
IBM® Rational® DOORS®	IBM Corporation	http://www-01.ibm.com/software/awdtools/doors/
SE Tools		

Vitech CORE®	Vitech Corporation	http://www.vitechcorp.com/products/core.shtml Tool has proprietary SDL.
SW Engineering Tools		
OSATE	Carnegie Mellon Software Engineering Institute	http://www.aadl.info/aadl/currentsite/ Tool supports SAE AADL modelling language.
Eclipse UML2 Tools	Eclipse Foundation	http://www.eclipse.org/modeling/mdt/?project=uml2 Tool supports UML modelling language.
ER/Studio® XE2	Embarcadero	http://www.embarcadero.com/products Tools support UML modelling language.
IBM® Rational® RequisitePro	IBM Corporation	http://www-01.ibm.com/software/awdtools/reqpro/
IBM® Rational® Rhapsody® Architect for Software	IBM Corporation	http://www-142.ibm.com/software/products/us/en/ratirhaparchforsoft Tool supports UML modelling language.
IBM® Rational® Team Concert™	IBM Corporation	http://www-01.ibm.com/software/rational/products/rtc/
Architecture Analysis and Design Language (AADL)	SAE	http://www.sei.cmu.edu/architecture/tools/index.cfm Modelling language standard.
New Generation MBSE Tools (SysML-based)		
UModel®	Altova	http://www.altova.com/umodel.html Tool supports BPMN, UML, SysML modelling languages. Tool interfaces to Visual Studio and Eclipse.
Artisan Studio	Atego	http://www.atego.com/products/artisan-studio/ Tool supports UML, SysML, UPDM modelling languages. Tool interfaces with IBM Rational DOORS® tool.
IBM® Rational® Rhapsody® Architect for Systems Engineers	IBM Corporation	http://www-142.ibm.com/software/products/us/en/ratirhaparchforsystengi Tool supports both UML and SysML modelling languages.
MagicDraw	No Magic	http://www.nomagic.com/products/magicdraw.html Tool supports BPMN, UML, SysML, UPDM modelling languages.
Enterprise Architect	Sparx Systems	http://www.sparxsystems.com.au/products/mdg/tech/archimate/archimate.html Modelling tool supports Archimate®, BPMN,

		UML, SysML, and UPDM modelling languages. Tool interfaces with IBM Rational DOORS®.
VP-UML	Visual Paradigm	Tool supports BPMN, UML, SysML modelling languages.
Vitech GENESYS™	Vitech Corporation	http://www.vitechcorp.com/products/genesys.shtml Tool supports the Vitech STRATA methodology. Tool supports SysML modelling language.

Appendix B: Object-oriented Modelling Language Origin and Concepts

B.1. Object-Oriented Programming Language Paradigm – Origin

A form of expression in terms of objects was evolved during the 1980s and 1990s; the main building block of SW being an object or a class. When first introduced, a number of proprietary approaches were developed using different diagramming techniques to provide visualisation of the SW systems. However, the rapid uptake in the SW engineering and IT communities spawned the formation of a consortium in 1989 called "The Object Management Group" (OMG)[87]. OMG initially comprised eleven large IT vendors, including Sun Microsystems, Apple Computer, IBM, Hewlett-Packard and Data General. OMG has since assumed responsibly for specification and management of numerous international standards, and owns numerous trademarks which have subsequently shaped the object-oriented paradigm.

The goal of OMG from the outset was to create a heterogeneous distributed object standard to encourage development of conforming products providing true interoperability between products. This was to be achieved by developing a specification for a common and interoperable object model with methods and data that work using "all types of development environments" and "all types of data" (presumably amongst the consortium members in the first instance) [88]. The desire was to create a defacto graphical modelling language for SW development that provided the semantics and notation for object-oriented problem solving (Dickerson & Mavris 2010).

The formal publication of graphical modelling language UML as ISO/IEC 19501 in 1997 by OMG marked a significant milestone in SW modelling and SW development. UML was initially developed by a consortium of large IT vendors separate from OMG, including IBM, Microsoft, Hewlett-Packard and Oracle Corporation. This was spearheaded by Booch, Rumbaugh and Jacobson[89] from Rational Software. Their primary objective was to combine and streamline several of the second generation proprietary approaches to provide a common object-oriented language construct that was non-vendor specific; possibly to encourage uptake and promote broader acceptance of the radically different object-oriented mindset. They then sought formal publication under the umbrella of OMG to facilitate the widespread adoption of UML across the international engineering and IT communities.

[87] [online] URL: http://en.wikipedia.org/wiki/Object_Management_Group; URL: http://www.omg.org/spec/UML/

[88] [online] URL: http://en.wikipedia.org/wiki/Object_Management_Group; URL: http://www.omg.org/spec/UML/

[89] Rumbaugh originally developed the OMT methodology, Booch developed OODA, and Jacobson developed Objectory OOSE The UML notation is a union of the graphical syntax of OMT, OODA, and Objectory, including notions of use cases from Objectory, class diagrams from OMT and OODA, state-machine diagrams from OODA and OMT, as well representations from OMT, OODA, Objectory and other methods.

B.2. Developing an Object-Oriented Architecture

Because the object and class groupings are discretionary and abstract notions in software engineering, the portrayal of an object-oriented system requires particular consideration, for example to decide:

- what would be a good object-oriented architecture (i.e. SW structure or composition and organisation of the SW piece-parts);
- what artefacts should be created; what properties or attributes are relevant;
- how and where the interfaces should be defined; and
- how should the properties and attributes be measured.

The UML graphical modelling language provides a standardised way and vernacular using numerous complementary perspectives to visualise a systems architecture, including in terms of its:

- activities,
- actors,
- business processes,
- logical components,
- database schemas,
- programming language statements, and
- SW components (some of which may be reused, or be reusable, or adhere to particular design patterns).

It has been specifically crafted to cater for the entire SW life cycle, including visualising, specifying, constructing, and documenting SW-intensive systems, from single user, single process SW applications to multiple user, concurrent, distributed SW-intensive systems (Booch et al. 1999).

UML is typically used to support faster and more cost-effective realisation of high quality, complex custom SW applications. The HW inferred is typically the host platform environment and other IT and telecommunications (i.e. ICT) infrastructure, such as databases and/or application and network servers. These subsystems and components can be readily procured off-the-shelf (OTS), either COTS or MOTS, and utilise commonly used standards (either open or proprietary) in their HW and SW products. This facilitates ease of selection to provide interface compatibility between piece-parts, with the required functionality, to enable assembly of a SW-intensive system with little or no customisation of the HW environment.

A plethora of vendor tools and methods supporting SW, systems and enterprise architecture modelling has since been spawned based on UML and newer generation object-oriented programming languages. This has led to the creation of a global user base of object-oriented SW and object-oriented modelling expertise spanning both the commercial and the Defence sectors. It has also stimulated interest in developing object-oriented modelling concepts further for application to a broader set of systems problems in engineering and larger concerns affecting entire enterprises.

A sample of vendors and tools is provided in Appendix A to this report.

B.3. Object-Oriented Modelling with UML

Originally released in 1997, UML[90] is a specification for graphical diagrams to support object-oriented development methods such as Model Driven Architecture (MDA) and Model Driven Development (MDD). Version 2.4.1, released in 2011, describes 14 different basic diagram types, organised into two categories as follows (UML 2011a) (UML 2011b) (Gomma 2011)[91]:

1. **Structural diagrams** – are used to define the static architecture. These comprise static constructs such as "classes", "objects", and "components", and the relationships between these constructs, illustrated in the form of:
    - Class diagrams (or structural diagrams) – these define the basic building blocks of the model, describing the structure of a system in terms of the system's classes, their attributes, and the relationships among the classes;
    - Object diagrams – show how instances of structural elements are related, i.e. they show a complete or partial view of the structure of the modelled system at a specific time;
    - Profile diagrams – are used at the meta-model level to show stereotypes as classes with the <<stereotype>> stereotype , and profiles as packages with the <<profile>> stereotype;
    - Package diagrams – are used to describe how a system is split into "logical groupings" or packages by showing the dependencies among these groupings;
    - Composite Structure diagrams – are used to model the internal parts contained within a class and the collaborations relationships between the parts that the internal structure makes possible;
    - Component diagrams – depict the components used to assemble and realise a physical system, including their interfaces and dependencies, both in "white box" and "black box" form;
    - Deployment diagrams – show the mapping of the SW onto the HW and to depict the HW topology in a real-world setting.
2. **Behavioural diagrams** – are used to represent the dynamic architecture. These comprise behavioural constructs such as activities, states, timelines, and the messages that are exchanged between different objects. These diagrams are used to represent the interactions between various model elements and instantaneous states over a specific time period. These are illustrated in the form of:
    - Use case diagrams – depict the use cases and actors tied to detailed scenario descriptions. They are used to reason about the desired behaviour of the

[90] UML Version 1.4.2 was released as international standard ISO/IEC 19505 in 2005.
[91] [online] URL: http://en.wikipedia.org/wiki/Unified_Modelling_Language

system as seen by its end users and other stakeholders.

- Activity diagrams – similar to flow charts, these depict program flows and complex business logic, including actions, decision points, branching, merging and parallel processing;
- State Machine diagrams – depict the instant states of an object defined by a class;
- Communication diagrams – show communication between objects at runtime during a collaboration sequence;
- Sequence diagrams – show a sequence of messages passed between objects on a vertical timeline;
- Timing diagrams – these specifically address modelling of performance. They depict the amount of time allowed for a participant to receive events and and switch between the states, and how long a participant can stay in a specific state;
- Interaction Overview diagrams – provide an overview of how several interactions work together to implement a system concern.

Definitions and specific examples of UML diagrams are provided in Appendix C to this report.

The concept of design patterns is also well supported in UML. When viewed from the outside, a design pattern is rendered as a parameterised collaboration, providing a set of abstractions whose structure and behaviour work together to perform a useful function.

The collaboration's parameters name the elements that a user of this pattern must bind. When viewed from the inside, the design pattern is the collaboration, and is rendered in terms of its structural and behavioural parts. The inside of the collaboration can be modelled in UML using a set of class diagrams for the structural aspect, and a set of interactions for the behavioural aspect. The collaboration's parameters name certain of these structural elements; when the design pattern is bound in a particular context, these structural elements are instantiated using abstractions from that context (Booch 1999).

B.4. The 4 + 1 Architecture View using UML2

Using UML2 notation, the architecture of a SW system can be represented in graphical form as a combination of:

- The structural elements and their interfaces that comprise or form the SW system;
- The behaviour represented by collaboration among the structural elements; and
- The composition of structural and behavioural elements into larger subsystems, where such compositions are guided by desired abilities (non-functional requirements) such as usability, maintainability, performance, and security, which apply across all the functional elements.

These are brought together in a 4 + 1 View Architecture Representation developed by

Krutchens at Rational Software as illustrated in Figure B.1; each architecture view highlighting information relevant to different stakeholders, masking the remaining information that is not relevant, but preserving the integrity of all the information collated within the SW architecture (Krutchen 1995), (Muchandi 2007).

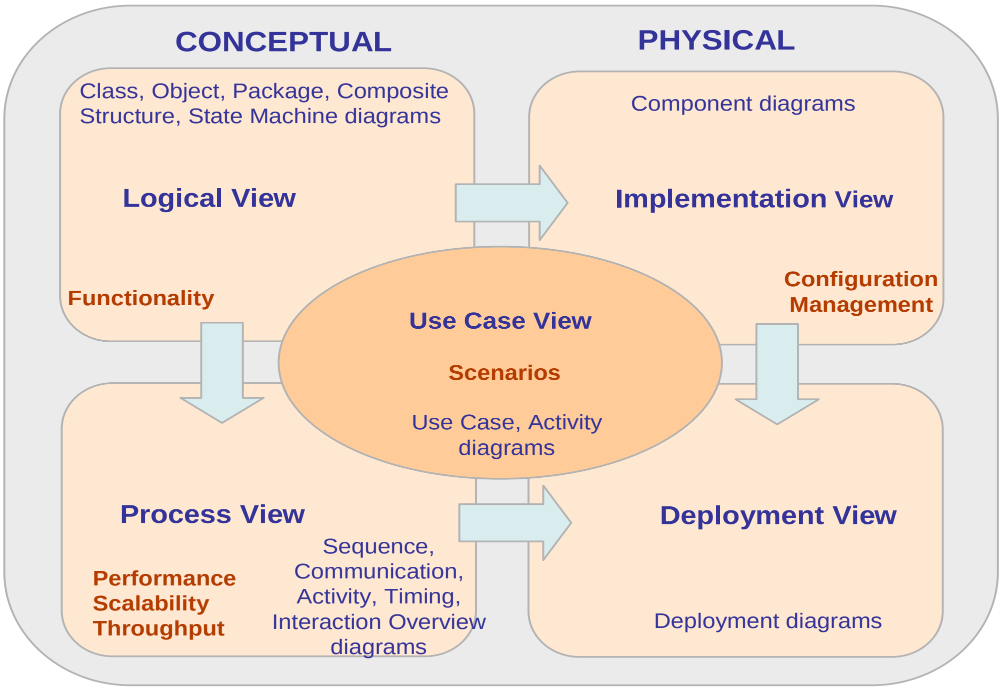


Figure B.1. *4 + 1 View Architecture Representations.*

The Logical View of the SW application architecture describes the kinds of objects that are used to realise the system implementation It provides a functional decomposition perspective of the Application, and is used to support functional analysis at different levels of abstraction. The Logical View provides an object-oriented representation of the application's functionality in terms of its structural elements, key abstractions and mechanisms, separation of concerns, and distribution of responsibilities as shown in Figure B.2. The logical architecture can be represented at different levels of abstraction, and progressively evolved through recursive iterations.

The SW application can be logically partitioned in two dimensions, either vertically into significant functional areas (allocated to specific subsystems), or horizontally into layers of different responsibility (e.g. service layer, data access layer, security layer, and API layer, as may defined in a designated reference model). Here, structural elements are represented as classes or objects and their relationships.

The Process View of the SW Application architecture provides a process decomposition perspective. A process is a group of tasks that form an executable unit: a SW system is partitioned into sets of tasks. Each task is a thread of control that executes with collaboration among different structural elements (as represented in the Logical View). It is therefore able to show the main abstractions from the Logical View executing over a thread as an operation as shown in Figure B.3.

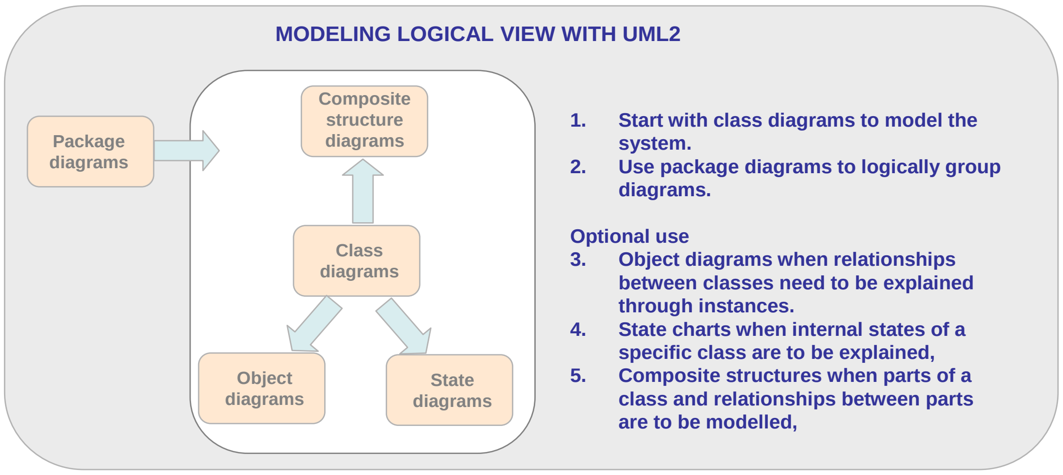

MODELING LOGICAL VIEW WITH UML2
Package diagrams
Composite structure diagrams
Class diagrams
Object diagrams
State diagrams
1. Start with class diagrams to model the system.
2. Use package diagrams to logically group diagrams.
Optional use
3. Object diagrams when relationships between classes need to be explained through instances.
4. State charts when internal states of a specific class are to be explained,
5. Composite structures when parts of a class and relationships between parts are to be modelled,

Figure B.2. Modelling the Logical View with UML2.

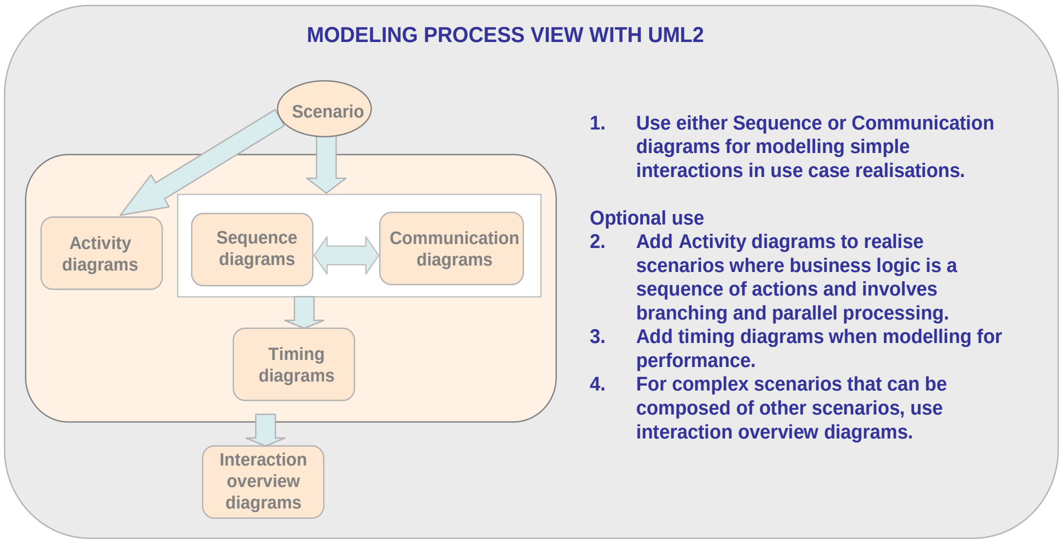

MODELING PROCESS VIEW WITH UML2
Scenario
Activity diagrams
Sequence diagrams
Communication diagrams
Timing diagrams
Interaction overview diagrams
1. Use either Sequence or Communication diagrams for modelling simple interactions in use case realisations.
Optional use
2. Add Activity diagrams to realise scenarios where business logic is a sequence of actions and involves branching and parallel processing.
3. Add timing diagrams when modelling for performance.
4. For complex scenarios that can be composed of other scenarios, use interaction overview diagrams.

Figure B.3. Modelling the Process View with UML2.

The Implementation View provides a subsystem decomposition perspective. This is a view of the system's architecture that encompasses the components used to assemble and realise the actual physical system. This view supports configuration management of the SW modules and their organisation in the development environment, where the SW is packaged into components that can be developed and tested both separately and together as an assembled system.

The Deployment or Physical View depicts the mapping of the SW application onto the host HW platform, and reveals the HW topology as processing nodes on which the SW is executing. It therefore depicts the physical disposition of the artefacts in the real-world setting.

The Use Case View brings together the other four views in the context of a scenario, and describes the behaviour of the system as seen by its end users and other stakeholders in the scenario.

B.5. Systems Modelling Language (SysML) – Origins

In simple terms, SE[92] entails the specification, design, implementation and verification of engineering solutions to systems problems, where the solution is realised as a technical system. The solution implementation may or may not contain significant HW content, and may often require HW customisation as well as SW development.

While the UML standard purports to support the engineering development of entire systems, it is essentially SW-centric, supporting various computer-aided SW engineering (CASE) methodologies. Notably, the language constructs do not support many of the broader notions in model-based SE including system requirements; system verification and validation; engineering and project management, and other underlying management disciplines, including configuration management, quality management, security management, and intellectual property management.

Similarly, the SW language constructs do not support the subordinate disciplines of electrical/electronic HW engineering and mechanical engineering with regard to specialised design, prototyping, construction, design verification, environmental qualification, production manufacturing, and logistics support; all of which are relevant to underpin notions of model-based SE.

At the time UML was being developed, the advantages of using a standard modelling language to tackle complex SW engineering problems were also visible to key members in the broader international SE community. This prompted the International Council on Systems Engineering (INCOSE) to approach OMG to propose a joint development to define a general purpose modelling language based on UML, but dedicated to SE, supporting the notion of "MBSE"[93].

In 2001, INCOSE in collaboration with OMG initiated the development of an OMG specification for a customised version of UML specifically for MBSE usage, changing some of the concepts and deleting superfluous constructs, together with adding some SE specific UML extensions. Widespread contributions to the specification and implementation of SysML were received from INCOSE, international IT and modelling tool vendors, Defence industry, commercial engineering organisations, and academia. These included Motorola, Northrop Grumman Corporation, Telelogic AB, Artisan Software Tools, IBM, The Boeing Company, Lockheed Martin Corporation, BAE SYSTEMS, Ratheon, THALES, Israel Aircraft Industries, National Aeronautics and Space Administration (NASA), and Georgia Institute of Technology.

[92] A detailed examination of SE processes is provided in Section 5 – Systems Engineering Concepts.
[93] [online] URL: http://en.wikipedia.org/wiki/Systems_Modeling_Language; URL: http://www.omgsysml.org/

The resultant Systems Modelling Language implementation, OMG SysML[94], was finally released in 2007. The current version of OMG SysML is v 1.3, which was released in June, 2012, based on UML 2.3, released in May 2012 (SysML 2012) . The relationship between UML and SysML is illustrated in Figure B.4. Differences in diagrammatic support between the two standards are shown in Figure B.5[95] (Hause 2006).

