CRANFIELD UNIVERSITY

TIMOTHY MACKLEY

A PROCESS FOR THE APPLICATION OF MODULAR ARCHITECTURAL PRINCIPLES TO SYSTEM CONCEPT DESIGN

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING

PhD
Academic Year: 2018

Supervisor: Professor Philip John
January 2018

CRANFIELD UNIVERSITY

SCHOOL OF AEROSPACE, TRANSPORT AND MANUFACTURING

PhD

Academic Year 2018

TIMOTHY MACKLEY

A process for the application of modular architectural principles to system concept design

Supervisor: Professor Philip John
January 2018

ABSTRACT

A system architecture can be configured in ways that simplify both a system design and its development, by using established architectural principles such as independence and modularity. Despite systems design having been recognised as a discipline and a process as early as the mid-1900s, there are currently few methods available that address how these principles can be applied in practice. The literature search for this research has established a set of principles that can be used to develop a modular design, but has also shown that there are few formal methods available that will allow a system designer to apply such principles. This thesis examines what the key principles of modular architecture are and develops a process that enables the application of these principles to a system concept design. Key principles used are those of simplicity, independence, modularity and similarity. The concept of 'context types' is developed to allow the system designer to choose an architectural strategy that suits the system context. Another novel concept of 'functional interaction types' helps the system designer to identify critical interactions within the architecture that need to be addressed. Finally, the concept of functional interaction types is combined with existing measures of architectural 'goodness' to generate a method of evaluating the architecture that focusses on critical aspects. The process proposed is demonstrated by using a range of system examples and compared with the two of the most well-known methods currently available; Systematic Design and Axiomatic Design.

Keywords:

systems engineering, system design, system architecture, modularity, critical interaction modular design methodology

ACKNOWLEDGEMENTS

I would like to acknowledge my supervisor, Professor Philip John for his patient and supportive guidance through this PhD, which due to work pressures has taken longer than either of us might have imagined. I also appreciate the guidance provided by Professor Hoi Yeung, Dr Craig Lawson and Dr Pavlos Zachos as members of my progress review team. I acknowledge all those family, friends and colleagues that beat me to the submission of a PhD thesis, but who have never-the-less spurred me on; my daughter Dr Emma Mackley, nephew Dr Andrew Mackley, Dr James Goss, Dr Piotr Sydor, Dr Richard Halliburton and Dr Gilbert Tang (and my son, Ben Mackley MMath, for not jumping on the bandwagon). Also Professors Iain Gray and Phillip Webb, Rachael Wiseman, Lisa Rice and Maeve Williamson for their help and support. Finally I must thank Gillian for her patience and support especially during the thesis write-up period.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

LIST OF ABBREVIATIONS

3D	Three dimensional
BITE	Built in test equipment
COTS	Commercial off the shelf
DODAF	Department of defence architectural framework
DP	Design parameters
DSM	Design structure matrix
EMC	Electromagnetic compatibility
FR	Functional requirement
GDT	General design theory
GRM	Generic reference model
HCI	Human computer interface
IEC	International electro-technical commission
IEEE	Institute of electrical and electronic engineers
INCOSE	International council on systems engineering
ISO	International standards organisation
LRU	Line replaceable unit
MCDA	Multi criteria decision analysis
MODAF	Ministry of defence architectural framework
NASA	National aeronautics and space organisation
NHS	National health service
OODA	Observe, orient, decide, act
PBN	Painting by numbers
PhD	Doctor of philosophy
RAS	Relative architectural score
SADT	Structured analysis and design technique
SOI	System of interest
SOSA	System of systems approach
TRV	Thermostatic radiator valve
UAV	Unmanned Air Vehicle
UDT	Universal design theory
UK MOD	United Kingdom ministry of defence
US DOD	United States department of defence

VDI	Verein Deutscher Ingenieure (Association of German engineers)
WSOI	Wider system of interest

1 INTRODUCTION

1.1 Background

Systems engineering, and systems design in particular, has been a recognised discipline for more than 50 years, but there are few methodologies in existence that allow the system designer to purposefully design system concepts to manage their quality attributes and the risk involved in the lifecycle. As 80% of whole-life costs are determined in the concept stage (Ehrlenspiel, Kiewert, & Lindemann, 2007) and only 29% of projects developed achieve full compliance with requirements (Standish Group, 2015), early consideration of such aspects in concept design would seem important.