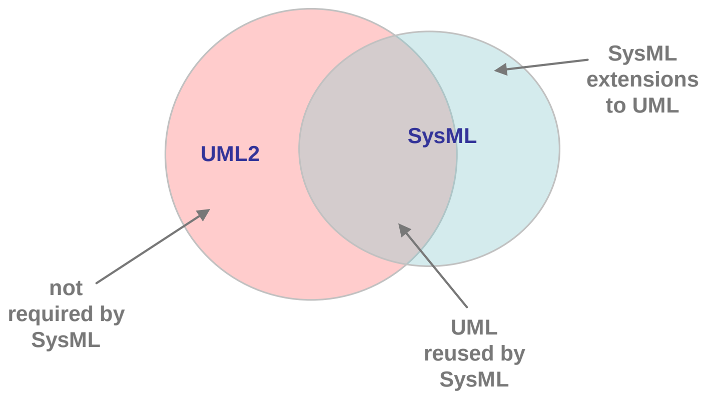


Figure B.4. Relationship between UML and SysML Language Constructs.

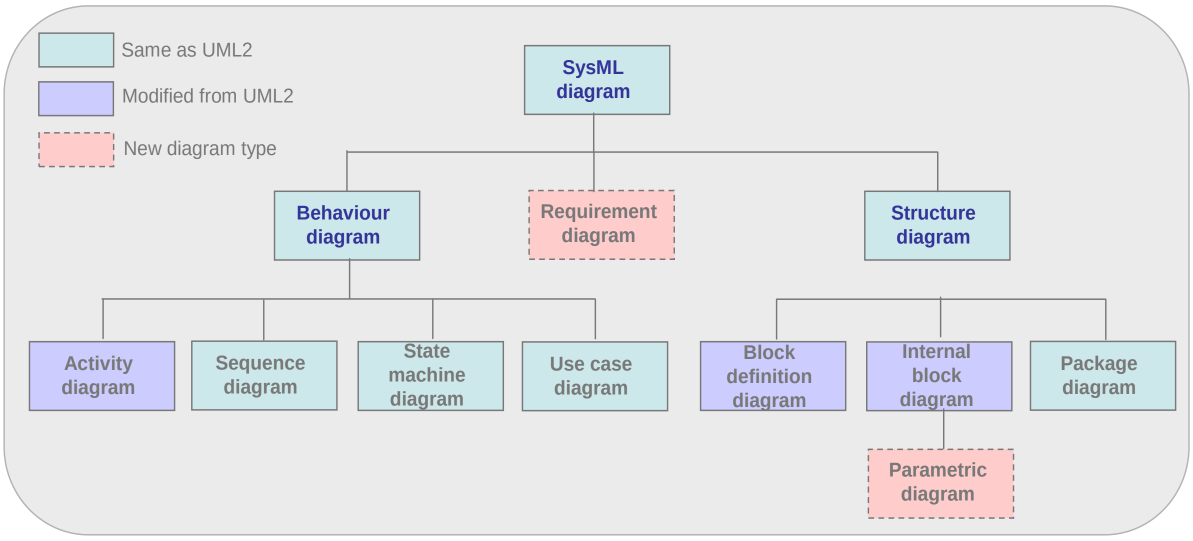


Figure B.5. Diagram Support for UML2 vs. SysML.

Because OMG SysML is specified as a profile of UML, which has come the defacto industry standard for modelling SW-intensive systems, OMG SysML has been implemented as a plug-in for a number of UML modelling tools. A number of other IT vendors have also

[94] Two iterations of the SysML specification were released for implementation, known respectively as SysML and OMG SysML. The first iteration of SysML was made available as an open source specification. This was later refined and re-released as OMG SysML to differentiate between the two versions. OMG SysML is a trademark owned by OMG. The term SysML is used interchangeably with OMG SysML in this report unless specifically stated otherwise.
[95] [online] URL: http://www.sysml.org

announced plans for updating their modelling tools to support SysML[96]. A wide UML user base with broad vendor tool support is therefore already established to facilitate ready uptake of this latest development supporting notions of MBSE.

Similarly, because OMG SysML is a derivative of UML 2, it was crafted to support exchange of SysML models using the OMG developed XML Metadata Interchange (XMI) standard, also used for UML model exchange.[97]

OMG is also progressing development of the new ISO 10303/AP-233 data exchange format standard (known as STEP, Standard for the Exchange of Product model data) for exchanging and sharing information between other SE applications and tools. The intent of AP-233 is to support the entire system development cycle ranging from requirements definition to system verification and validation. Different application areas range from engineering analysis, algorithm design, planning tools, testing tools, software design, mechanical computer-aided design (CAD) and electrical computer aided engineering (CAE) (Hause 2006), (Friedenthal 2008).

STEP provides a neutral computer interpretable representation of product data throughout the life cycle of a product, independent of any particular system. STEP is actually a suite of international standards built around an integrated architecture of domain specific application protocols (AP) and generic integrated resources. The AP's break STEP into manageable and comprehensible "chunks" that can be more readily implemented. However, the language constructs between AP-233 and SysML are not entirely aligned, as shown in Figure B.6[98], particularly relating to engineering management concerns.

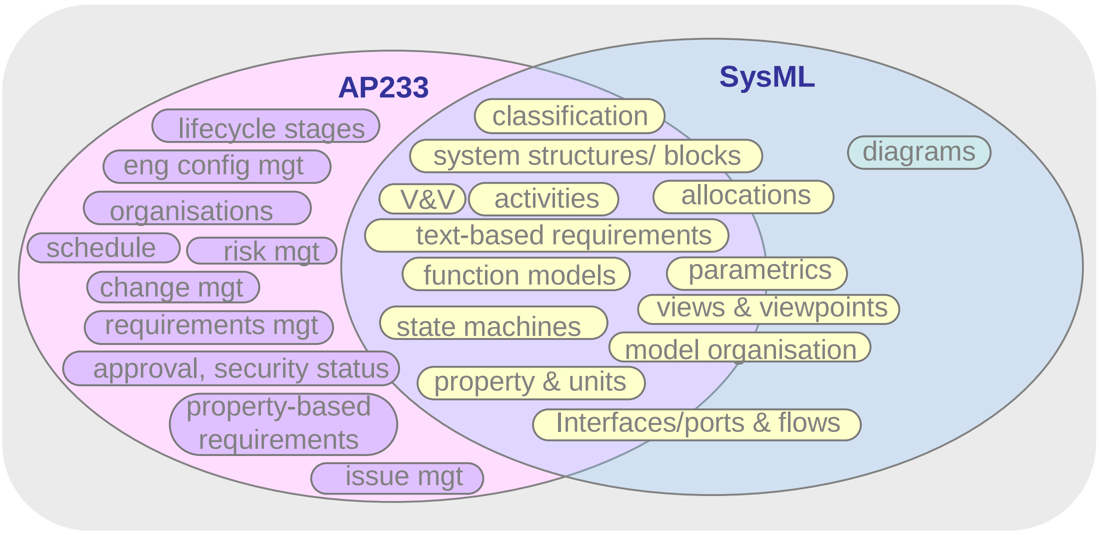


Figure B.6. SysML/AP-233 Data Overlaps.

[96] Lists of tool vendors who support, or have announced plans to support SysML and OMG SysML can be found on the SysML Forum and OMG SysML Forum websites [online] URL: http://www.sysmlforum.com/and URL: http://www.omgsysml.org/ respectively.
[97] Different modelling tool vendors have interpreted the XMI standard differently. The resulting incompatibilities make it difficult to pass models from one tool vendor to another using XMI.
[98] [online] URL: http://www.omgwiki.org/OMGSysML/doku.php?id=sysml-ap233:mapping_between_sysml_and_ap233).

A number of specific MBSE methodologies are now being developed using the respective MBSE tools and systems modelling languages.

B.6. Systems Modelling Language (SysML) – Concepts

The primary goal set by INCOSE for SysML was to define:

"a standard modelling language for systems engineering to analyse, specify, design, and verify complex systems, is intended to enhance system quality, improve improve the ability to exchange systems engineering information amongst tools, and help bridge the semantic gap between systems, software, and other engineering disciplines".

This MBSE approach sought to provide the ability to model a much wider range of systems, encompassing HW, SW, information processes, personnel, and facilities, not just SW-intensive systems. It is purported to support SE process activities across the entire system life cycle including specification, analysis, design, verification and validation activities for a broad range of systems and SoS types. However, the language constructs thus far still focus only on driving the technology solution. They do not explicitly include non-technical SE process considerations such as engineering management and project management, which focus on SE process activity management, and cost, schedule and risk reporting and management.

Since SysML removes many of the SW-centric constructs, only re-using seven of UML 2's fourteen diagrams, the overall language is smaller than UML. With fewer overall constructs and fewer diagrams, SysML aims to be easier to learn and apply compared with UML. With fewer restrictions than UML, the semantics are also aimed to be more expressive and flexible. Also important, these language constructs have been deliberately aligned with IEEE-1471-2000 recommended practice for architectural descriptions of SW-intensive systems and ISO/IEC 42010-2010 Systems and Software Engineering – Architecture Description to promote semantic consistency.

SysML has overcome some significant limitations in UML providing important additional functionality supporting SE notions of process. Two new diagram types have been added, a requirements diagram and a parametrics diagram. Expanded ability for allocation tables allows requirements allocation, functional allocation, and structural allocation to be derived dynamically from SysML allocation relationships. The flexible tabular formation of the allocation tables facilitates automated traceability tracking, and hence automated verification and validation tracking, and gap analysis[99].

The requirements diagram supports the broader notion of requirements engineering, a sub-discipline with SE. In particular, it aims to provide for efficient capture of functional, performance and interface requirements, whereas UML constrains requirement articulation within the context of the particular set of use cases, which typically only describe a limited number of high-level functions.

The parametric diagram is used to explicitly define performance and quantitative constraints to facilitate performance analysis and quantitative analysis, whereas UML language constructs do not readily support these notions.

[99] Detailed insight into the specific features of the SysML modelling language and its application to MBSE is provided in (Friedenthal et al. 2008).

Another significant change from UML is the introduction of the “block” as an element, to be used in lieu of an “object” or “class”. The “block” is conceptually a “black box” representation of a system piece-part. The “block” concept better represents the expanded SE notions of HW, SW, data, processes, personnel and facilities. Blocks can be components, subsystems or systems, thus allowing a system to be portrayed as an ordered composite of building blocks or “black boxes”, supporting the SE notion of system hierarchy and recursive decomposition.

Similarly, the concept of a “flow port” has been introduced in SysML to describe what can go in or out of a block at its interfaces, whether it be data, energy, or physical matter, thus providing the ability to explicitly portray interfaces with a commensurate scale of abstraction.

Appendix C: UML and SysML Common Terms and Diagrams

C.1. Object-Oriented Problem Solving – Overview

The construction of a model is at the heart of object-oriented problem solving. The model serves to abstract the essential details of the underlying problem from its usually very complex real world environment. Thus in modelling terms, the model is the abstraction of the problem, and the domain is the actual world from which the problem comes.

UML enables all stakeholders, from business analysts to programmers, to communicate about a SW design using a common vocabulary; the more complex the system, the greater the importance of communication among everyone involved in creating and deploying the resultant system implementation.

UML and SysML diagrams can be used in two ways:

- To specify models from which to construct an executable system (forward engineering),
- To reconstruct models from parts of an executable system (reverse engineering).

Two different approaches can be adopted to model a system at different levels of abstraction:

- Presenting diagrams with different level of detail against the same model, or
- By creating models at different levels of abstraction with diagrams that trace from one model to another.

C.2. Important Terms and Concepts

An introduction to object-oriented modelling concepts and UML basic terms is provided in a series of IBM developerWorks e-zines, collated and published as a single book by IBM Press (Quatrani & Palistrat 2006). To use UML as a modelling language, it is essential to have a deep understanding of the terms and concepts as can be found in texts and tool vendor tutorials, for example, in (Booch et al. 1999), (Bittner & Spence 2002), (Weilkiens 2006), (Booch et al. 2007), (Sparx 2007a), (Sparx 2007b), (Friedenthal 2008).

In the context of UML, a "system" is a collection of subsystems organised to accomplish a purpose – described by a set of models, possibly from different viewpoints. A "subsystem" is a grouping of elements, of which some constitute the specification of behaviour offered by the other contained elements.

A UML "model" is a semantically closed abstraction of a system such that it represents a complete and self-consistent simplification of reality, created in order to better understand the system. The "system" is the "thing" which is the implemented outcome[100]; which can be viewed from different perspectives by different models, with those views visually

[100] Also sometimes called "target system".

presented in the form of diagrams. The diagrams are graphical projections into the elements that make up a system.

An object-oriented model composed in UML consists of “objects” that interact by sending each other “messages”. Objects have things they know (“attributes”), and things they do (“behaviours” or “operations”), where the value of an object’s attributes determines its “state”. A “class” wraps attributes (data) and behaviours (methods or functions) into a single entity. Objects are “instances” of classes. A “role” is a behaviour of an entity participating in a particular context.

A “Class Diagram” gives an overview of a system by showing its classes, interfaces, and collaborations, and the relationships between them; they show what interacts, but not what happens when they do interact.

The “package” in UML is a general purpose mechanism for organising logically related modelling elements into groups. Importantly, it provides the ability to scale up and provide a higher degree of abstraction whilst maintaining the integrity of the relationships between the elements within the package. It is particularly useful for visualising, specifying, constructing and documenting large systems involves manipulating potentially large numbers of classes, interfaces, components, nodes, and diagrams (Booch et al. 1999).

An “interface” is a named collection of operations that are used to specify a service of a class or component. A class may realise many interfaces, so each instance of that class must therefore support all those interfaces. However, an interface may only present one or more of its interfaces as being relevant in a particular context. Each interface represents a role that the object plays, where the role names a behaviour of an entity participating in a particular context.

A “component” is a module of code, where the component diagram shows the physical instantiation of the class diagrams. The physical HW is made up of “nodes”, where each component belongs on a node. Deployment diagrams show the physical configurations of HW and SW.

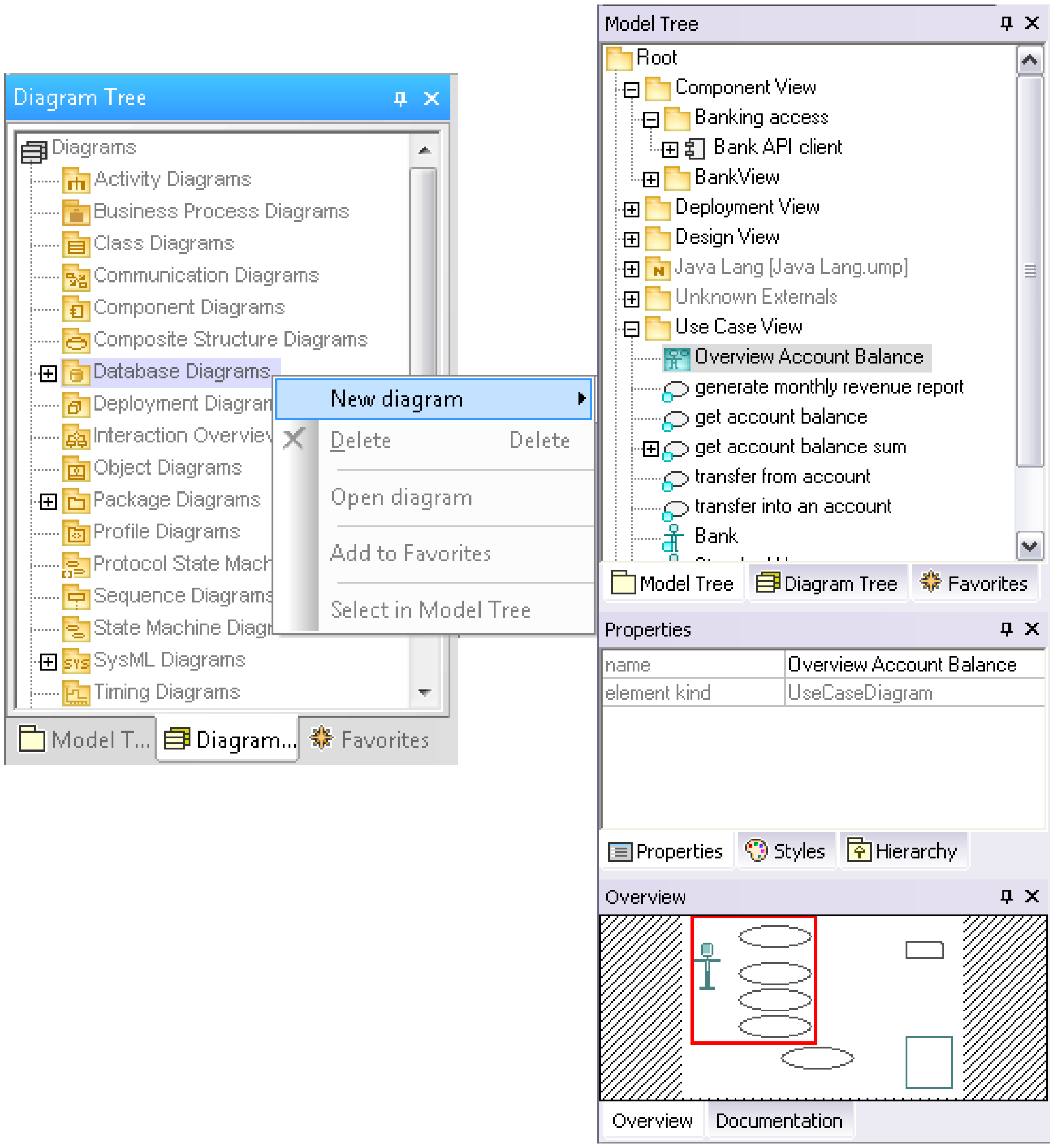
A “use case” is a summary of “scenarios” for a single task or goal, where a “scenario” is a description of what happens if there is an interaction between the system and the external environment. An “actor” is “who” or “what” initiates the events involved in that task, i.e. they are roles that people or objects play. The connection between an actor and a use case is known as a “communication association”.

The “Use Case Diagram” is a collection of actors, use cases and their communications. It describes what a system does from the standpoint of an external observer; the emphasis on “what” the system does rather than “how”. Use case diagrams are prominent in UML modelling where they can be useful to determine new requirements during analysis and design activity, as well as suggesting suitable test cases for the scenarios for V&V activity.

The “stereotype” is also an important concept in UML modelling where a stereotype defines how a model element may be extended, and enables the use of platform or domain specific terminology or notation in lieu of, or in addition to, the ones used for an extended meta-class.

C.3. Tool User Interface

Examples of a tool user interface to construct and view a model of a banking system from different diagrammatic perspectives are shown in Figures C.1 and C.2. The model tree view of Figure C.2 provides an overview of the project, organised by the relationships between individual diagrams. The diagram tree sorts the contents of the model by diagram type to facilitate data analysis, depending on the nature of the enquiry. A dialog box to enter in text -based requirements and to edit properties is shown in Figure C.3.



name	Overview Account Balance
element kind	UseCaseDiagram

Figure C.1. Example of UML Tool User Interface showing the Model Tree and Diagram Tree Perspectives

[online] URL: http://www.altova.com/umodel/visual-modeling.html).