The author has over thirty years of experience in the field of systems design and systems engineering; twenty with a prime contractor of engineering systems. Throughout this time, there have been few significant changes to the way system design is performed. The process typically involves a systematic partitioning of requirements over successive hierarchical levels of the system with little guidance on how the partitioning should be achieved, unless there is previous experience of similar designs to learn from. Some methods have been developed and introduced over the years, many of which are discussed in Tomiyama's paper (Tomiyama et al., 2009), but these have struggled to gain acceptance in industry (Yang, 2007). However, arguably the need for a method or approach to designing effective systems has never been greater.

Systems are becoming both more complex and complicated as the promise of performance benefits from increasing integration and interdependency are sought. However, the pursuit of this often leads to unanticipated cause and effect (Perrow, 1999). The UK MOD and US DOD initiatives to develop systems of systems, represent examples of how increased dependencies are being used to increase capability, but even everyday examples such as the Ford Focus car offer examples of such increased 'dynamic complexity' (P. M. Senge, Kleiner, Roberts, Ross, & Smith, 1994).

Design authorities are also increasingly being asked to take responsibility for the overall provision of a capability or service, rather than just provision of a product. Responsibility for anticipating and managing wider systemic effects is therefore increased. Technological advances lead to pressure to improve the time to market so as to keep up with market opportunities, and competitive pressures require that time and cost in development is optimised with regard to the capability of the product produced.

The context of a system is changing more rapidly: technological advances are often disruptive requiring radical new solutions for which past experience is no

longer relevant (Clark, 1987). There is also a desire to utilise all available information and assets to best effect such as with the UK System of Systems Approach (SOSA) (Coffield, 2016). At the same time, stakeholder expectations are increased along with improvements in our engineering and processes, including areas of:

- Safety, reliability and security
- Cost in competitive environment
- Latest technology
- Reduced risk generally
- Out of the box interoperability.

The above challenges can be summarised as a need for dealing with increased scope, increased complication/complexity in both product design and enterprise approach and greater expectations of value, overall effectiveness and risk. With the increased need, what might be the reason for a lack of methods? Firstly, it is a difficult problem; “system” is a broadly applicable term that refers to concepts in many widely differing domains requiring different skills and experience to design. A system is characterised by many different parameters; it is a multi-criteria problem where an objective “best” solution is hard to determine and justify.

The aim of this research will be to develop a system design methodology that can be used to create system designs and specifically address the need to manage design effectiveness and lifecycle risk at the concept stage. The output of the concept stage is the systems architecture rather than a detailed design, and following a literature search, a modular approach is chosen as this represents a means of managing systems and controlling system behaviours more effectively. The research question will be:

How can modular architectural principles be applied to early system concept design to manage system effectiveness and reduce lifecycle risk?

In conducting the research it has been apparent that satisfactory solutions for the architecture of systems often require a variety of concepts from systems thinking. Therefore a further aim is that the approach will provide a means of unifying various strands of the systems theory and practice, so that they might be addressed and integrated within a common approach. These strands of theory and practice have originated, in part, to address problems that have been created by different contextual situations. Therefore in order to achieve a common approach systems of varying types and contexts will be examined.

Note: the literature review for this research encompasses a period of almost a century and inevitably accepted terms have changed. This is the case for the

term complex. According to the Cynefin Framework (Snowden & Boone, 2007) complex is where “the relationship between cause and effect can only be perceived in retrospect”. Prior to this model complex was often also used to describe complicated, where “the relationship between cause and effect requires analysis or some other form of investigation and/or the application of expert knowledge”. In reviewing the literature historical terms will be used and it is suggested that this distinction is borne in mind when dealing with the term “complex”.