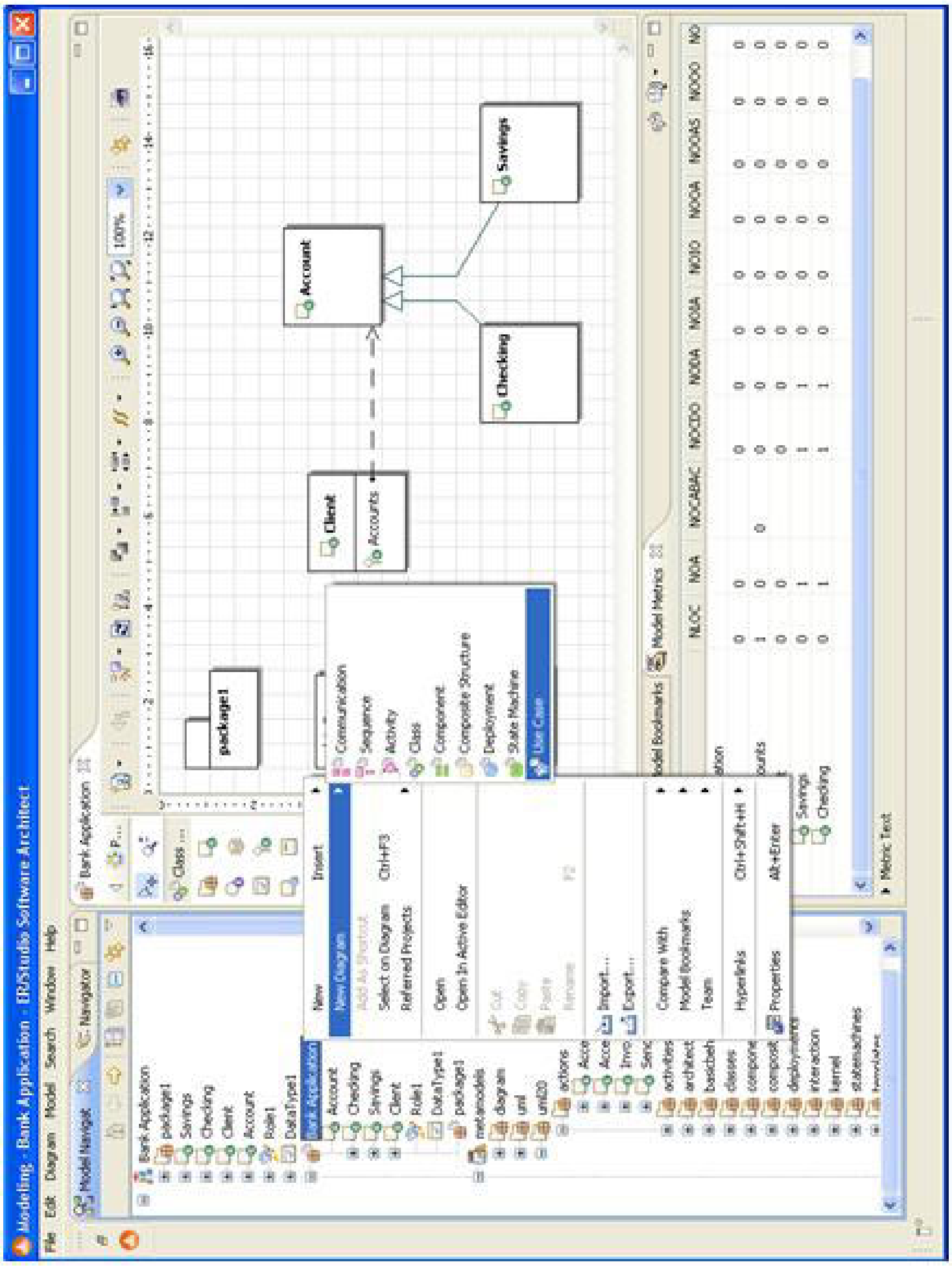
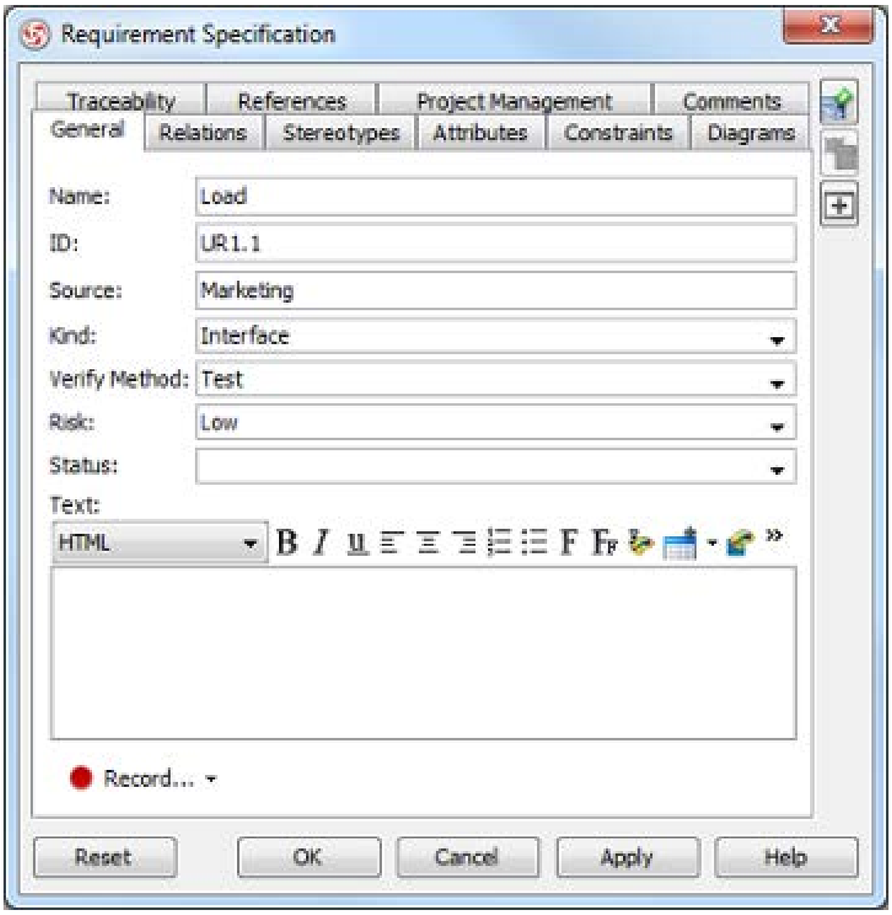


Figure C.2. Example Tool User Interface to the Totality of the UML Modelling Environment ([online] URL: http://www.embarcadero.com/products/er-studio)).



Name:	Load
ID:	UR1.1
Source:	Marketing
Kind:	Interface
Verify Method:	Test
Risk:	Low
Status:	
Text:	

Figure C.3. Example of SysML Tool User Interface to enter or edit Requirement Properties ([online] URL: http://www.visual-paradigm.com/support/documents/vpumluserguide/1281/158/6516_creatingrequ.html)

.

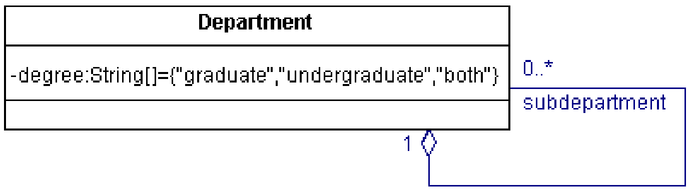
C.4. UML Diagrams

C.4.1 UML2 Object and Class and Diagrams

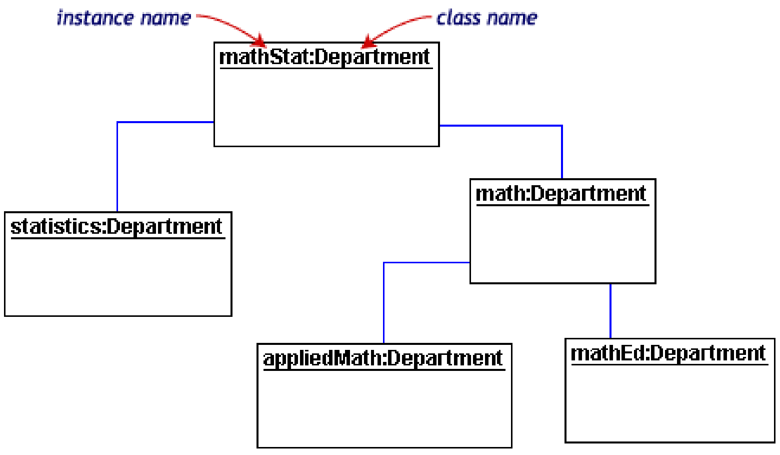
Examples of different types of UML2 modelling diagrams are provided in Figures C.4. to C15.

The Class Diagram provides an overview of the target system by describing the objects and classes inside the system, and the relationships between them. It can be used in different ways, for example, from modelling domain-specific data structures to the detailed design of the target system. The class model can be reused in the Interaction Diagram for modelling the detailed design of the dynamic behaviour.

A simple example of a Class Diagram is shown in Figure C.4 (a). with the Object Diagram showing an instantiation of the class **Department** in Figure C.4(b). The Object Diagram shows a snapshot of instances of things in Class Diagrams. Similar to Class Diagrams, Object Diagrams show the static design of the system, but from the real or prototypical view.



Class diagram – University Department



(b) Object Diagram – Maths Department

Figure C.4. Example of a UML2 Class Diagram and associated Object Diagram

([online] URL: http://edn.embarcadero.com).

C.4.2 UML2 Package Diagram

The Package Diagram of Figure C.5 shows the arrangement and organisation of model elements in larger scale projects, both in terms of the structure and the dependencies between lower-tier sub-systems or modules.

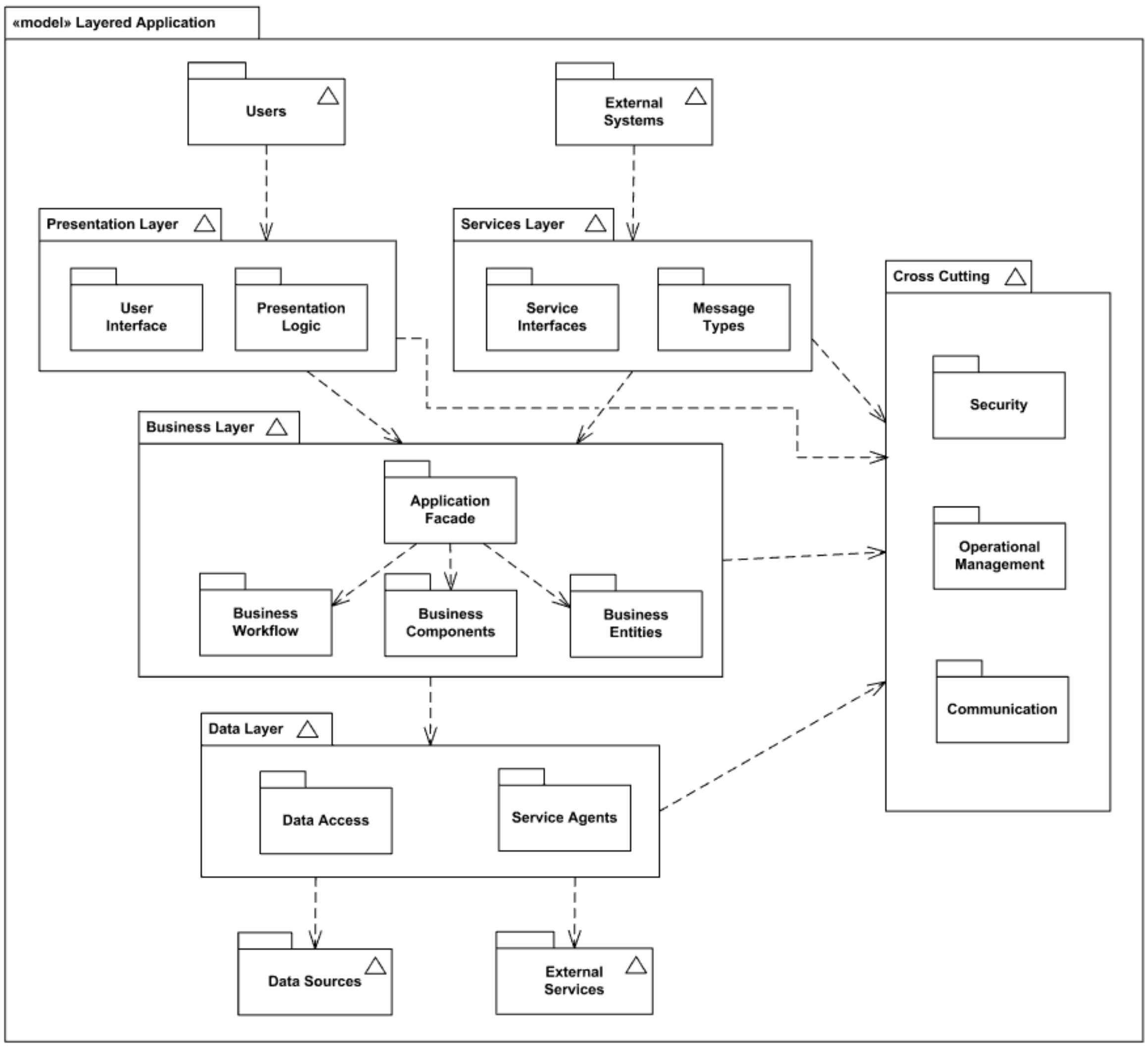


Figure C.5. Example of a UML2 Package Diagram

([online] URL: http://www.uml-diagrams.org/package-diagrams-examples.htm).

C.4.3 UML2 State Chart and Activity Diagram

Examples of a State Chart Diagram and an Activity Diagram associated with using an Automatic Teller Machine (ATM) are shown in Figures C.6 and C.7. A State Chart or State Machine Diagram can show the history of an entity, where the behaviour is not only dependent on its input, but it is also dependent on its previous state. The State Machine Diagram can also show the different states of an entity, and how the entity might respond to various events by changing from one state to another.

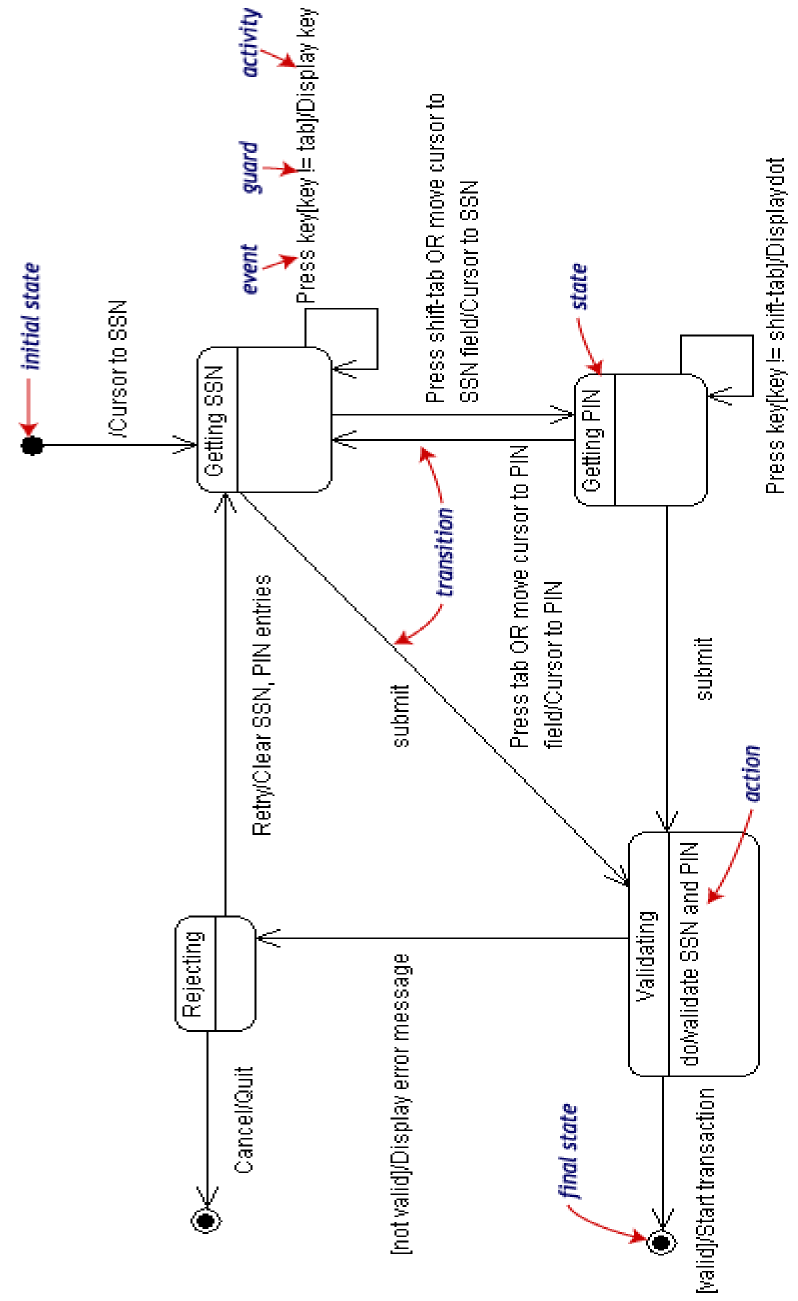

initial state
/Cursor to SSN
Getting SSN
event
guard
activity
Press key[key != tab]/Display key
Press shift-tab OR move cursor to SSN field/Cursor to SSN
state
Getting PIN
Press key[key != shift-tab]/Displaydot
transition
Press tab OR move cursor to PIN field/Cursor to PIN
submit
submit
Retry/Clear SSN, PIN entries
action
Validating
do/Validate SSN and PIN
Rejecting
Cancel/Quit
[not valid]/Display error message
final state
[valid]/Start transaction

Figure C.6. Example of a UML2 State Chart showing ATM Operation

([online] URL: http://edn.embarcadero.com).

The Activity Diagram of Figure C.7 is used to describe the flow of control of the target system, incorporating business rules and operations.

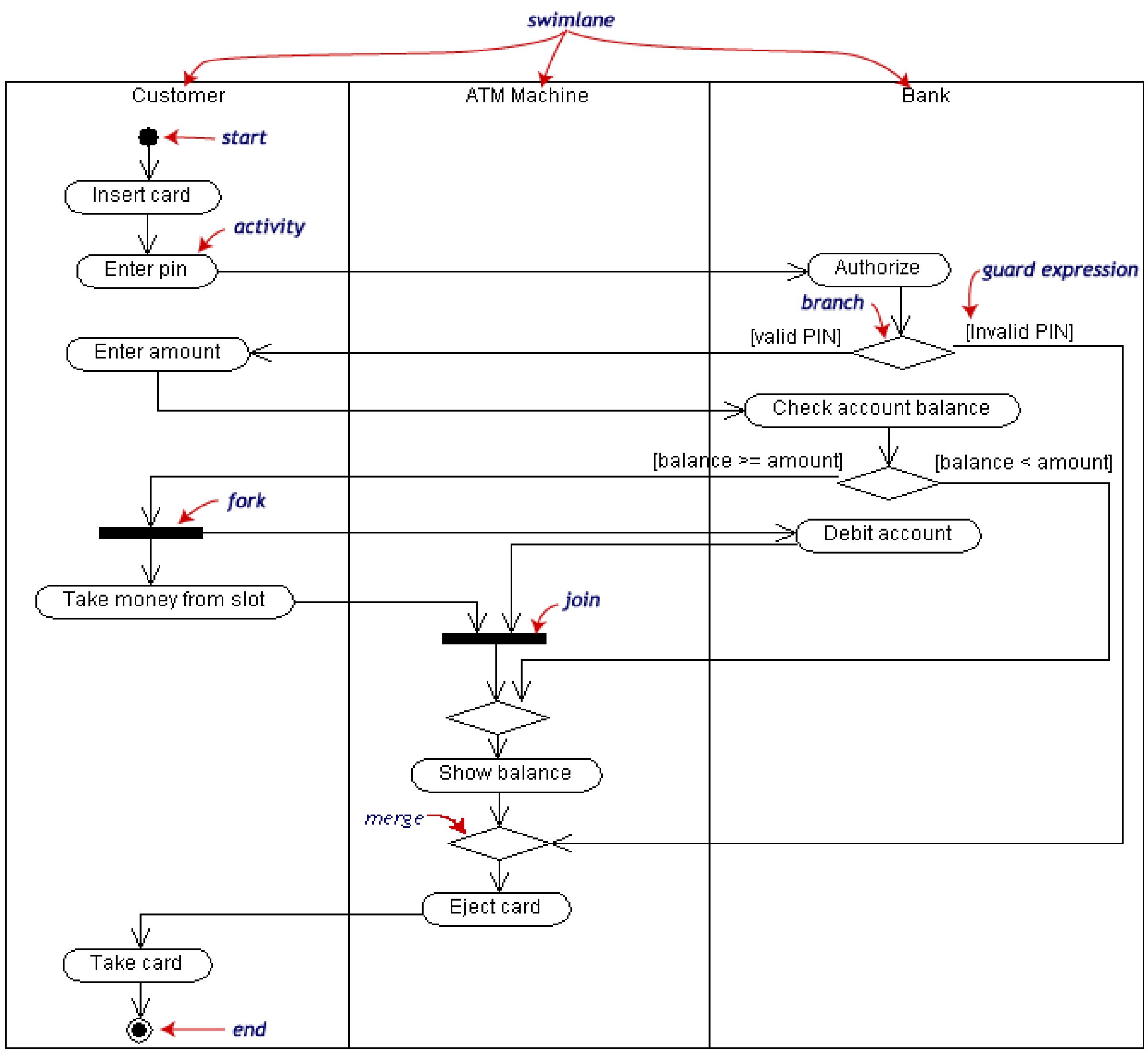


Figure C.7. Example of a UML2 Activity Diagram showing ATM Operation

([online] URL: http://edn.embarcadero.com).

C.4.4 UML2 Sequence Diagram

The Sequence Diagram, as shown in Figure C.8, models the collaboration of objects based on a time sequence. It shows how objects interact with others in a particular scenario of a use case.