1.2 Thesis Structure

Section 2 of this thesis describes the literature search informing the research; it addresses areas of definitions, existing systems design methodologies, architecting techniques, the effect on system context and means of architecture evaluation.

Having identified the relevant literature, Section 3 develops the research objectives, a research question and the methodology that will be employed for the research.

Sections 4, 5 and 6 describe the analysis behind the research for characterising the systems context, applying architectural principles in the different stages of development of the systems design and evaluation of the designed architecture.

Section 7 summarises the proposed methodology from the earlier analysis as a set of prescribed steps that the systems designer should apply. This methodology is then applied to three design cases with different levels of complication.

Section 8 presents two practical examples of how the proposed approach can be employed to problems of varying complication. Practical examples have been chosen to test the approach in different situations. The first example is a simple system used in a continuous professional development course run at Cranfield University, with a view to demonstrate the approach at a level that is easy to assimilate. A missile system example is then used to demonstrate the method for a more complicated application. A final example of household central heating is chosen as a design problem in order to compare the proposed method, the Critical interaction modular design methodology, with two well established methods that arguably represent the current state of the art; Axiomatic design and Systematic design. Central heating is chosen as it is an area that is well understood and there is plenty of design practice against which the results from each approach can be compared, and this is reported in Section 9.

Section 10 presents the conclusions, makes suggestions of how research might continue in this area and contains some ‘lessons learnt’ about research as a result of this PhD study.

2 LITERATURE SEARCH

2.1 Overview

A search of the literature has been performed to establish the current state of knowledge for this research and the results of this will be summarised in this chapter. The literature is arranged in sections to support the research as follows:

- **System process definitions**: the processes of System Design and Systems Architecting are often used interchangeably within the literature. Establishing their similarities and differences is necessary to help in critical analysis of previous research
- **Systems Design methodologies**: there are a number of documented Systems Engineering processes with different approaches to System Design that should be reviewed and assessed in terms of their scope, efficacy and the degree to which they are currently in used by the System Engineering community
- **System architecting techniques**: will examine the current role of patterns, architecting strategies and specific architecting methods that can be employed in the systems design.
- **The impact of context on systems design**: examines how the context of a problem can be seen to influence how a system should be architected and designed
- **System and architecture evaluation**: will review how architectures are characterised and how this might be used to evaluate the particular merit of one architectural design against another.

The literature will be analysed to determine the state of knowledge in the field to reveal particular areas of need and any current gaps that can be exploited by this research, At the conclusion of the thesis, these needs and gaps will be reviewed to establish how the research develops current thinking in this area and therefore contributes to the 'body of knowledge'.

2.2 System process definitions

2.2.1 System process definition literature

System design and *system architecture* are often used interchangeably. They are also used as both verbs *and* nouns; being parts of the System Engineering process, but also products of that process. This presents a potential source of confusion and the various terms are analysed in this section.

Systems engineering is a term that has been used for the last 70 years, but has only emerged as a discipline since 1990 with the creation of the International Council of Systems Engineering (INCOSE). The INCOSE definition of Systems Engineering is:

> *"Systems engineering is an engineering discipline whose responsibility is creating and executing an interdisciplinary process to ensure that the customer and stakeholder's needs are satisfied in a high quality, trustworthy, cost efficient and schedule compliant manner throughout a system's entire life cycle."* www.incose.org

Whilst this is a detailed definition for the 'what', a view of the 'how' of Systems Engineering can be derived from the Merriam-Webster online dictionary:

- *Engineering* is defined as "the work of ***designing*** and creating large structures or new products or systems by using ***scientific*** methods", where
- *Designing* is "to plan and make decisions about something that is being built or created", and
- *Scientific* is "knowledge about … the natural world based on facts learned through experiments and observation".

Therefore engineering, and specifically Systems engineering, might be restated as:

> "The work of planning and making decisions about building or creating systems by using methods based on evidence based (scientific) knowledge".