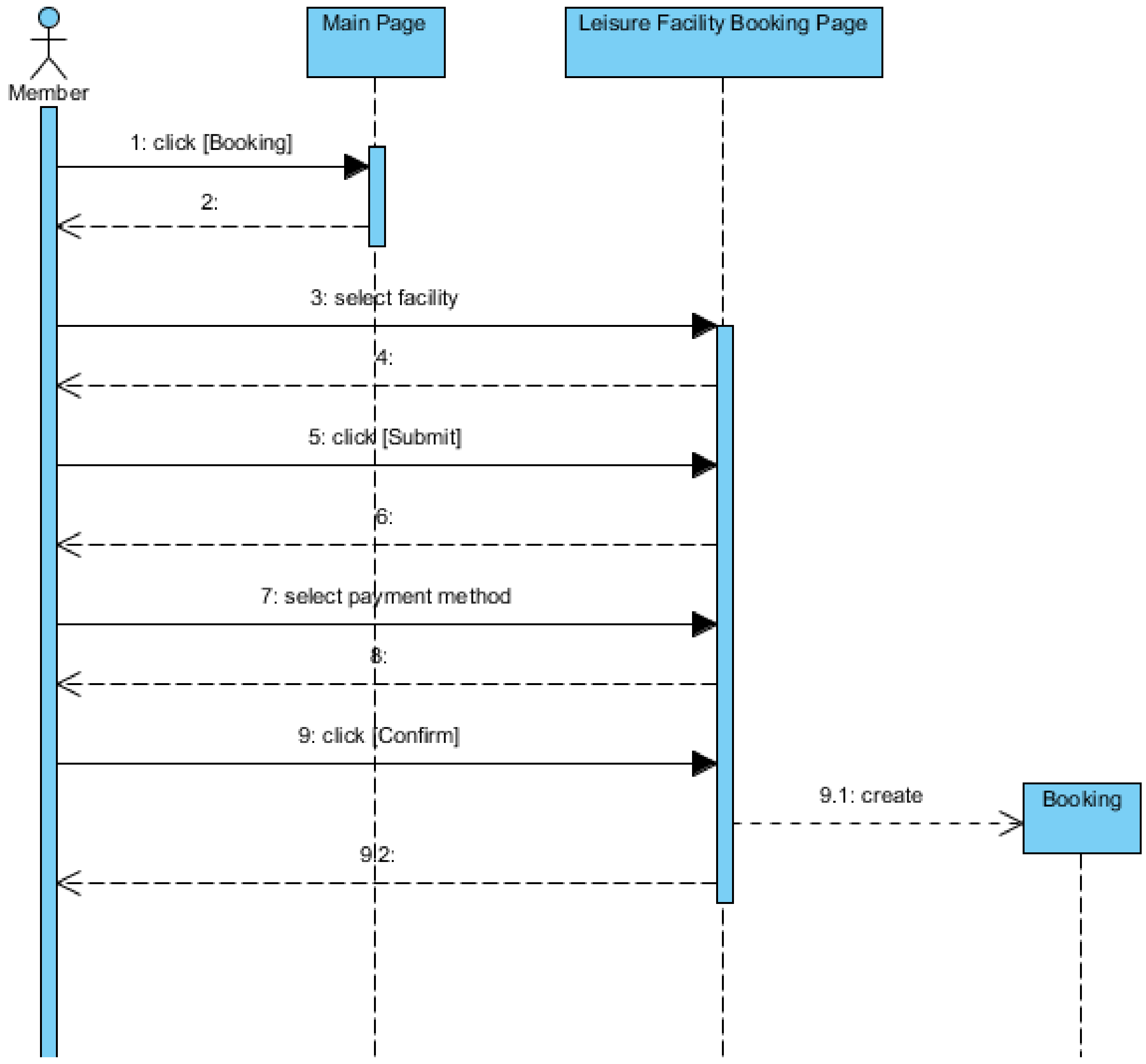


Figure C.8. Example of a UML2 Sequence Diagram showing part of a Leisure Complex Booking System

([online] URL: http://www.visual-paradigm.com/VPGallery/diagrams/Sequence.html).

C.4.5 UML2 Communication Diagram

Similar to a Sequence Diagram, a Communication Diagram, as shown in Figure C.9 is also used to model the dynamic behaviour of the use case. However, it focusses more on showing the collaboration of objects rather than the time sequence.

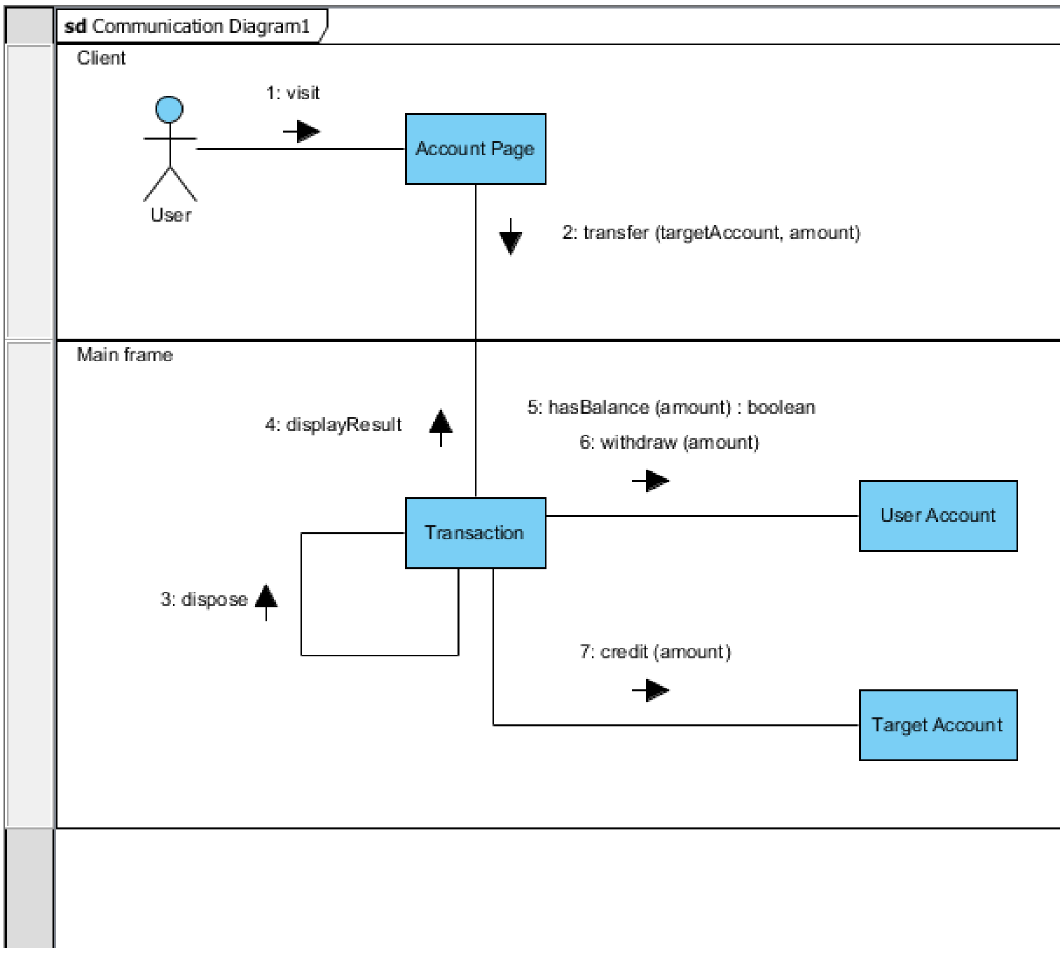


Figure C.9. Example of a UML2 Communication Diagram, part of the Leisure Centre Booking System

([online] URL: http://www.visual-paradigm.com/VPGallery/diagrams/Collaboration.html).

C.4.6 UML2 Timing Diagram

The Timing Diagram, as shown in Figure C.10, shows the behaviour of objects in a given period of time. The Timing Diagram is a special form of Sequence Diagram, but the axes are reversed so that time increased from left to right in the diagram, and lifelines are shown in separate compartments arranged vertically.

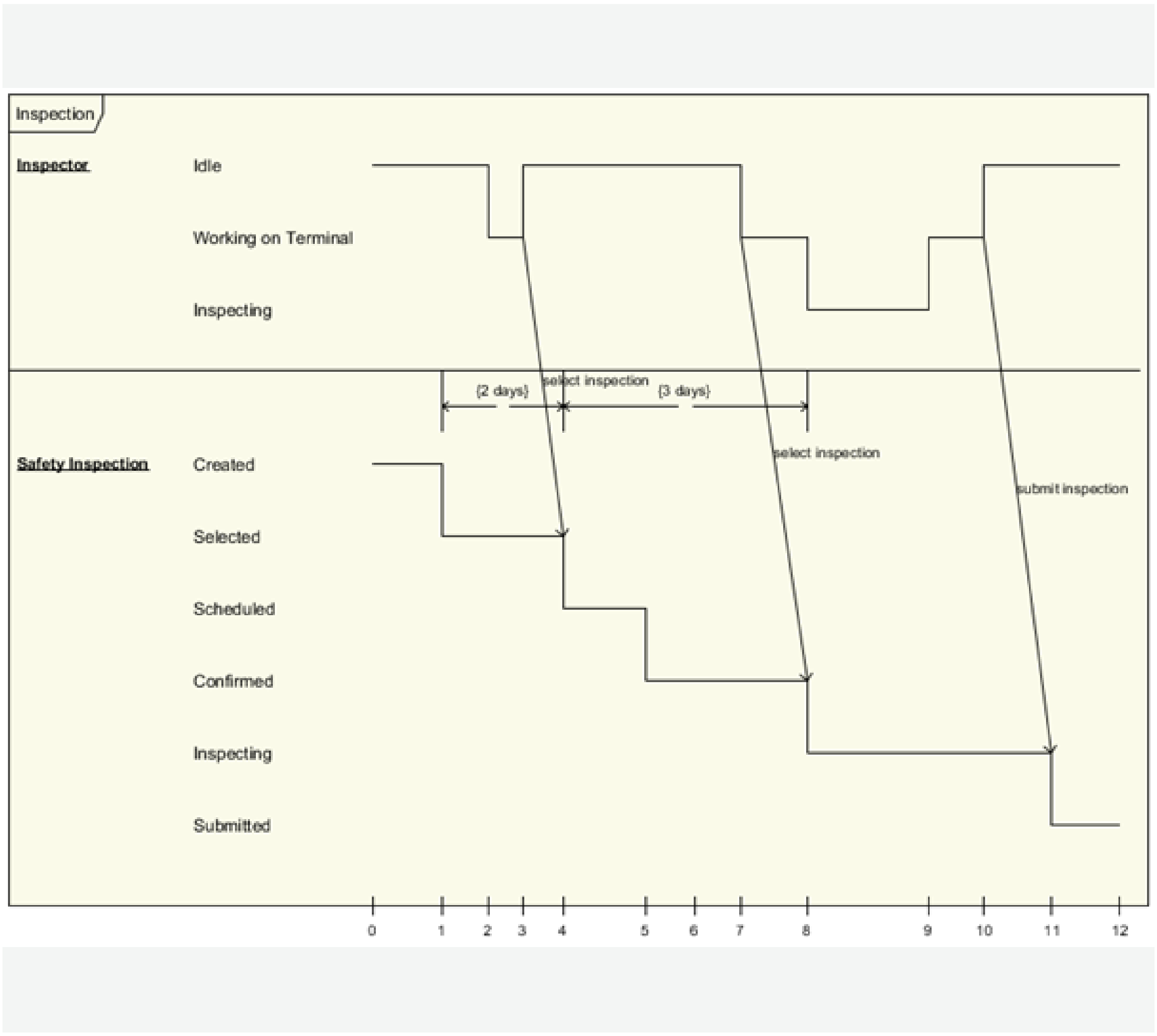


Figure C.10. Example of a UML2 Timing Diagram, part of a Safety Inspection System

([online] URL: http://www.visual-paradigm.com/VPGallery/diagrams/TimingDiagram.html).

C.4.7 UML2 Interaction Overview Diagram

The Interaction Overview Diagram, as shown in Figure C.11, provides an overview of the flow of controls of the interactions. It is a variant of the Activity Diagram where the nodes are the interactions or interaction occurrences. It describes interactions where messages and lifelines are hidden.

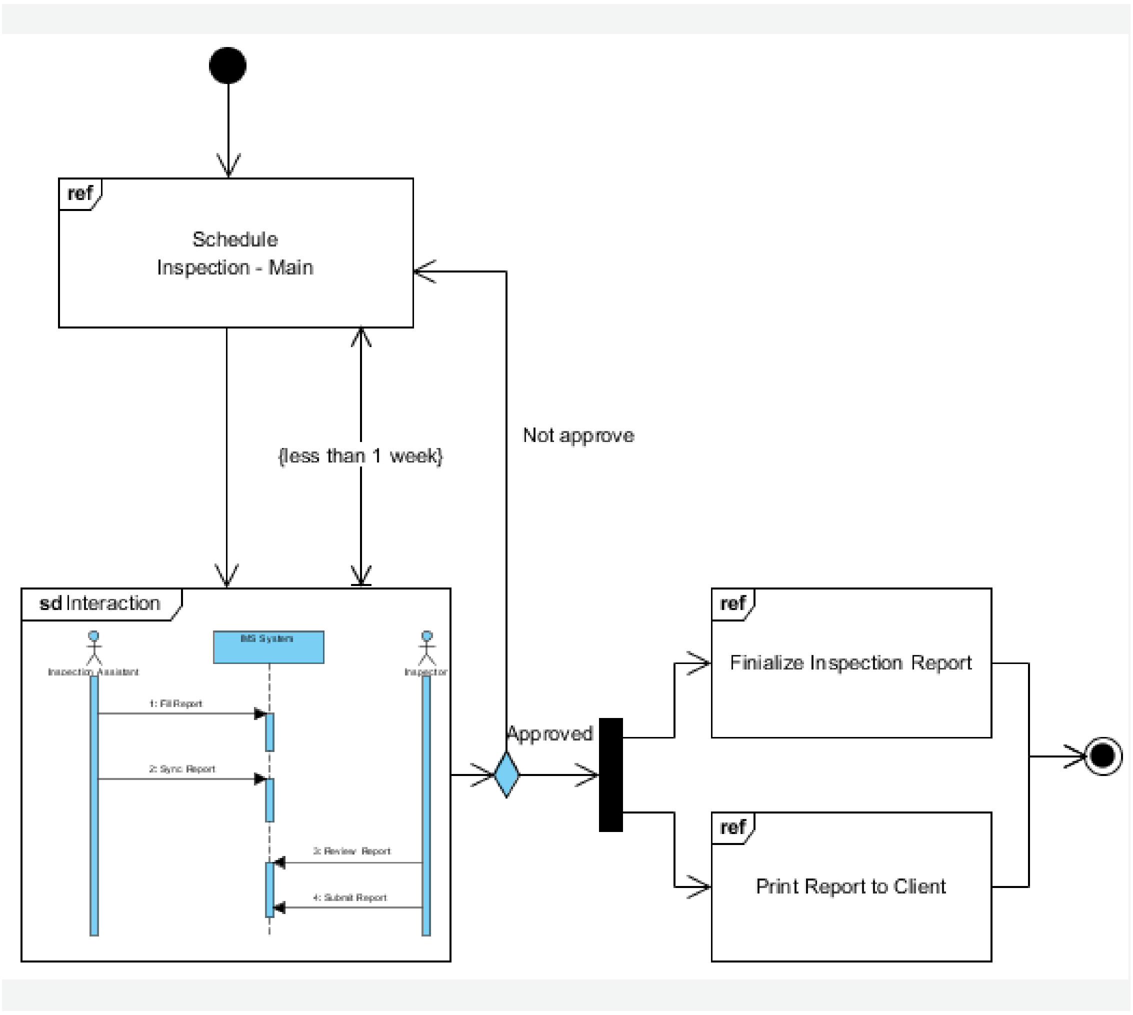


Figure C.11. Example of a UML2 Interaction Diagram, part of a Safety Inspection System ([online] URL: http://www.visual-paradigm.com/VPGallery/diagrams/InteractionOverviewDiagram.html).

C.4.8 UML2 Component Diagram

The Component Diagram, as shown in Figure C.12, assists to model the physical aspects of an object-oriented SW system. It includes the SW architectures of the SW components and the dependencies between them (both source code and run-time components).

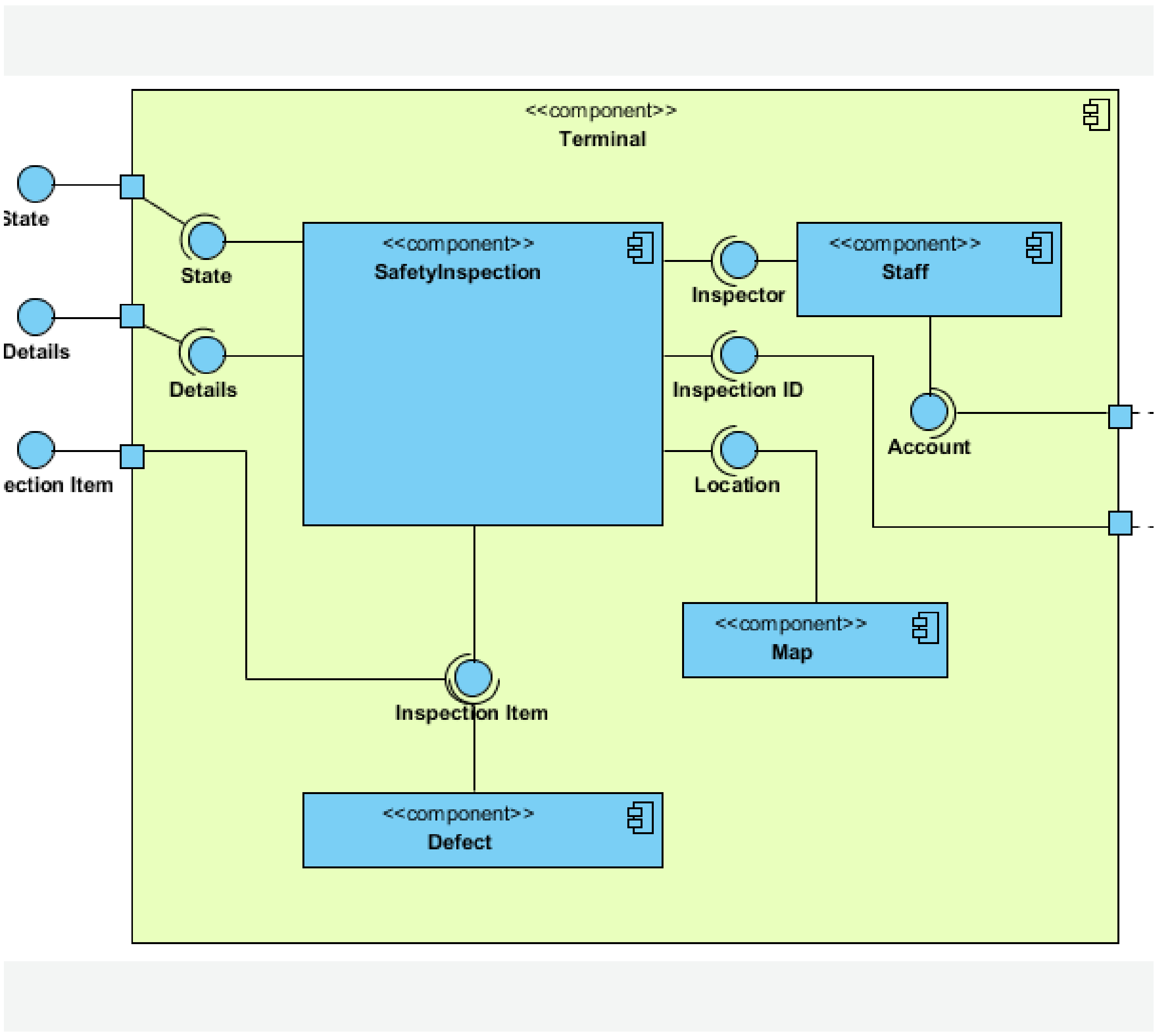


Figure C.12. Example of a UML2 Component Diagram, part of a Safety Inspection System ([online URL: http://www.visual-paradigm.com/VPGallery/diagrams/Component.html).

C.4.9 UML2 Deployment Diagram

The Deployment Diagram, as shown in Figure C.13, is also used to model the physical aspect of an object-oriented SW system, where it provides a static view of the run-time configuration, and provides visualisation of the distribution or mapping of the SW components onto the respective HW configurations that host the SW.

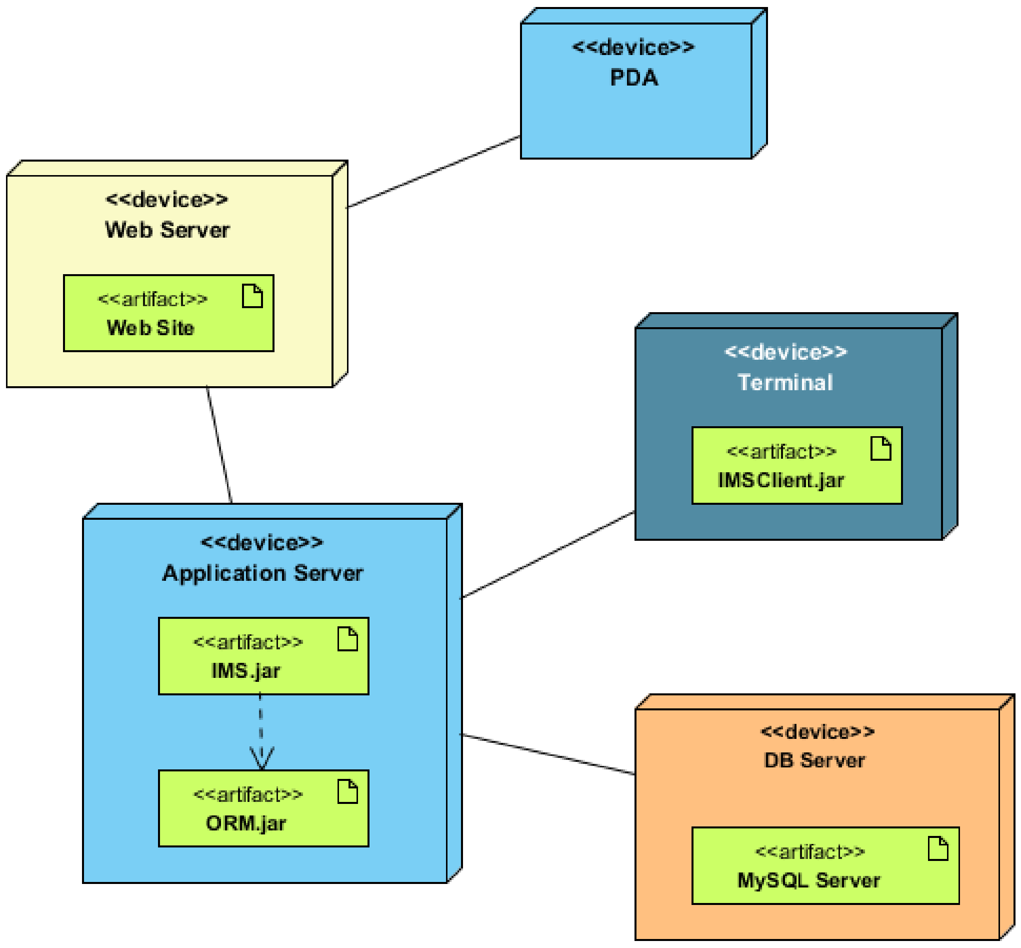


Figure C.13. Example of a UML2 Deployment Diagram

([online] URL: http://www.visual-paradigm.com/VPGallery/index.html)

C.4.10 UML2 Composite Structure Diagram

The Composite Structure Diagram, as shown in Figure C.14, shows the internal structure, including parts and connectors of a structured classifier or collaboration.

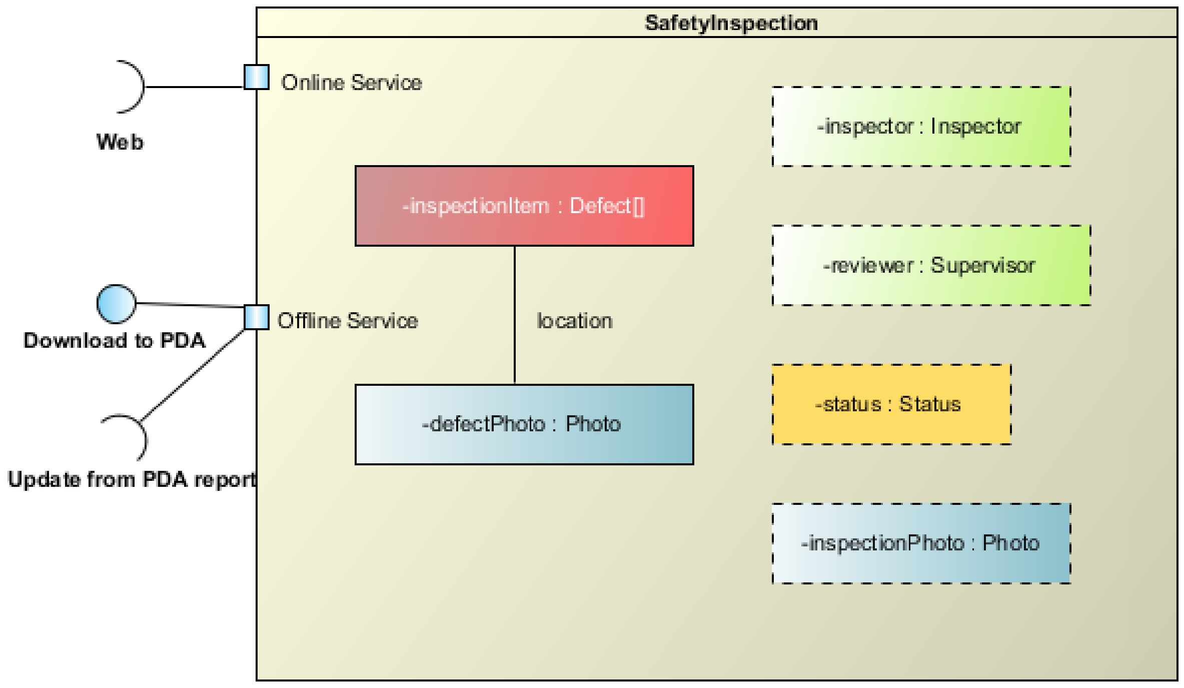


Figure C.14. Example of a UML2 Composite Diagram of a Safety Inspection System

([online] URL: http://www.visual-paradigm.com/VPGallery/diagrams/CompositeStructureDiagram.html).

C.4.11 UML2 Use Case Diagrams

Use Case Diagrams are created for describing the behaviour of the target system from an external point of view as shown in Figure C.15.

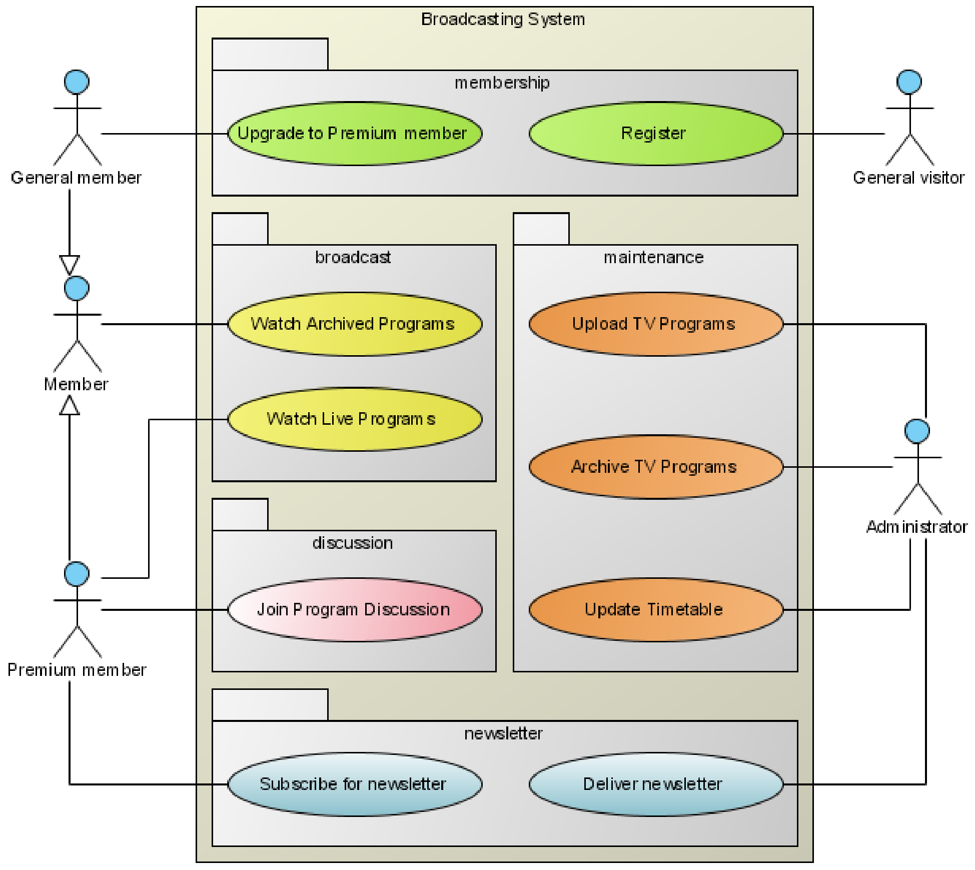


Figure C.15. Example of a UML2 Use Case Diagram

([online] URL: http://www.visual-paradigm.com/VPGallery/diagrams/UseCase.html).

C.5. SysML Diagrams

C.5.1 SysML System Block Definition Diagram

Shortcomings in the use of the UML modelling languages, and changes between the UML 2 and SysML modelling languages to overcome these shortfalls are described in (Hause 2006). Similar to modelling in UML, a deep understanding of the concepts and terms in SysML is prerequisite to modelling using SysML or the UPDM language profile, as can be found in numerous text books and tool vendor tutorials including (Weilkiens 2006), (Friedenthal et al. 2006), (Friedenthal et al. 2008), (Carson et al. 2009) and (IBM UPDM 2012).

Additional diagrams offered by the SysML language over and above UML include Requirements Diagrams, System Block Definition Diagrams and Parametric Diagrams, as shown in the examples of Figures C.16 to C.21.

SysML uses the concepts of a block to specify hierarchies and interconnections within a system design as shown in Figure C.16. It can also describe relationships between blocks such as composition, association and specialisation.

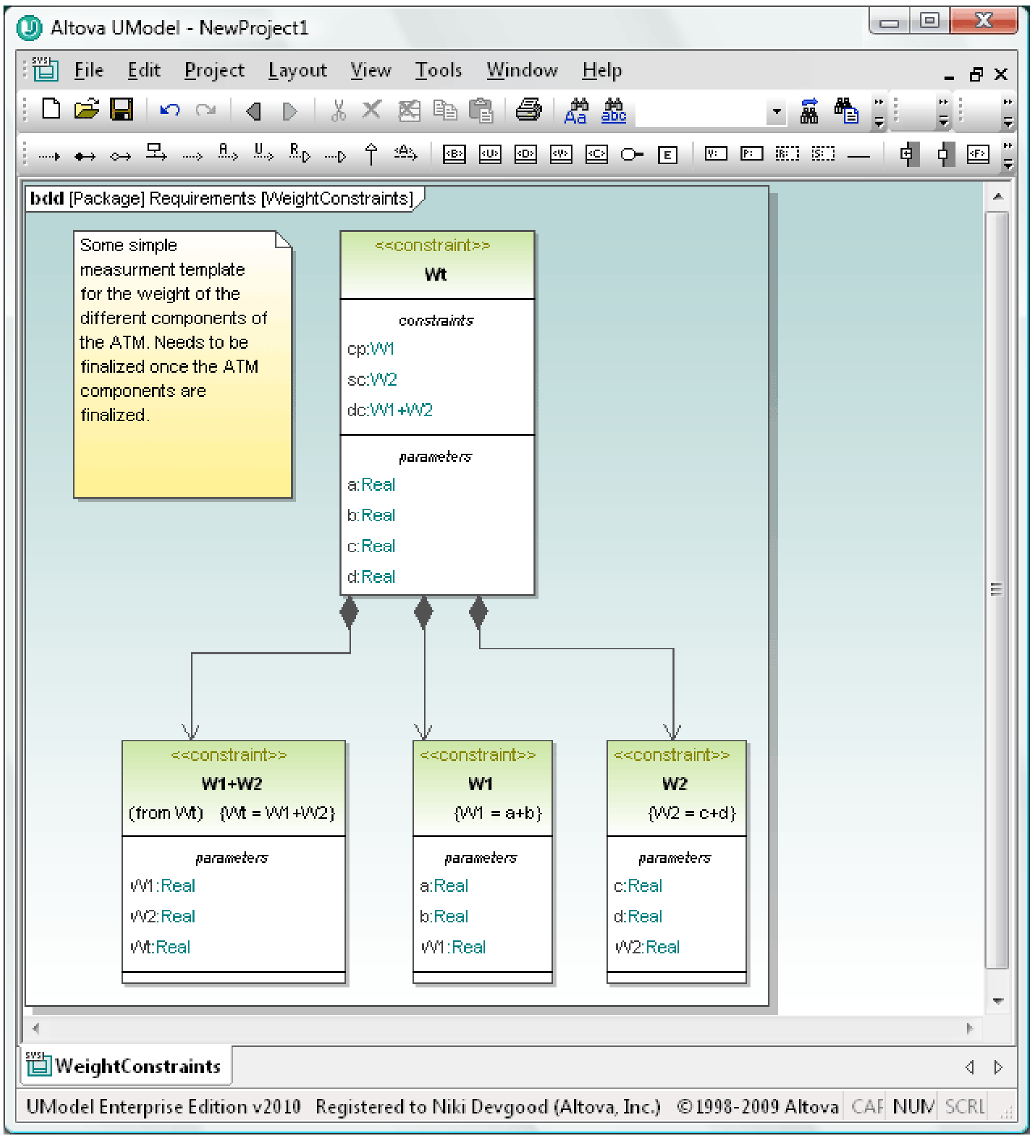
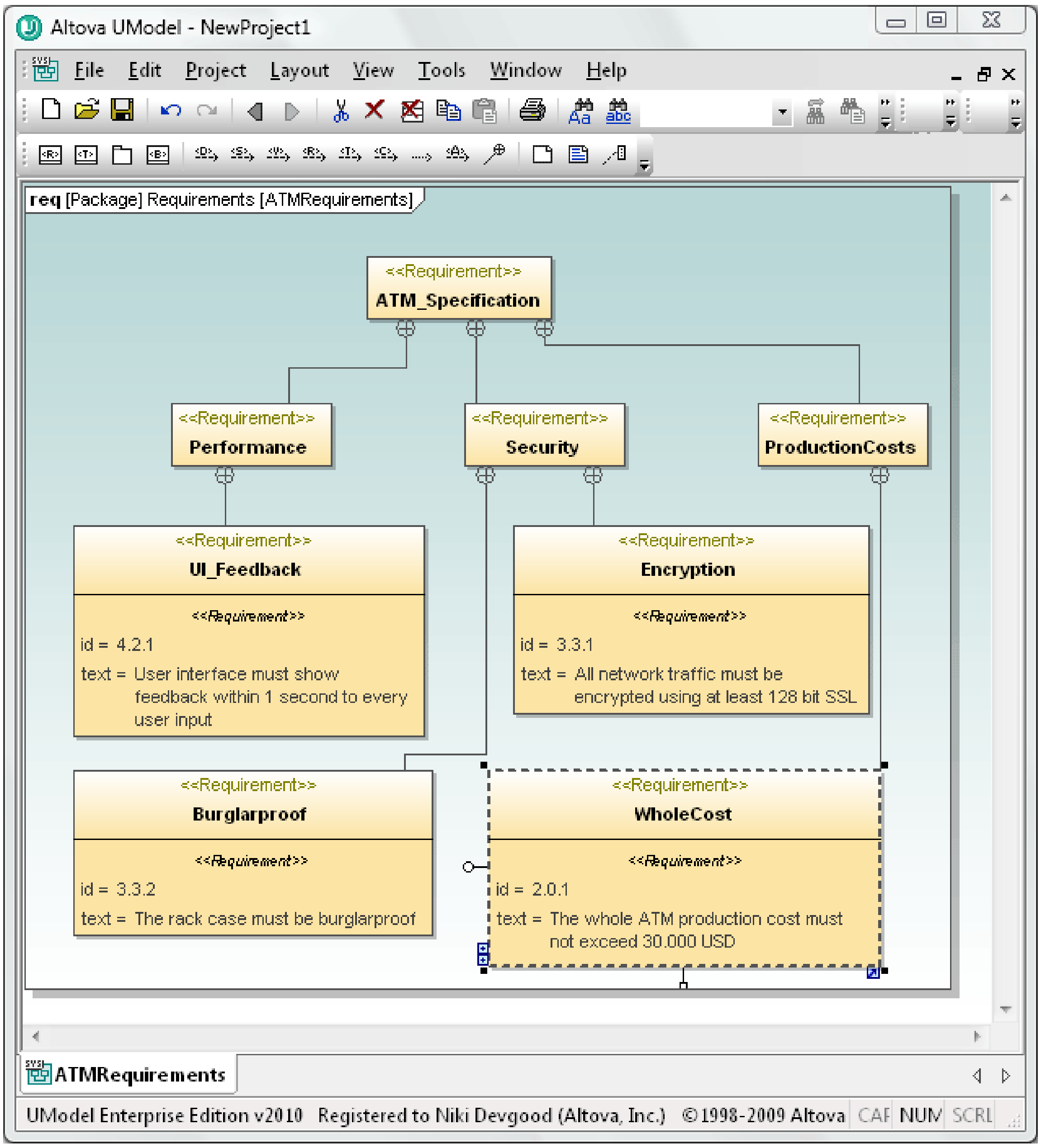

Some simple measurment template for the weight of the different components of the ATM. Needs to be finalized once the ATM components are finalized.

Figure C.16. Example of a SysML System Block Definition Diagram ([online] URL: http://www.altova.com/umodel/sysml.html#BlockDef).

C.5.2 SysML Requirements Diagram

The Requirements Diagram, as shown in Figure C.17, describes the functional, performance and interface requirements, including physical properties and constraints, typically not captured in the UML Use Case Diagrams.


Altova UModel - NewProject1
File Edit Project Layout View Tools Window Help
req [Package] Requirements [ATMRequirements]
<<Requirement>> ATM_Specification
<<Requirement>> Performance
<<Requirement>> Security
<<Requirement>> ProductionCosts
<<Requirement>> UI_Feedback
<<Requirement>> Encryption
<<Requirement>> Burglarproof
<<Requirement>> WholeCost
ATMRequirements

UI_Feedback: id = 4.2.1; text = User interface must show feedback within 1 second to every user input

Encryption: id = 3.3.1; text = All network traffic must be encrypted using at least 128 bit SSL

Burglarproof: id = 3.3.2; text = The rack case must be burglarproof

WholeCost: id = 2.0.1; text = The whole ATM production cost must not exceed 30.000 USD

UModel Enterprise Edition v2010 Registered to Niki Devgood (Altova, Inc.) ©1998-2009 Altova CAP NUM SCRL

Figure C.17. Example of a SysML Requirements Diagram

([online] URL: http://www.altova.com/umodel/sysml.html#Requirements).

C.5.3 SysML Package Diagram

The SysML Package Diagram as shown in Figure C.18 is similar to the UML Package Diagram in that it shows grouping of elements within a hierarchical structure.

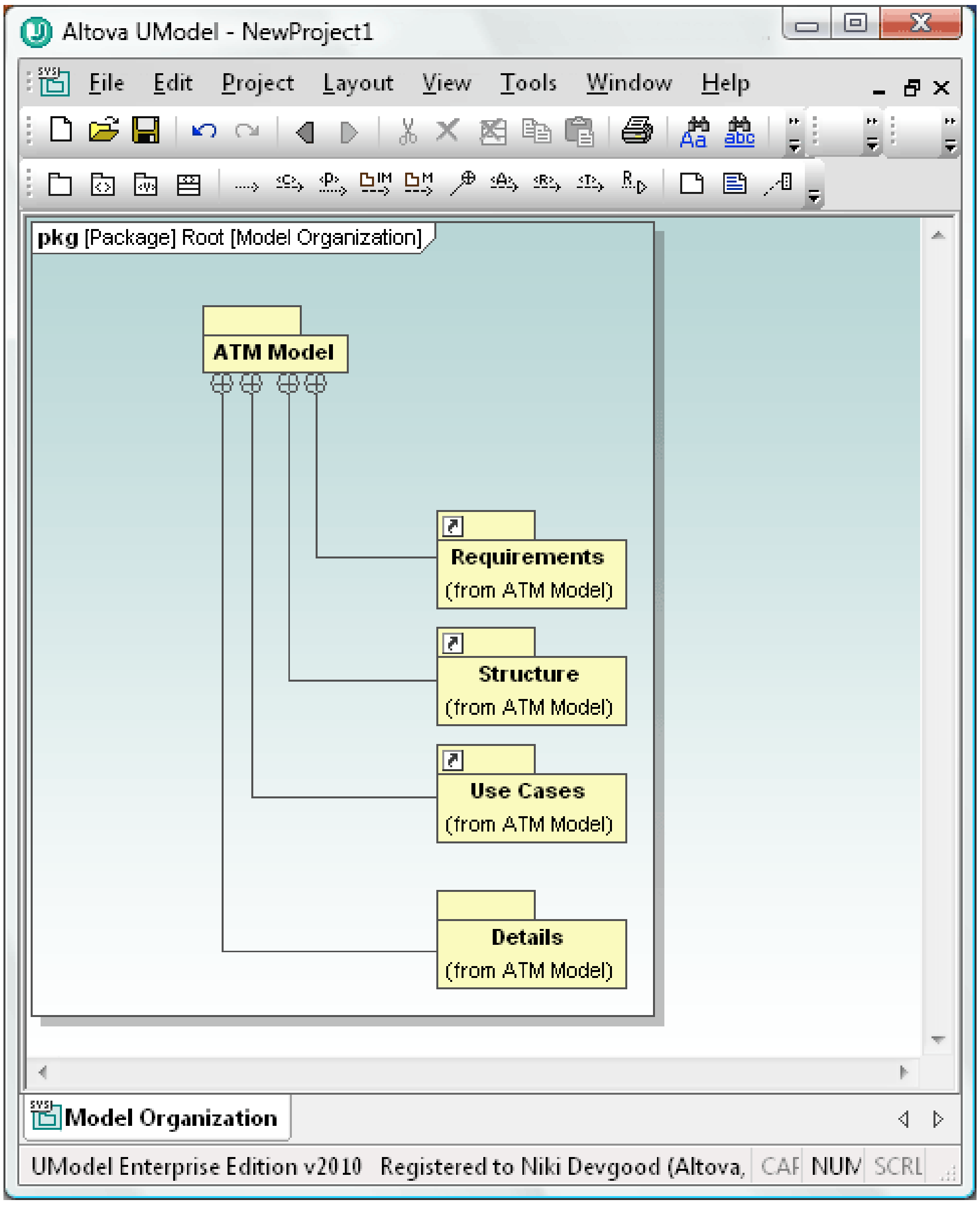


Figure C.18. Example of a SysML Package Diagram

([online] URL: http://www.altova.com/umodel/sysml.html#Package).

C.5.4 SysML Internal Block Diagram

The SysML Internal Block Diagram, as shown in Figure C.19, shows the internal structure of a block, together with its properties and connectors.