The reason for pursuing this line of ontological development is that it tells us that the system engineering activity is a process that should be followed, but one that requires an understanding of the system based on valid scientific or evidence based methods and techniques. There is support for this in the history of *Mechanical engineering*, where 'Design Science', arguably introduced by Redtenbacher in the 1850s (Pahl, Beitz, Feldhusen, & Grote, 2007), is defined as "scientific methods to analyse the structures of technical systems and their relationships with the environment". Only later in the 1940s was a process and methodology associated with it. Pahl and Beitz's methodology, Systematic design, dealt with electromechanical systems, but it was at this time when electronic and computer systems were only just being introduced and with it, the "traditional" engineering problem developed greater abstraction and complication. The need for systems engineering to cope with this increased complication was recognised in the late 1940s and early 1950s, and this was able

to draw upon developments in *systems science*, which has its origins earlier in the late 1920s. Therefore in the search of a methodology for systems design we should expect a process to be followed, but also an evidence based way of understanding the behaviour of the system being developed. The literature demonstrates that there is no shortage of process definitions (as explored in the next section), but a lack of definitions of how the *system design* should be designed.

The genesis of systems engineering as a formal approach is generally recognised to have been in the early 1940s at Bell Laboratories (Buede, 2000),(Kelly, 1950). Fitts (Fitts & Washington, 1951) recognised the need to identify system functions and then address how they are allocated to the elements of the system; a fundamental element of the currently known concept of *systems design*. Chestnut (Chestnut, 1965) identified *systems design* as just one of the primary functions of systems development, along with: systems analysis, systems test, systems evaluation, systems operation and systems management.

A review of electronic academic databases[1] by searching on the terms of "system architecture" and "software architecture" gives an indication of when the terminology was present in the academic community. System architecture appears to have been introduced in 1969 (Hammond, 1969) and software architecture in 1971 (Spooner, 1971), though the principle of architecture in software, in all but name, was accepted to have been first identified in 1968 (Dijkstra, 1968).

In historical terms then, it seems clear that systems engineering and systems design pre-date the use of systems architecture by 20 years or more, but the terms of system and software architecture arose at about the same time. This is perhaps not surprising as the advent of software arguably provided increased flexibility of how functions could be managed within a system, and with it, the sense that this needed to be more formally controlled. In practice the term architecture has, until recently, been more associated with software than systems (Clements & Northrop, 1996).

Ulrich (Ulrich, 1995) provides a definition of product architecture as "the scheme by which the function of the product is allocated to physical components". This is similar to the definitions for systems design, but Hitchins (Hitchins, 2008), believes that there is a fundamental difference between the systems design process and architecting:

> *"Systems Design is sometimes viewed as an esoteric, even arcane, practice; so much so, that teachers, references and books no longer refer*

1 INSPEC/ Elsevier, ISI Web of Knowledge, Scirus

to systems design choosing instead to talk about 'architecting', suggesting perhaps that design and architecting are substantially the same thing, which may not be entirely correct."

Hitchins offers the thought that a system architecture may represent different ways of viewing useful patterns within a system.

> *"Some prefer the term 'systems architect' to 'systems designer' and there does seem to be some correspondence between the ideals and goals of the civil architect and those of the designer… However, systems architecture is less well understood. At its most basic, systems architecture is the pattern formed by linked clusters and subsystems. Since such clustering and linking can occur in many different ways, there are many different patterns, so many different system architectures…"*

He also feels the need for scientific justification behind systems architecture does not currently exist.

> *"There ought to be a science of systems architecture, systems architronics, which would indicate the most appropriate architecture for systems in different situations, to assure the best system solution; no such science appears to have been formulated."*

Hitchins believes that Systems Design has traditionally represented a limited view of patterns within a system (that is to say a mapping of functions to subsystems), whereas there may be more than one architectural view and different views might be employed in different situations to achieve different goals or criteria. He suggests that the notion of systems architecture is open to acceptance of different architectural views and their corresponding optimal solutions, though it says nothing about how these different views might be reconciled for a multi-criteria problem.