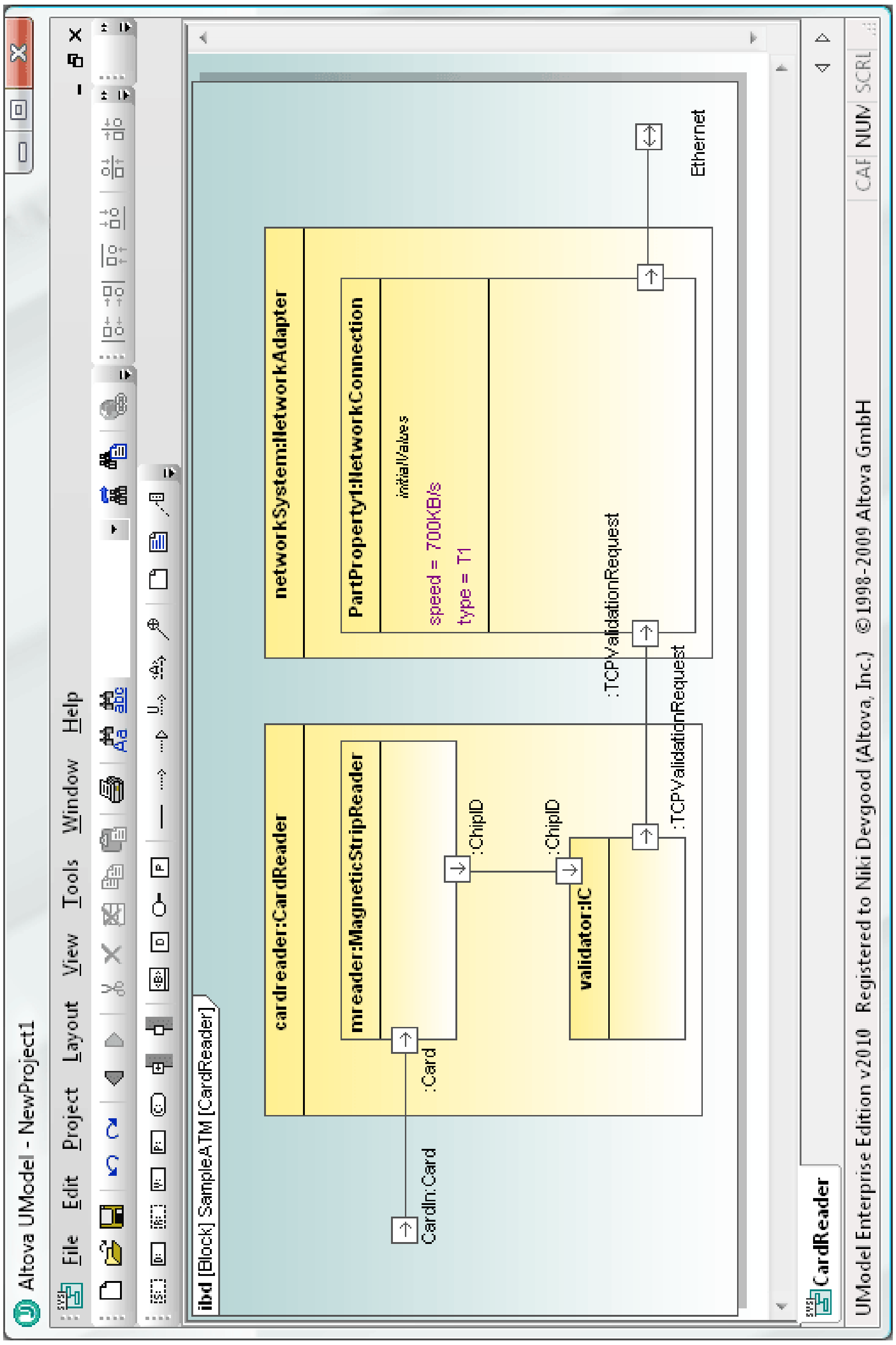

Altova UModel - NewProject1
File
Edit
Project
Layout
View
Tools
Window
Help
ibd [Block] SampleATM [CardReader]
cardreader:CardReader
mreader:MagneticStripReader
validator:IC
:Card
CardIn:Card
:ChipID
:ChipID
:TCPValidationRequest
:TCPValidationRequest
networkSystem:NetworkAdapter
PartProperty1:NetworkConnection
initialValues
speed = 700KB/s
type = T1
Ethernet
CardReader
UModel Enterprise Edition v2010 Registered to Niki Devgood (Altova, Inc.) ©1998-2009 Altova GmbH
CAP NUM SCRL

Figure C.19. Example of a SysML Internal Block Diagram ([online] URL: http://www.altova.com/images/shots/UML_SysMLInternalBlock6.gif).

C.5.5 SysML Parametric Diagram

The SysML Parametric Diagram, as shown in Figure C.20, provides a means to capture system constraints such as performance, reliability, and physical properties.

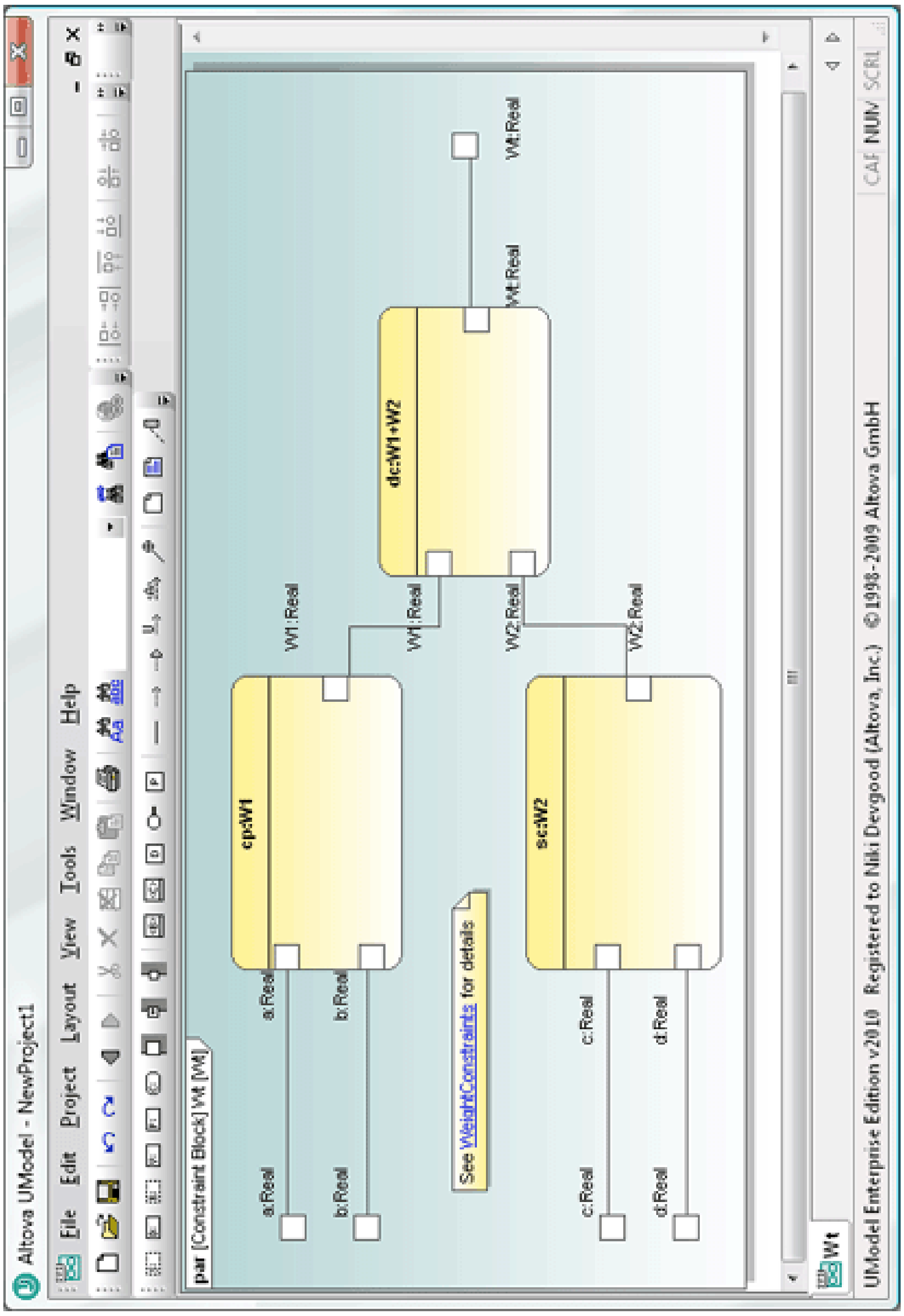

Altova UModel - NewProject1
File Edit Project Layout View Tools Window Help
par [Constraint Block] Wt [Wt]
a:Real
b:Real
cp:W1
a:Real
b:Real
W1:Real
W1:Real
W2:Real
dc:W1+W2
Wt:Real
Wt:Real
See WeightConstraints for details
c:Real
d:Real
se:W2
c:Real
d:Real
W2:Real
Wt
UModel Enterprise Edition v2010 Registered to Niki Devgood (Altova, Inc.) ©1998-2009 Altova GmbH
CAP NUM SCRL

Figure C.20. Example of a SysML Parametric Diagram

([online] URL: http://www.altova.com/umodel/sysml.html#Parametric).

C.5.6 SysML Use Case Diagrams

The SysML Use Case Diagram, as shown in Figure C.21, provide the same features as the UML Use Case Diagram, but adds an allocation relationship element.

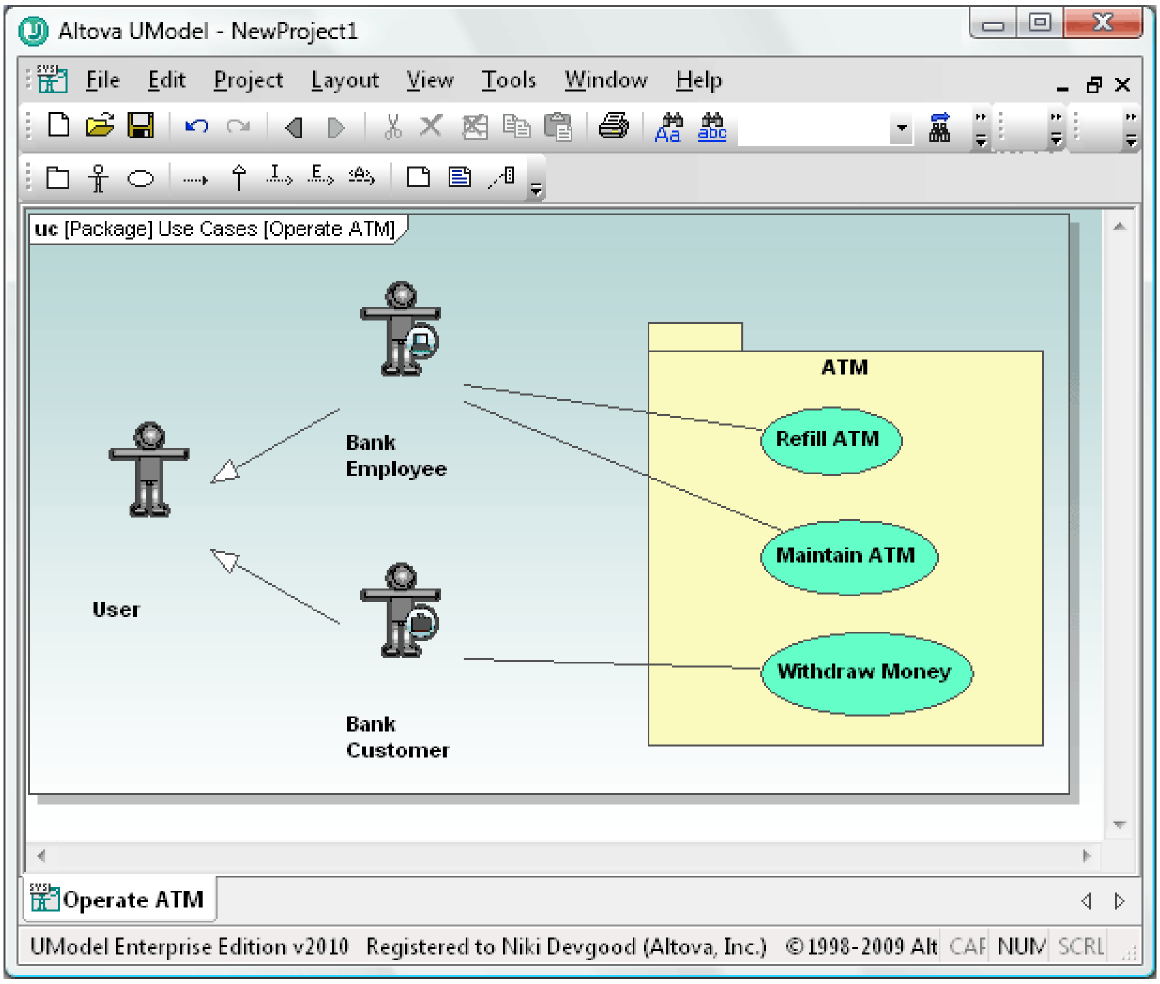

Altova UModel - NewProject1
File
Edit
Project
Layout
View
Tools
Window
Help
uc [Package] Use Cases [Operate ATM]
User
Bank Employee
Bank Customer
ATM
Refill ATM
Maintain ATM
Withdraw Money
Operate ATM
UModel Enterprise Edition v2010 Registered to Niki Devgood (Altova, Inc.) ©1998-2009 Alt
CAP
NUM
SCRL

Figure C.21. Example of a SysML Use Case Diagram

([online] URL: http://www.altova.com/umodel/sysml.html#UseCase).

Appendix D: Zachman Framework for Enterprise Architecture Overview

D.1. Introduction

The usefulness of a logical construct or architecture for defining and controlling the interfaces and integration of system components systematically across an entire enterprise was recognised as early as the 1980's, when Zachman released his seminal paper on a framework for information systems architecture, known as The Zachman Framework for Enterprise Architecture (ZF) (Hue 2008), (Zachman 1987)[101]. Notably, the ZF is tool, method and process agnostic, instead, describing a number of different lenses or filters to view different aspects of the enterprise from different stakeholder perspectives.

D.2. Zachman Framework Reference Model

The mainstay of the ZF is the ZF reference model as shown in Figure D.1, which prescribes a composite of different views of an enterprise, reflecting different stakeholder's perspectives (Zachman 2003), (Frankel et al. 2003), (VA 1992).

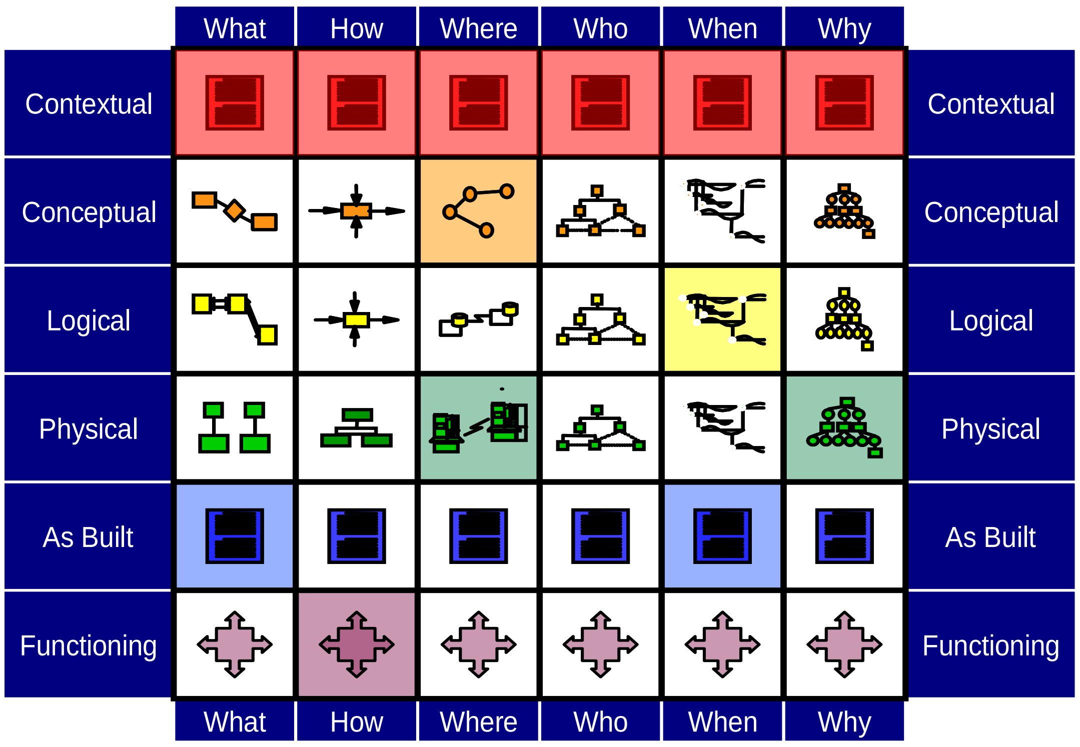


Figure D.1. Zachman Framework Reference Model (VA 1992)

[101] Zachman was employed by IBM at the time, and his seminal paper was published in the IBM Systems Journal (Zachman 1987).

The reference model provides an ontology for classifying the basic elements in the enterprise architecture, which collectively collates to become a form of knowledge representation about the enterprise. In the ZF, different analytical techniques can be employed to address the specific questions asked of the enterprise associated with each cell in the reference model. In this reference model, the columns have no order; each column has a simple basic model; the basic model of each column is unique (most are independent); each row represents a distinct view; and each cell is unique.

Different methods of enquiry, analytical techniques and tools can be used to address the enterprise-wide problem space, partitioned into specific focus areas as represented by each cell in the reference model. Different textual and graphical diagrammatic representations are used to record the output or outcomes of the activities as appropriate to the respective topics. However, the focus of the framework is based around the type of questions to be asked, rather than the means by which the insights and answers might be garnered.

In this case, the questions supported by the framework include who, what, when, where, why, and how in the context of different stakeholders views of the enterprise as follows:

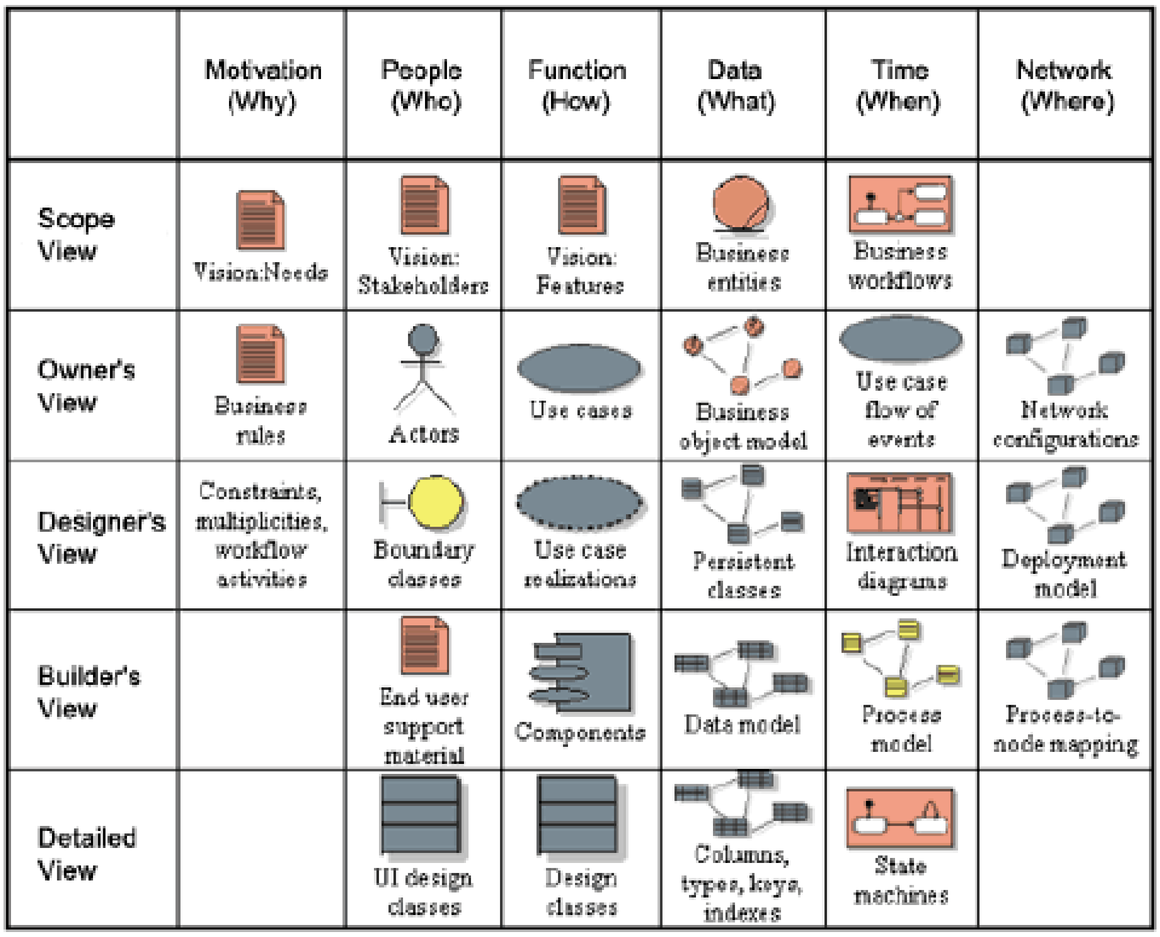
- Planner's View: Scope
    - Focus: external requirements and drivers.
    - Purpose: to ensure designated business goals, objectives and performance measures are aligned; identify and align high-level business functions; high-level data classes related to each function; stakeholders related to each function; cycles and events related to each function.
    - Utilises business functional modelling.
- Owner's View: Enterprise Model
    - Focus: Business function allocation and elimination of function overlap and ambiguity.
    - Purpose: to ensure designated policies, procedures and standards associated with business processes are aligned; identify and align the respective business processes; roles and responsibilities, and locations associated with each process; events for each process and sequencing of integration and process improvements.
    - Utilises business process models.
- Designer's View: System Model
    - Focus: requirements definition and project management.
    - Purpose: to ensure designated policies, standards and procedures associated with a business rule model are aligned; provide a logical representation of information systems and their relationships; logical data models of data and data relationships underlying designated information; logical representation of access privileges constrained by roles and responsibilities; logical representation of the distributed system architecture for designated locations; and logical events and their triggered responses are constrained by business events and their responses.
    - Utilises logical models.