Wasson (Wasson, 2006) on the other hand, does not use the term systems design, preferring to use systems architecture in its place. He refers to 'system architecture levels of abstraction' as 'system, segment, product, subsystem, assembly…', which is similar to a hierarchy of system within systems design. He also talks about logical entity relationships, physical entity relationships as architectural concepts and the partitioning, sequencing and evolution from logical to physical, which is a reiteration of the principles of the system design process. Wasson's definition of system architecture as *"…structure and framework that supports and/or enables the integrated elements of the system to provide the systems capabilities and perform missions",* perhaps suggests that architecting may have a role in organising how a system integrates within its wider system and environment. This outward looking approach contrasts with the traditional

system design process, which has tended to represent a top down, internally focused approach.

The NASA Systems Engineering Handbook (NASA, 2007), for a long time a key reference for systems engineering, takes a more limited view of architecture, seeing it as just a part of the system design process. It quotes the US Department of Defense, who in turn quote IEEE SSTD 610.12, stating that an architecture can be understood as “the structure of components and the principles and guidelines governing the design and evolution over time.” Importantly perhaps, this determines that the architecture is the structure and relationships between elements, not the whole design.

It can be seen that from the Hitchins and NASA standpoint:

- system architecture is a concept used at the system design level, representing the structure, but not the whole design
- system architecture introduces the concept of viewing a system in different ways, each view representing different patterns or architectures.

Wasson expands upon this by proposing that system architecture can be used represent the way that a system needs to be designed to interoperate within its wider system. This latter concept might lead some to believe that system architecting is a higher level process compared with systems design or even systems engineering (Rechtin, 1992)(Maier, 1998). With the hierarchical nature of systems, it is tempting to suggest that there should always be a responsible designer at the higher level. However, in practice there will be a level at which the system is not ‘designed’ in an engineering sense. Instead, at this level, one can conceive of a framework that systems will integrate into, such as is the case with a system of systems (Maier, 1998). Establishing how the system would fit into this framework then becomes the responsibility of an architect requiring a different approach to that of traditional systems design; the need behind Architectural Frameworks such as DoDAF and MoDAF.

Software and systems design have taken a somewhat parallel path from the 1960s. Software has arguably taken a more formal, structured approach, which may be because it is able to work with a representation that is largely functional and abstract. Over the years there has been much documented work on software architecture. Some of the early work in this area has been carried out by Carnegie Mellon University. In his work for establishing criteria for decomposing systems into modules, Morris and Parnas identified unconventional ways of arriving at decompositions that provided certain benefits to the designer (Morris & Parnas, 1971). They argue that it is almost always incorrect to start decomposition on the basis of a functional flowchart, and that benefits can be achieved by beginning

with a list of difficult design decisions that are likely to change. Modules should then be designed to hide or internalise such decisions from others – a technique termed encapsulation.

Later developments in the field of software architecture are detailed in Garlan and Shaws' "An Introduction to Software Architecture" (Garlan & Shaw, 1993). This documents a number of common architectural styles, including

- Pipes and filters
- Data abstraction and Object-Oriented Organisation
- Event based, Implicit Invocation
- Layered systems
- Repositories
- Table driven Interpreters

The authors recognise that these are styles that are perhaps particular to the software discipline, and recognises other styles that are used in different domains, a few of the important ones being:

- Distributed processes (an example being the "client server" organisation)
- State transition systems
- Process control systems

A feature of a style is that it determines the way patterns are made up by components, connectors and constraints. A summary of these is outlined in Shaw's paper (Shaw, 1995), where she emphasises the importance of matching the architecture to the problem along with appropriate descriptions. However, patterns have different components and these components interact in different ways that need to be distinguished. Shaw contends that a shortcoming of conventional approaches is that they often don't recognise this need.