- Builder's View: Technology Model
    - Focus: Solution definition, development, and management.
    - Purpose: to ensure designated business rules are constrained by information system standards; provide specifications of applications that operate on particular technology platforms; database management systems type requirements which are constrained by logical data models; specification of the network.
    - Utilises physical models.
- Integrator's View: As Built
    - Focus: as built configuration management during system deployment.
    - Purpose: to ensure designated business rules are constrained by specific technology standards; SW applications coded to operate on specific technology platforms; data definitions constrained by physical data models; access privileges coded to control access to specific platforms and technologies; network devices configured to conform to node specifications; timing definitions coded to sequence activities on specific platforms and technologies.
    - Scrutinises real-world implementation.
- User's View: Functioning Enterprise
    - Focus: functioning enterprise, operations management, and evaluation.
    - Purpose: to ensure operating characteristics of specific technologies are constrained by standards; verify computer instructions are functioning correctly; verify data values are stored correctly in databases; verify designated personnel and key stakeholders are working well within their roles and responsibilities; verify the network is sending and receiving messages appropriately; and verify timing and sequencing of activities is correct.
    - Scrutinises real-world implementation (VA 1992), (Sowa & Zachman 1992).

A methodology for the practical implementation of the ZF to analyse the business within an enterprise using a commercial tool suite was developed by Rational Software in 2001, based on the capabilities of the Rational tool suite. The mapping of enterprise formalisms to ZF cells is shown in Figure D.2, with the associated mapping of RUP formalisms aligned to ZF cells is shown in Figure D.3 (de Villiers 2001). An example of UML tool support provided for the ZF is provided in Figure D.4.

	Data (What)	**Function (How)**	**Network (Where)**	**People (Who)**	**Time (When)**	**Motivation (Why)**
Scope View	List of things important to the enterprise.	List of processes the enterprise performs.	List of locations where the enterprise operates.	List of organisational units.	List of business events/cycles.	List of business objectives.
Owner's View	Entity relationship diagram.	Business process model (physical data flow diagram).	Logistics network (nodes and links).	Organisational chart with roles, skill sets, security issues.	Business master schedule.	Business rules.
Designer's View	Data model (converged entities, fully normalised).	Essential data flow diagram, application architecture.	Distributed system architecture.	Human interface architecture (roles, data, access).	Dependency diagram, entire life history (process structures).	Business rule model.
Builder's View	Data architecture (tables and columns) map to legacy data.	System design: structure chart, pseudo code.	System architecture (hardware, software types).	User interface (how the system will behave), security design.	"Control flow" diagram (control structures).	Business rule design.
Detailed View	Data design (de-normalised) physical storage design.	Detailed program design.	Network architecture.	Screens, security architecture (who can see what?).	Timing definitions.	Rule specification in program logic.
Operational View	Converted data.	Executable programs.	Communication facilities.	Trained people.	Business events.	Enforced rules.

Figure D.2. Mapping of Enterprise Formalisms to the Zachman Framework (de Villiers 2001).



	Motivation (Why)	People (Who)	Function (How)	Data (What)	Time (When)	Network (Where)
Scope View	Vision:Needs	Vision: Stakeholders	Vision: Features	Business entities	Business workflows	
Owner's View	Business rules	Actors	Use cases	Business object model	Use case flow of events	Network configurations
Designer's View	Constraints, multiplicities, workflow activities	Boundary classes	Use case realizations	Persistent classes	Interaction diagrams	Deployment model
Builder's View		End user support material	Components	Data model	Process model	Process-to-node mapping
Detailed View		UI design classes	Design classes	Columns, types, keys, indexes	State machines	

Figure D.3. Mapping of RUP Formalisms to the Zachman Framework (de Villiers 2001).

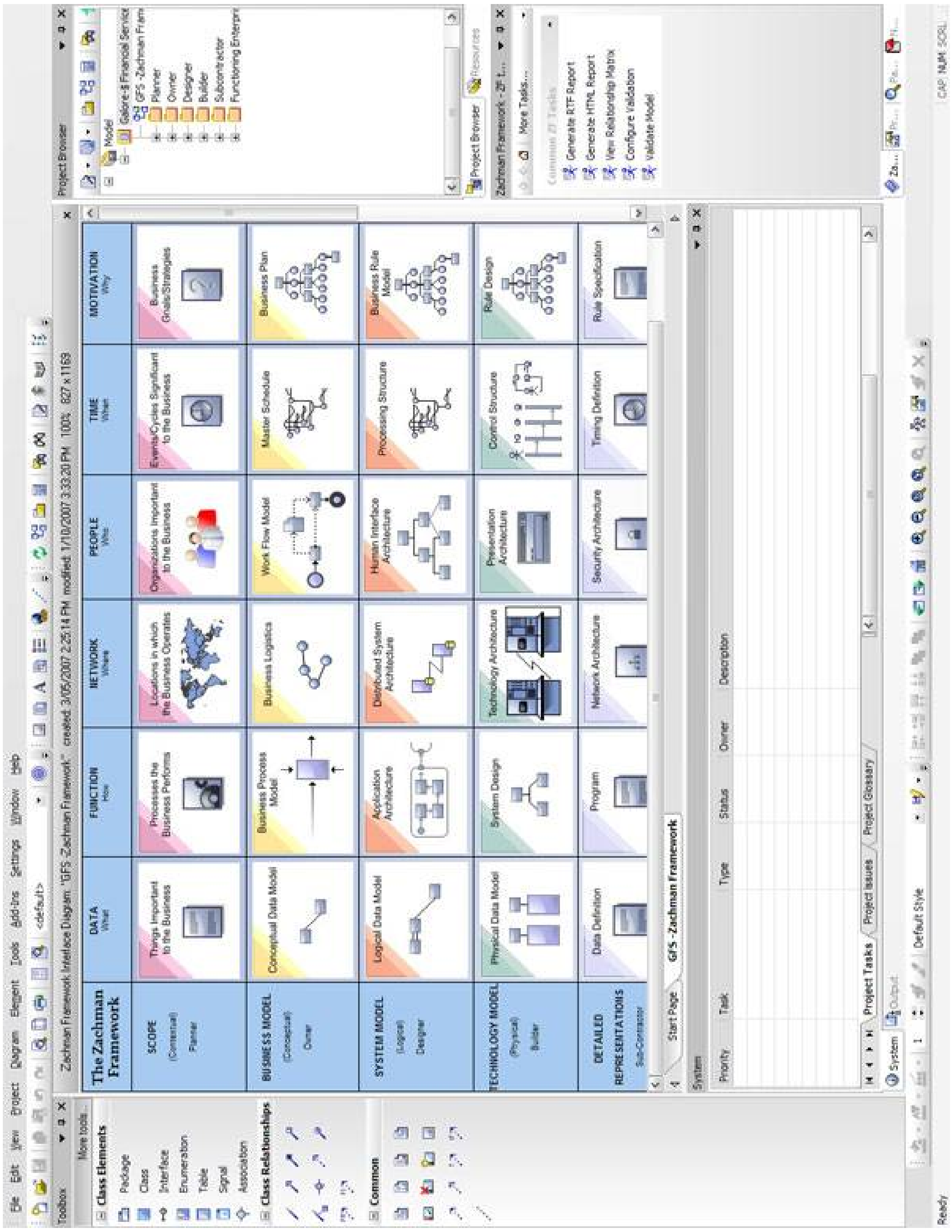


Figure D.4. Example of Zachman Framework UML Tool Support provided by Sparx Systems Enterprise Architect Tool ([online] URL: http://www.sparxsystems.com.au/products/mdg/tech/zachman/index.html).

Appendix E: Rational Unified Process for Systems Engineering (RUP SE) Overview

E.1. Introduction

The Rational Unified Process for Systems Engineering (RUP SE) was released by IBM in 2003 as an extension of the RUP to incorporate notions of systems engineering (Cantor 2003a), (Cantor 2003b), (Cantor 2003c). The RUP SE embraces life cycle stages in a similar manner to the RUP, but expresses them in a modelling context as system model levels rather than within a process framework as for RUP. It similarly uses notions of "Black box" and "White box" to reflect different decomposition perspectives, but has a much broader perspective than SW implementation, introducing different viewpoints to incorporate and represent different stakeholder perspectives.

E.2. RUP SE Representations

The RUP SE embraces the SE notion of system decomposition to provide a basis for separating different concerns, thus allowing different teams of people to contribute different reasoning about the system, without compromising the integrity of the interdependencies of the different perspectives. It also introduces additional vernacular to assist framing different perspectives.

The SE RUP supports system decomposition from two different perspectives:

- Logical decomposition into further logical systems, subsystems and components; and
- Physical system components that make up the delivered system.

The RUP SE system model has two dimensions:

- Viewpoint dimension: the context for addressing a limited set of quality concerns; and
- Model level dimension: comprising a set of UML diagrams that capture a specific level of design detail.

Here the RUP SE system model is a representation of the system including a set of views that capture all areas of concern, levels of specificity, and model entity relationships, described using UML graphical modelling techniques. Different levels of the model are constructed in terms of the degree of abstraction employed, either to hide detail, or to be more specific and expose more detail, thus presenting different viewpoints.

A View of the model shows entities that a relevant from a particular viewpoint: thus the intersection of viewpoint and level of model abstraction will reveal views of the model relevant to that viewpoint or concern at that level of abstraction (Cantor 2003a).

The RUP SE framework therefore prescribes various model viewpoints as described in Table E.1.

Table E.1. RUP SE System Model Viewpoints (Cantor 2003a).

Viewpoint	**Expresses**	**Concern**
Worker	Roles and responsibilities of system workers	• Worker activities • Automation decisions • Human/system interaction • Human performance specifications
Logical	Logical decomposition of the system as a coherent set of UML subsystems that collaborate to provide the desired behaviour.	• Adequate functionality to realise use cases • Extensibility and maintainability • Internal Reuse • Good cohesion and connectivity
Physical	Physical decomposition of the system and specification of the physical components	• Adequate physical characteristics to host functionality and meet supplementary requirements
Information	Information processed and stored by the system	• Sufficient capacity to store data • Sufficient throughput to provide timely access to the data
Process	Threads of control which carry out the computation elements	• Sufficient partitioning of processing to support currency and reliability needs.

Additional domain-specific viewpoints can be included as relevant to provide additional direction for system implementation, including but not limited to safety, security, mechanical, environmental considerations. These viewpoints represent different areas of concern in the system architecture that must be addressed during implementation.

In addition to the viewpoints, different levels of levels of specification within the model are accommodated, akin to the SE notion of life cycle stages, to reflect the evolution of the design from a general, abstract specification initially, to more detailed specifications as the design and implementation is progressed towards the final physical realisation. The RUP SE model levels are as described in Table E.2.

Table E.2. RUP SE Model Levels (Cantor 2003a).

Model Level	**Expresses**
Context	The system and its actors.
Analysis	Initial system partitioning in each of the viewpoints to establish the conceptual approach.
Design	Realisation of the analysis level to hardware, software, and people.
Implementation	Realisation of the design model into specific configurations.

Finally, the system architecture of the SW system is captured in a set of views that expresses the architecture from different viewpoints and model levels as shown in Table E.3. Each cell in the table provides a separate view of the system, whilst maintaining the integrity of the interrelationships between the individual entities that make up the entirety of the SW system.

Table E.3. RUP SE Model Framework for SW Intensive Systems (Cantor 2003a).

Model Level	**Model Viewpoints**				
	Worker	***Logical***	***Information***	***Physical***	***Process***
Context	UML organisation view	System context diagram	Enterprise data view	Enterprise locality (distribution of enterprise resources)	Business processes
Analysis	Generalised system worker view	Subsystem view	System data view	System locality view	System process view
Design	System worker view	Subsystem class views, Software component views	System data schema	Descriptor node view	Detailed process view.
Implementation	Worker role specifications and instructions	Configurations: deployment diagram with hardware and software system components			

Importantly, activity moving down model levels is recursive, adding more and more specificity to the model; in SE fashion, the model elements at one level establish the requirements at the next level down. Each model level therefore realises the requirements articulated at the higher levels above.

Therefore:

- The context model level reveals the general high level requirements;
- The analysis model level reveals how the requirements specified in the context model are met;
- The design model reveals how requirements arising from the system analysis model are met; and
- The implementation model level meets the design specification.

An overview of the RUP SE process is shown in Figure E.1, highlighting the recursive nature of the design and synthesis activity during decomposition.

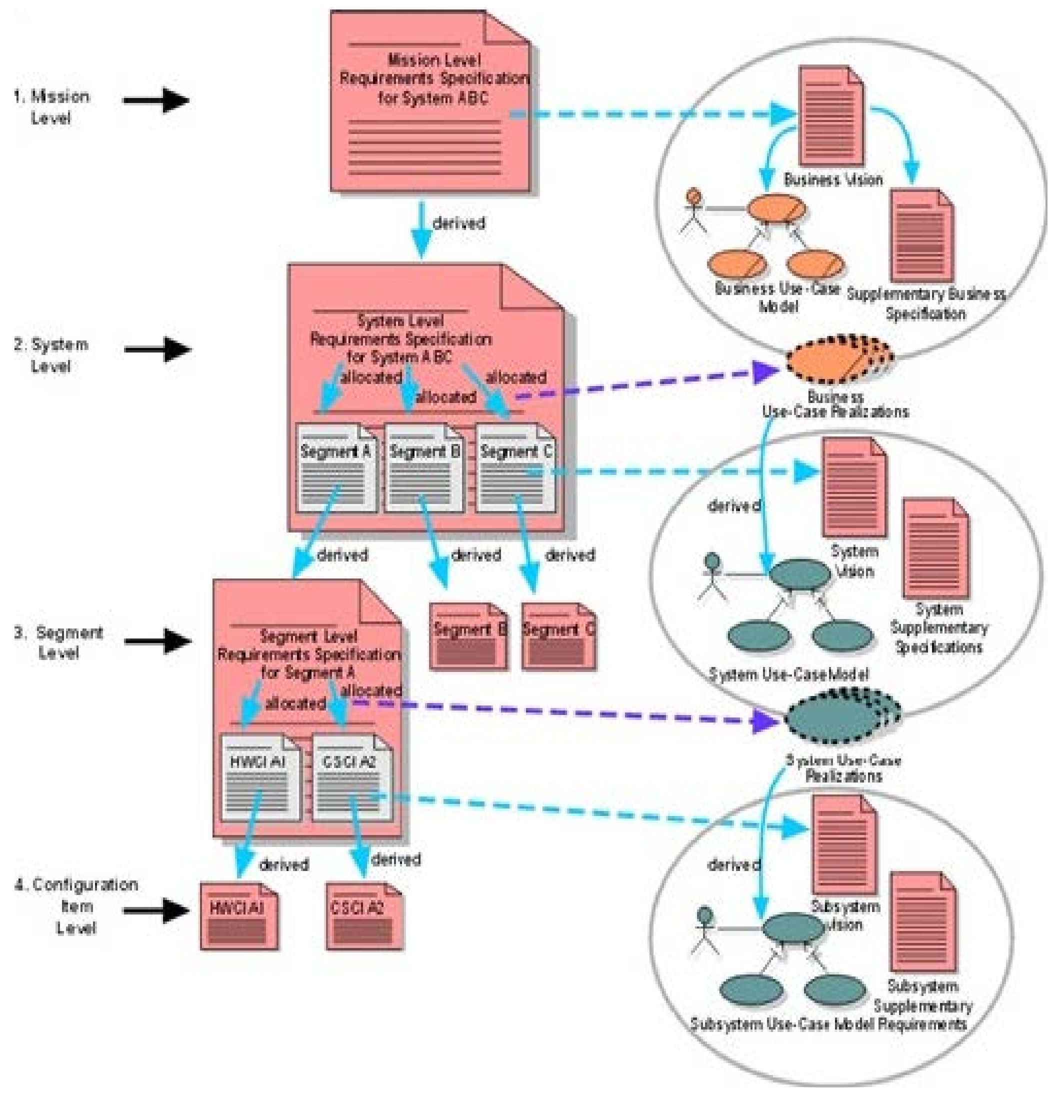

1. Mission Level
Mission Level Requirements Specification for System ABC
derived
2. System Level
System Level Requirements Specification for System ABC
allocated
allocated
allocated
Segment A
Segment B
Segment C
derived
derived
derived
3. Segment Level
Segment Level Requirements Specification for Segment A
allocated
allocated
HWCI A1
CSCI A2
Segment B
Segment C
derived
derived
4. Configuration Item Level
HWCI A1
CSCI A2
Business Vision
Business Use-Case Model
Supplementary Business Specification
Business Use-Case Realizations
derived
System Vision
System Supplementary Specifications
System Use-Case Model
System Use-Case Realizations
derived
Subsystem Vision
Subsystem Supplementary Requirements
Subsystem Use-Case Model

Figure E.1. RUP SE Process Overview (Estefan 2008).

Appendix F: UML Tool Support for DoDAF

An example of tool support provided for the DoDAF and MODAF using the UPDM ADL is provided in Figure F.1. An overview of the relationships between the primary DoDAF entities and the respective DoDAF artefacts is shown in Figure F.2. Examples of UML2 artefacts are provided in Figures F.3 and F.4. UML support for DoDAF artefacts is described in Table F.1.

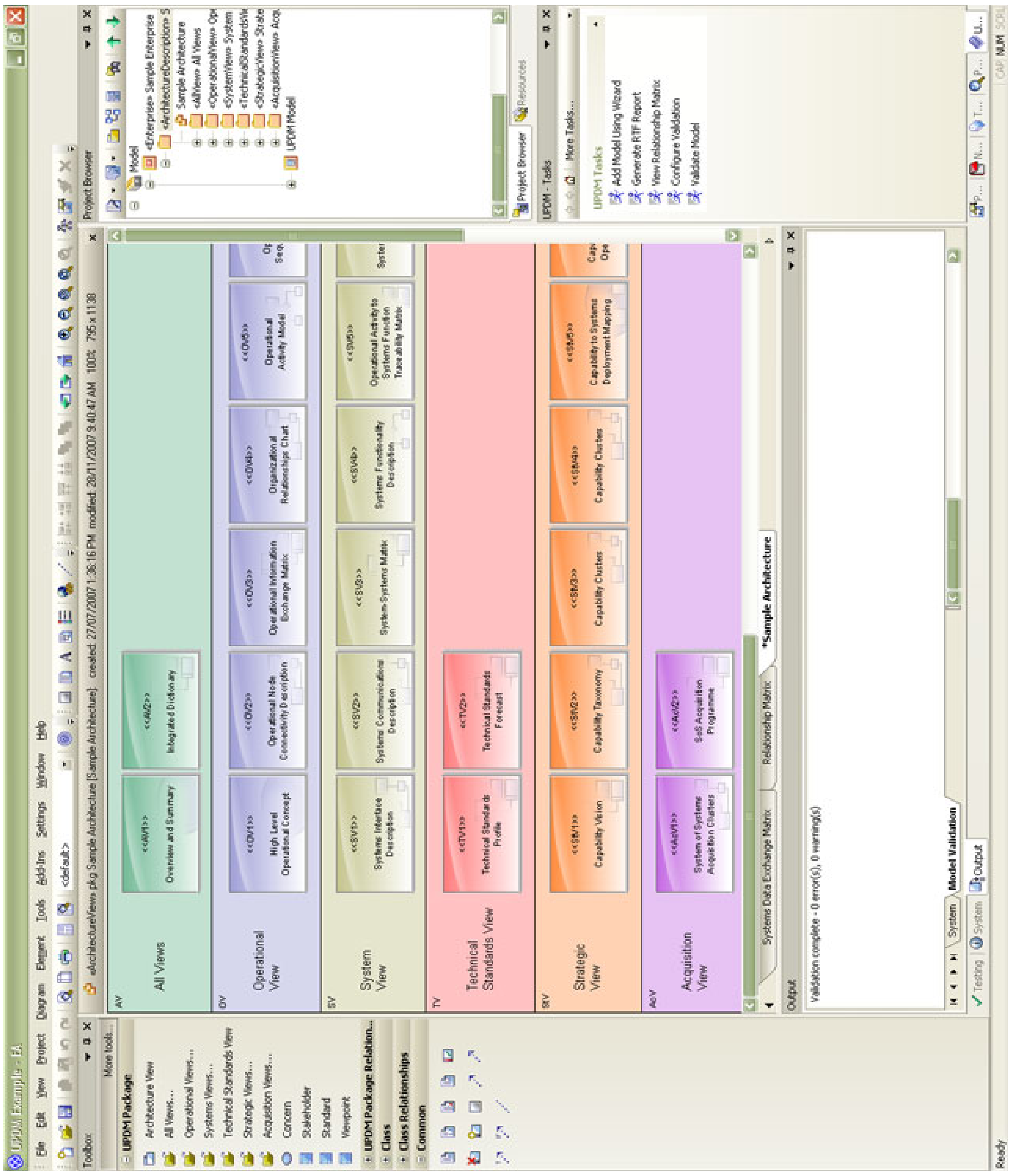

UPDM Example - EA
All Views
Overview and Summary
Integrated Dictionary
Operational View
High Level Operational Concept
Operational Node Connectivity Description
Operational Information Exchange Matrix
Organizational Relationships Chart
Operational Activity Model
System View
Systems Interface Description
Systems Communications Description
System-Systems Matrix
Systems Functionality Description
Operational Activity to Systems Function Traceability Matrix
Technical Standards View
Technical Standards Profile
Technical Standards Forecast
Strategic View
Capability Vision
Capability Taxonomy
Capability Clusters
Capability to Systems Deployment Mapping
Acquisition View
System of Systems Acquisition Clusters
SoS Acquisition Programme
Project Browser
UPDM Tasks
Add Model Using Wizard
Generate RTF Report
View Relationship Matrix
Configure Validation
Validate Model
Toolbox
UPDM Package
Validation complete - 0 error(s), 0 warning(s)
Ready

Figure F.1. Example of Tool Support for MODAF and DoDAF Artefacts using UPDM ([online} URL: http://www.sparxsystems.com/products/mdg/tech/dodaf-modaf/index.html)