2.2.2 Observations on systems process definitions

The terms *systems engineering* and *systems design* have been in use since the 1940s, but the terms *systems and software architecture* are more recent and emphasise that they address different concepts. The terms systems design and systems architecture are ambiguous in the literature. System design can mean both the process and the product and you are often said to be performing systems design when you create the system design: the systems design (artefact rather than process) is also often used interchangeably with the systems architecture. This does not help in the understanding of the process and so this research will use singular, widely accepted definitions for process and artefact:

- The process: systems design - *"Systems design is process of developing technical requirements, logical decompositions, and design solutions, resulting in a validated set of requirements and a validated design solution that satisfies a set of stakeholder expectations"* (NASA, 2007)
- The artefact: system architecture – *"fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution…where an architecture is what is fundamental to a system — not necessarily everything about a system, but the essentials." ISO/IEEE (ISO/IEC/IEEE, 2011)*

The introduction of the term *system architecture* has introduced further concepts, which were not addressed previously in system design:

- that an architecture can describe patterns for more than function
- system architecture can be used as a framework for systems that are not developed or owned by a single design authority i.e. a systems of systems and the basis behind architectural frameworks.

The literature describes an apparent lack of science behind *systems design* and *system architecture*, which results in a lack of understanding of how system design process can produce an effective architecture.

2.3 System Design Methodologies

2.3.1 Existing methodologies

As system design is part of the system engineering process, an understanding of the broader engineering process is required to appreciate what needs to be achieved as well as some of the constraints posed.

In their book Engineering Design (Pahl et al., 2007) the authors document the development of engineering design from 1953 to 2002. They assert that these often independent developments resemble each other in many ways and that this similarity has led to a consensus approach on engineering design known as VDI guidelines 2222 and 2221. Whilst accepting that the consensus approach is a unification of similar processes, Roozenburg and Cross (Roozenburg & Cross, 1991) maintain that it is not a universally accepted approach, describing it as "a weak or heuristic" methodology based on "weak knowledge (experience)" requiring "interpretation by the designer of the vaguely defined 'rules' and terms, and, even if properly applied success is not guaranteed". Their argument, supported by Finger and Dixon (Finger & Dixon, 1989a), is that this methodology and the many others that represent this consensus, present a process without defining the evidence based knowledge that will ensure a successful system; the what, but not the how.

Instead Finger and Dixon propose that a more useful methodology would be based on developments in “architectural and industrial design”. Hillier (Hillier, Musgrove, & O’Sullivan, 1972) originally proposed this, arguing that “we cannot escape from the fact that designers must, and do, pre-structure their problems in order to solve them”. Finger and Dixon refer to such methodologies as ones that describe what attributes the product should have rather than how the process should proceed; examples given are those of Suh (Suh, 1997) and Taguchi (Taguchi, 1986). Roozenburg and Cross however readily admit that “there is no well-formulated ‘consensus’ model of the design process in architecture and industrial design” (Roozenburg & Cross, 1991).

According to Wynn and Lawson abstract methodologies are “characterised by a small number of stages or activities and do not describe the specific steps or techniques which might be used to reach a solution” (Wynn & Clarkson, 2005), and are “…about as much help in navigating a designer through his task as a diagram showing how to walk would be to a one year old child...” (Lawson, 1980). These comments relate to abstract methodologies such as General Design Theory (GDT) (Yoshikawa, 1981) and Universal Design Theory (UDT) (Grabowski, Lossack, & El-Mejbri, 1999).

There are many methodologies associated with design and there have also been many attempts to classify them in more detail. However, according to (Malmqvist & Axelsson, 1996) this has not resulted in a reduction of competing methodologies or the emergence of a particularly favoured option. Classifications from various sources are available in order to establish a view on the important aspects that a methodology of design should address (Wynn & Clarkson, 2005), (Evbuomwan & Sivaloganathan, 1996), (Tomiyama et al., 2009), (Finger & Dixon, 1989a) and (Finger & Dixon, 1989b). These are used in the next section to generate a means to determine the completeness of a system design methodology.

It is not possible to describe all methods here as many have been proposed over time (Pahl et al., 2007). Tomiyama (Tomiyama et al., 2009) lists a variety of systems engineering/design methodologies suggesting those that have made the greatest impact in both academia and in industry. However, a recent survey of design methods in industry (Yang, 2007) determined that many of the main methodologies in the academic world are not well appreciated or used within industry. For those methods highlighted by Tomiyama the relevant data is given in

Table 1.

Table 1: Comparison of popularity of systems design methodologies

	Not familiar	Used in work (not useful)	Used in work (useful)
Axiomatic design	81.4%	0.0%	3.5%
Systematic design	77.9%	0.0%	3.5%
Pugh Concept Selection (Total design)	68.6%	2.3%	12.8%
TRIZ	79.1%	0.0%	2.3%

Clearly usage of these methods within industry is at a very low level, and a motivation for the development of a methodology that has greater appeal to industry. In the absence of a recognised methodology in academia, what is the current practice employed by industry? In the author's experience, the NASA Systems Engineering Handbook (NASA, 2007) has long been considered a reference text for systems engineering. This provides a description for systems design (Figure 1), but again describes more of what to do and not how to perform it. For instance, it describes the need to partition systems, but not guidance or rules about how this should be achieved.

Similarly INCOSE, the international organisation advising on Systems Engineering, have published a Systems Engineering Handbook (Walden, Roedler, Forsberg, Hamelin, & Shortell, 2015). Whilst it references certain methods/techniques, these are merely examples and this description too provides a process description with little guidance on how it should be applied to different situations. Figure 2 summarises the process flow, but there is no evidence based methods that determine how to define, refine, synthesise, analyse or select favourable architectures.

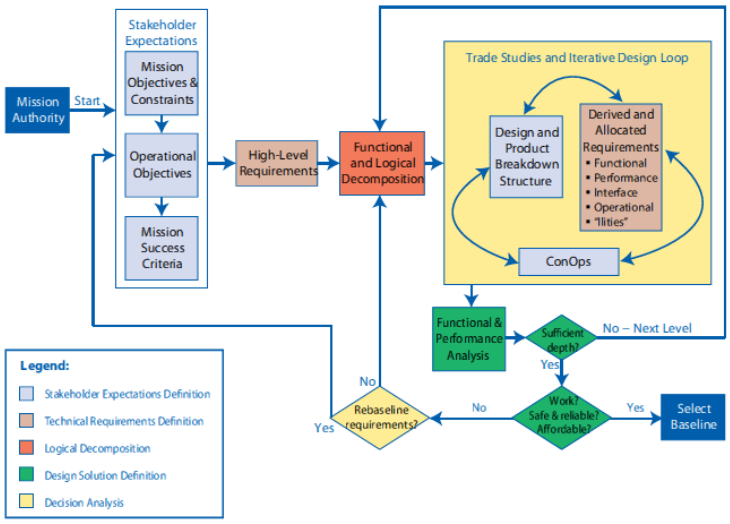


Figure 1: Systems design process (NASA Systems Engineering Handbook)