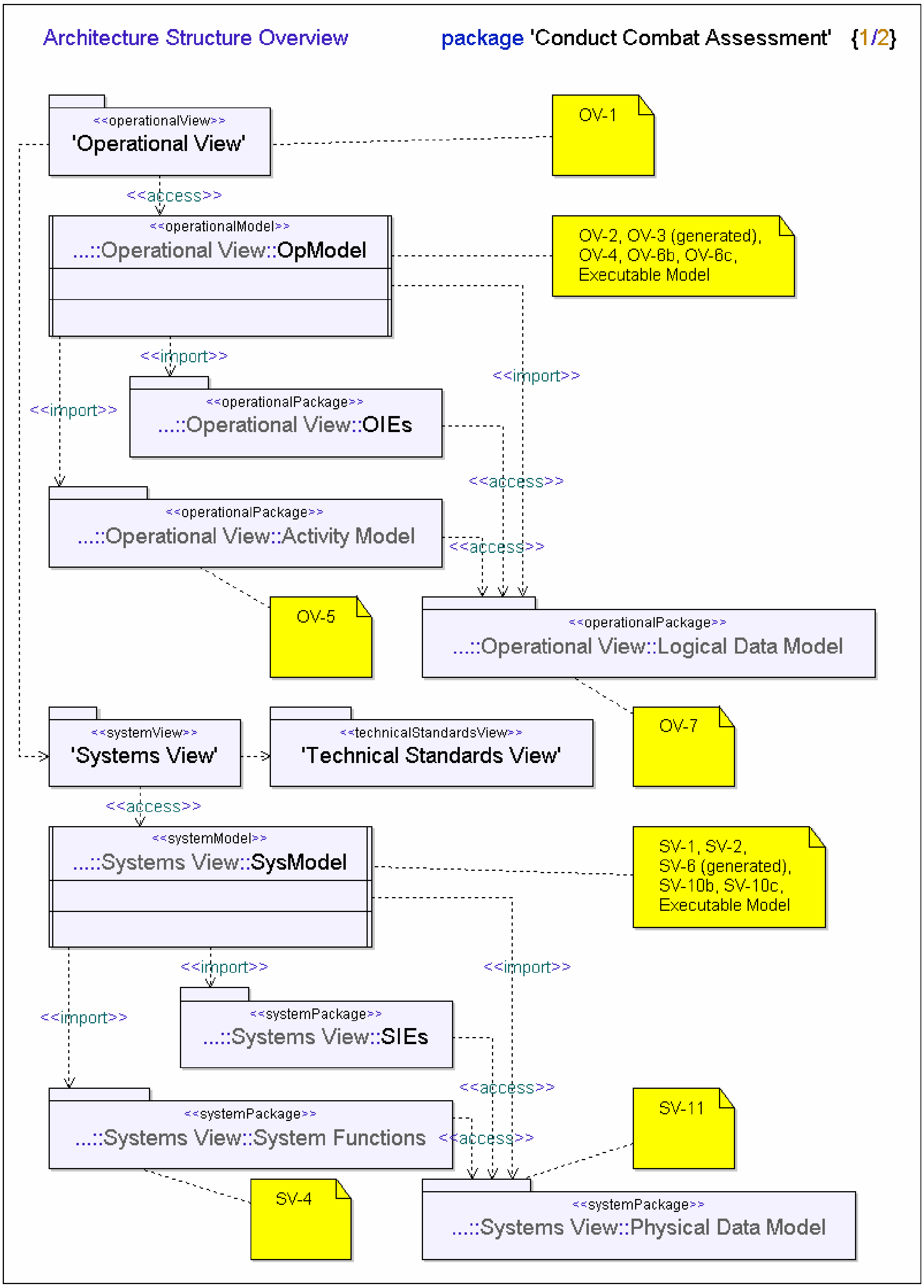

Architecture Structure Overview
package 'Conduct Combat Assessment' {1/2}
<<operationalView>> 'Operational View'
OV-1
<<access>>
<<operationalModel>> ...::Operational View::OpModel
OV-2, OV-3 (generated), OV-4, OV-6b, OV-6c, Executable Model
<<import>>
<<import>>
<<import>>
<<operationalPackage>> ...::Operational View::OIEs
<<access>>
<<operationalPackage>> ...::Operational View::Activity Model
<<access>>
OV-5
<<operationalPackage>> ...::Operational View::Logical Data Model
OV-7
<<systemView>> 'Systems View'
<<technicalStandardsView>> 'Technical Standards View'
<<access>>
<<systemModel>> ...::Systems View::SysModel
SV-1, SV-2, SV-6 (generated), SV-10b, SV-10c, Executable Model
<<import>>
<<import>>
<<import>>
<<systemPackage>> ...::Systems View::SIEs
<<access>>
<<systemPackage>> ...::Systems View::System Functions
<<access>>
SV-11
SV-4
<<systemPackage>> ...::Systems View::Physical Data Model

Figure F.2. Example of the Top-Level UML2 Package Diagram Organising the DoDAF Views (Kobryn & Sibbald 2004).

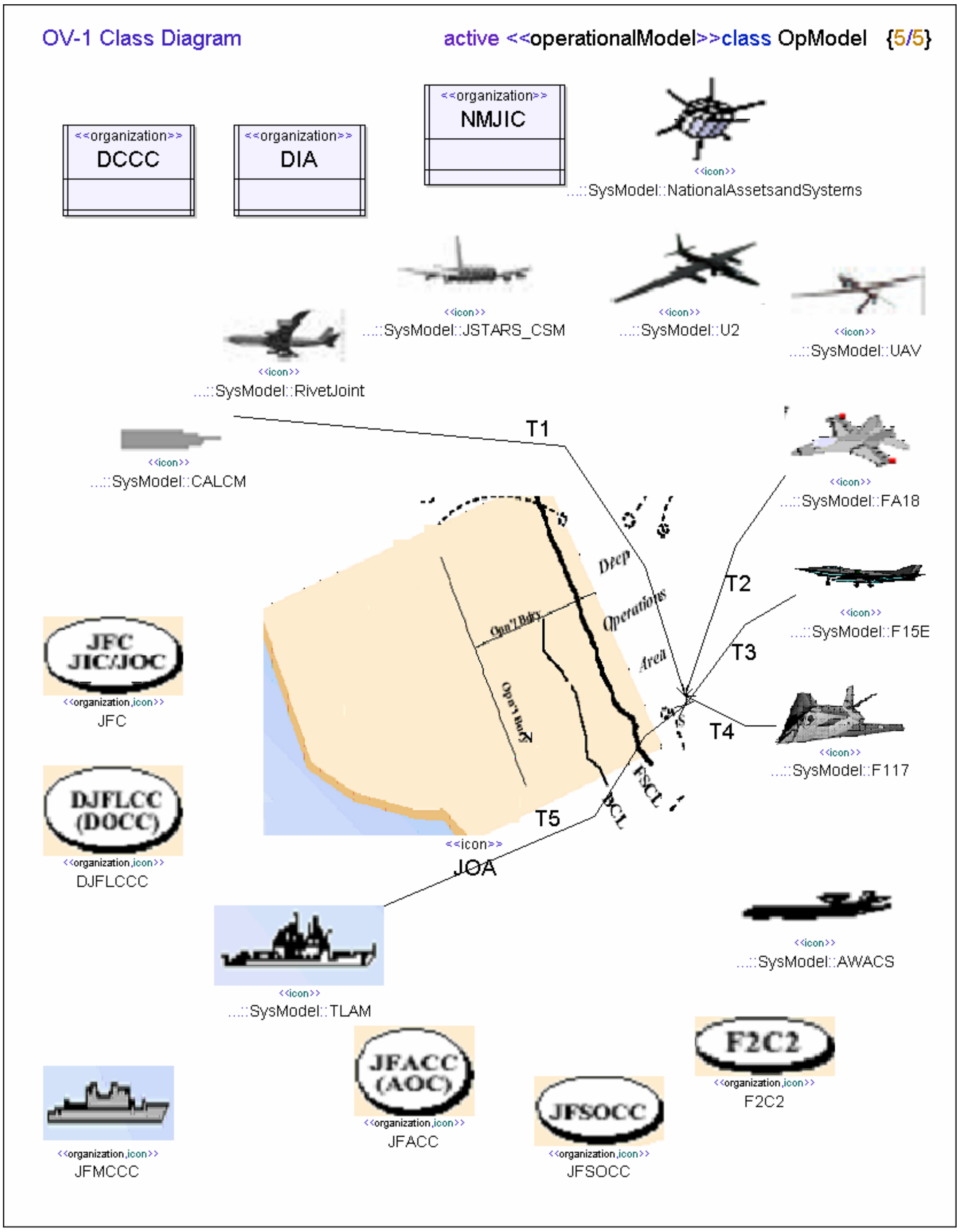

OV-1 Class Diagram
active <<operationalModel>>class OpModel {5/5}
<<organization>> DCCC
<<organization>> DIA
<<organization>> NMJIC
<<icon>> ...::SysModel::NationalAssetsandSystems
<<icon>> ...::SysModel::JSTARS_CSM
<<icon>> ...::SysModel::U2
<<icon>> ...::SysModel::UAV
<<icon>> ...::SysModel::RivetJoint
<<icon>> ...::SysModel::CALCM
T1
<<icon>> ...::SysModel::FA18
T2
<<icon>> ...::SysModel::F15E
T3
T4
<<icon>> ...::SysModel::F117
T5
Deep Operations Area
Opn'l Bdry
FSCL
BCL
<<icon>> JOA
JFC JIC/JOC
<<organization,icon>> JFC
DJFLCC (DOCC)
<<organization,icon>> DJFLCCC
<<icon>> ...::SysModel::AWACS
<<icon>> ...::SysModel::TLAM
F2C2
<<organization,icon>> F2C2
JFACC (AOC)
<<organization,icon>> JFACC
JFSOCC
<<organization,icon>> JFSOCC
<<organization,icon>> JFMCCC

Figure F.3. Example of an OV-1 Implemented using a UML2 Class Diagram (Kobryn & Sibbald 2004).

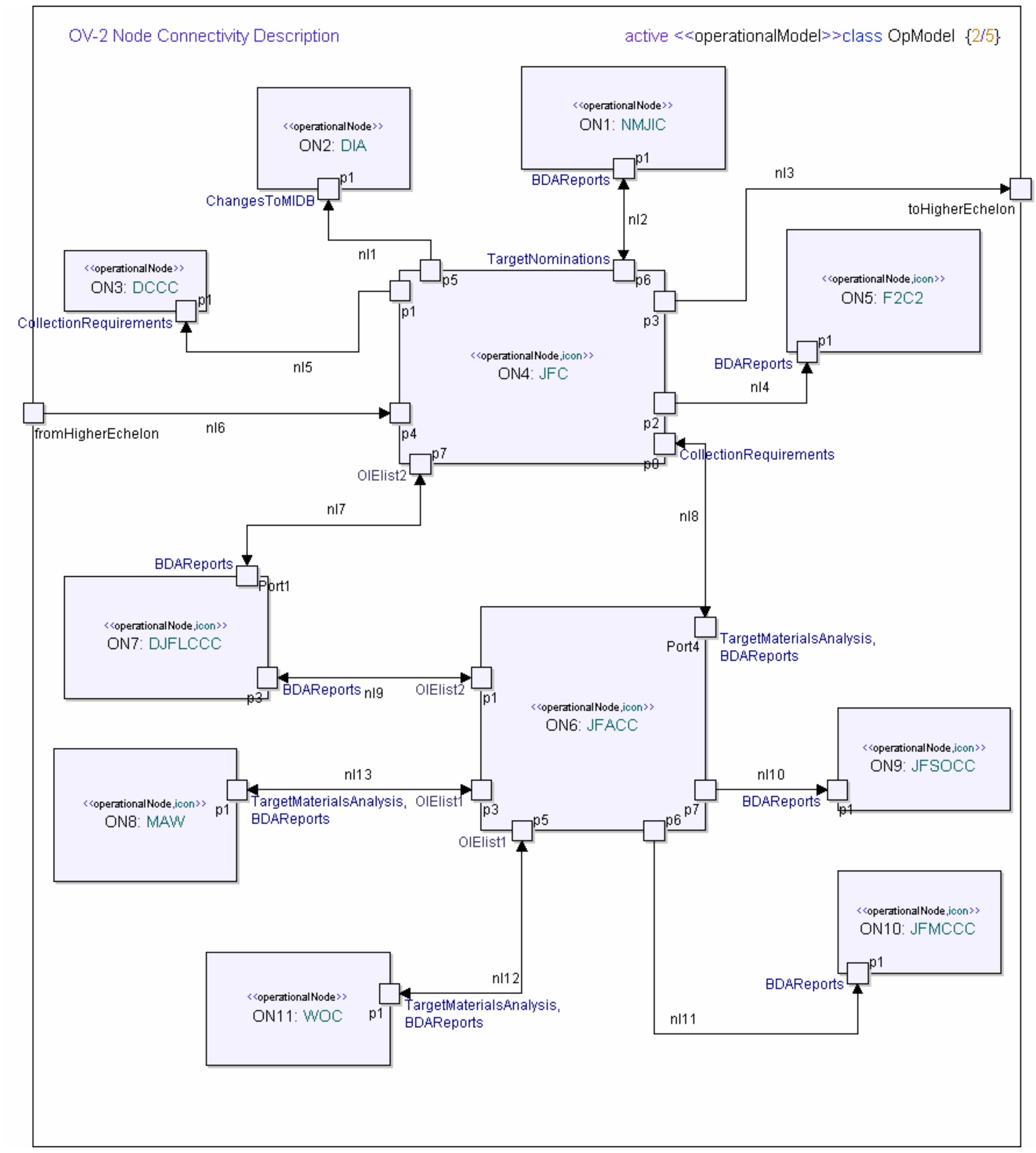

OV-2 Node Connectivity Description
active <<operationalModel>>class OpModel {2/5}
<<operationalNode>> ON1: NMJIC
<<operationalNode>> ON2: DIA
<<operationalNode>> ON3: DCCC
<<operationalNode,icon>> ON4: JFC
<<operationalNode,icon>> ON5: F2C2
<<operationalNode,icon>> ON6: JFACC
<<operationalNode,icon>> ON7: DJFLCCC
<<operationalNode,icon>> ON8: MAW
<<operationalNode,icon>> ON9: JFSOCC
<<operationalNode,icon>> ON10: JFMCCC
<<operationalNode>> ON11: WOC
ChangesToMIDB
BDAReports
TargetNominations
CollectionRequirements
toHigherEchelon
fromHigherEchelon
OIElist2
OIElist1
TargetMaterialsAnalysis, BDAReports
nl1
nl2
nl3
nl4
nl5
nl6
nl7
nl8
nl9
nl10
nl11
nl12
nl13
Port1
Port4

Figure F.4. Example of an OV2 Implemented Using a UML2 Composite Structure Diagram (Kobryn & Sibbald 2004).

Table F.1. UML Support for DoDAF Artefacts (Kobryn & Sibbald 2004).

Applicable View	**Framework Artefact**	**Framework Artefact Name**	**General Description**	**UML Diagrams**
All Views	AV-1	Overview and Summary Information	Scope, purpose, intended users, environment depicted, analytical findings.	Text (or DOORS) documents.
All Views	AV-2	Integrated Dictionary	Data Repository with definitions of all terms used in all artefacts.	UML model queries + report generator.
Operational	OV-1	High-Level Operational Concept Graphic	High-level graphical/textual description of operational concept.	Class or Use Case Diagram.
Operational	OV-2	Operational Node Connectivity Description	Operational nodes, operational activities performed at each node, connectivity and information exchange needlines between nodes.	Composite Structure Diagram.
Operational	OV-3	Operational Information Exchange Matrix	Information exchanged between nodes and the relevant attributes of that exchange.	UML model queries + report generator.
Operational	OV-4	Organisational Relationships Chart	Organisational, role or other relationships among organisations.	Class Diagram.
Operational	OV-5	Operational Activity Model	Operational activities, relationships amongst activities, inputs and outputs.	Activity diagram with Object Flows.
Operational	OV-6a	Operational Rules Model	One of three products used to describe operational activity sequence and timing – identifies business rules that constrain operation.	Text (or DOORS) document linked to Activities.
Operational	OV-6b	Operational State Transition Diagram	One of three products used to describe operational activity sequence and timing – identifies business process responses to events.	State Machine Diagram.
Operational	OV-6c	Operational Event-Trace Description	One of three products used to describe operational activity sequence and timing – traces actions in a scenario or sequence of	Sequence Diagram.

			events and specifies timing of events.	
Operational	OV-7	Logical Data Model	Documentation of the data requirements and structural business process rules of the Operational View.	Class Diagram.
Systems	SV-1	Systems Interface Description	Identification of systems and systems components and their interconnections, within and between nodes.	Composite Structure Diagram.
Systems	SV-2	Systems Communication Description	System nodes and their related communications lay-downs.	Composite Structure Diagram.
Systems	SV-3	Systems-Systems Matrix	Relationships among systems in a given architecture showing relationships of interest, e.g. system-type interfaces, planned vs. existing relationships.	DOORS Traceability View automatically populated as interfaces are defined in model. (now also supported by SysML).
Systems	SV-4	Systems Functionality Description	Functions performed by the systems and the information flow among system functions.	Activity Diagram with Object Flows.
Systems	SV-5	Operational Activity to Systems Function Traceability Matrix	Mapping of systems back to operational capabilities or of system functions back to operational activities.	DOORS Traceability View automatically populated as allocation is performed in model.
Systems	SV-6	Systems Data Exchange Matrix	Provides details of system data being exchanged between systems.	UML model queries + report generator.
Systems	SV-7	Systems Performance Parameters Matrix	Performance characteristics of each system(s) hardware and software elements, for the appropriate timeframe.	UML model queries + report generator Or linked DOORS document(s).
Systems	SV-8	Systems Evolution Description	Planned incremental steps towards migrating a suite of systems to a more efficient suite, or towards evolving a current system to a future implementation.	Project planning document linked to model elements.
Systems	SV-9	Systems Technology	Emerging technologies and hardware/software	Text (or DOORS) document.

		Forecast	products that are expected to be available in a given set of timeframes, and that will affect the future development of the architecture.	
Systems	SV-10a	Systems Rules Models	One of three artefacts used to describe the system activity sequence and timing – responses of a system to events.	Text (or DOORS) document linked to System Functions.
Systems	SV-10b	System State Transition Diagram	One of three artefacts used to describe the system activity sequence and timing – Responses of a system to events.	State Machine Diagram.
Systems	SV-10c	System Event-Trace Description	One of three artefacts used to describe the system activity sequence and timing – System-specific refines of critical sequences of events and the timing of these events.	Sequence diagram.
Systems	SV-11	Physical schema	Physical implementation of the information of the Logical Data Model, e.g. message formats, file structures, physical schema.	Class Diagram.
Technical	TV-1	Technical Standards Profile	Extraction of standards that apply to a given architecture.	Text (or DOORS) document linked to Systems.
Technical	TV-2	Technical Standards Forecast	Description of emerging standards that are expected to apply to the given architecture within an appropriate set of time frames.	Text (or DOORS) document linked to Systems.

Page classification: UNCLASSIFIED

DEFENCE SCIENCE AND TECHNOLOGY ORGANISATION DOCUMENT CONTROL DATA	1. DLM/CAVEAT (OF DOCUMENT)
2. TITLE An Analysis of SE and MBSE Concepts to Support Defence Capability Acquisition	3. SECURITY CLASSIFICATION (FOR UNCLASSIFIED REPORTS THAT ARE LIMITED RELEASE USE (L) NEXT TO DOCUMENT CLASSIFICATION) Document (U) Title (U) Abstract (U)
4. AUTHOR(S) Meredith Hue	5. CORPORATE AUTHOR DSTO Defence Science and Technology Organisation PO Box 1500 Edinburgh South Australia 5111 Australia

6a. DSTO NUMBER DSTO-TR-3039	6b. AR NUMBER AR- 016-126	6c. TYPE OF REPORT Technical Report	7. DOCUMENT DATE September 2014

8. FILE NUMBER 2014/1108347/1	9. TASK NUMBER ERP 07/369	10. TASK SPONSOR CJOAD	11. NO. OF PAGES 260	12. NO. OF REFERENCES 179

13. DOWNGRADING/DELIMITING INSTRUCTIONS Not applicable	14. RELEASE AUTHORITY Chief Joint and Operations Analysis Division

15. SECONDARY RELEASE STATEMENT OF THIS DOCUMENT

Approved for Public Release

OVERSEAS ENQUIRIES OUTSIDE STATED LIMITATIONS SHOULD BE REFERRED THROUGH DOCUMENT EXCHANGE, PO BOX 1500, EDINBURGH, SA 5111

16. DELIBERATE ANNOUNCEMENT
No Limitations

17. CITATION IN OTHER DOCUMENTS Yes

18. DSTO RESEARCH LIBRARY THESAURUS http://web-vic.dsto.defence.gov.au/workareas/library/resources/dsto_thesaurus.htm

capability development, methodology, systems engineering, complex systems, enterprise architecture, operations research

19. ABSTRACT
System modelling has been an enduring method of enquiry supporting systems analysis and design synthesis in systems engineering for decades. New generation systems modelling tools provide sophisticated modelling capability, coined Model-based Systems Engineering (MBSE). The underpinning fundamentals of systems engineering and MBSE are scrutinised in the context of the current Defence capability development process and enterprise architecture initiatives. The capabilities, relevance, and utility of new generation MBSE tools and methodologies are then examined, contrasting Defence and industry perspectives to reveal potential implications for Defence. Potential benefits to Defence are highlighted together with potential issues of concern. Related aspects of software engineering, enterprise engineering, enterprise architecting and operations research are also clarified to assist unravelling some of the complexities and interdependencies between the respective professional disciplines.

Page classification: UNCLASSIFIED