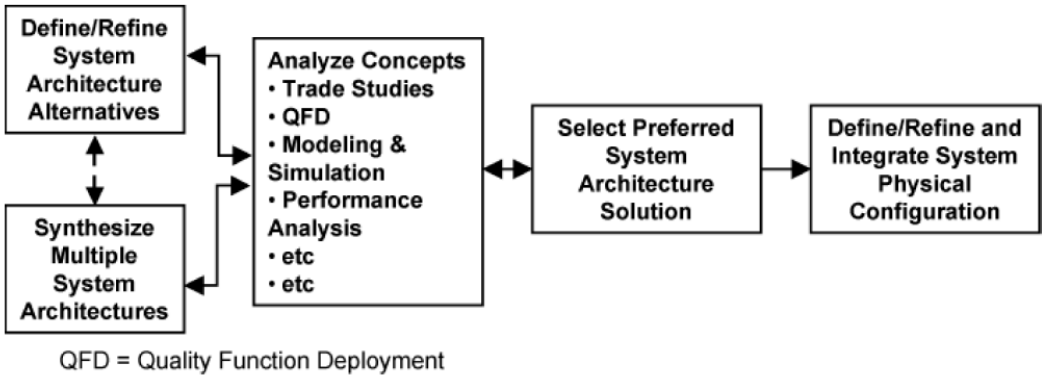


Figure 2: INCOSE system design process

2.3.2 Observations on current methodologies

The literature has many process definitions, but many are vague and left free to the interpretation of the system designer and therefore relying on experience of similar designs. It has been argued that methods based on architecture can be more beneficial, but there is no consensus on what this should be and usage by industry is very low. The focus of many methodologies is on “what” needs to be done in system design to develop an architecture, rather than “how” it can be achieved.

The widely differing approaches suggest the need of a way of classifying a methodology. Various classifications of systems engineering methodology have been identified (Evbuomwan & Sivaloganathan, 1996; Finger & Dixon, 1989a; Tomiyama et al., 2009; Wynn & Clarkson, 2005) with different authors adopting different categories and no one classification method covering all. The following categories emerge from combining the classification methods:

- Scope: how much of the lifecycle does it cover? This is important to see if the methodology can “stand-alone” or will need to be augmented/ combined with another methodology.
- Starting point: what is the starting point for the problem to be solved? Not all problems start at the same point, so how flexible is the methodology to different problems?
- Approach: what type of approach to the problem can be employed? The approach used will impact on its validity and on the requirements placed on the competencies of the user. For instance is it a process to actively influence the design toward an outcome or is it one to ensure whether an experienced person has followed the right steps to achieve an outcome? And what is its purpose; is it a prescription to follow or a description to educate?
- Models: what are the steps of the process and how are they described?
- Support: what support does the methodology provide?
    - Methods: how are individual tasks addressed? Methods provide a clear prescription of how the task should be performed.
    - Means: what means are employed to perform the tasks? Are there tools that can facilitate the process?
    - Representations: in what way is the information represented? Notations can improve the formality of the output
- Aim: what does the methodology consider success to be? Is the process to optimise a design or assure a compliant solution?

Each of the methodology classifications is mapped against these categories in

Table 2, showing the specific terms used in each case. The final row of the table contains a synthesis of all the classifications. This synthesised classification can be used to show how complete a methodology is.

Table 2: Classifications of system design methodologies

	Scope	Starting point	Approach	Models	Support			Aim
					Methods	Tools	Representations	
Wynn		Problem focussed, Solution focussed	Abstract, Analytical, Procedural (design focused), Procedural (project focussed)	Stage based, Activity based	Methods for activities			
Evbuomwam		Routine/ non-routine design, Re-design of existing, Creation of new design	Semantic school, Syntax school, Past-experience school, Prescriptive, Descriptive		Methods for activities	Computer based		Prescriptive models based on product attributes
Tomiyama		New design, Improved design	Managed process				Represent design	
Finger			Prescriptive, Descriptive		Analysis, Design for…	Computer based	Representation	Prescriptive models of the design artefact
Mackley	Complete lifecycle or Partial lifecycle	Problem solving or Solution improving	Abstract or concrete, Procedural (design or project) or Analytical, Prescriptive or Exemplar	Lifecycle stage based or Activity driven	Method supported	Tool supported	Notation support	Focus on improvement or assurance

This classification will be used later to analyse existing systems design methodologies in terms of coverage and completeness.

2.4 System architecting techniques

Architecture establishes patterns that can aid in the design process. According to Crawley (Crawley, Weck, & Eppinger, 2004), these can contribute to understanding, designing and managing complex systems. However, the ways in which elements of a system should be arranged as a pattern in order to influence either system features, attributes or properties are not evident. Literature has been found that has focussed on:

- Intrinsic patterns that need to be respected within the design
- Architecting strategies or rules that encourage good design
- Architectural perspectives that aim to control particular attributes of the design
- Architecting methods

2.4.1 Patterns

The “pattern” concept in architectural design is attributed to Alexander (Alexander, 1979), consisting of three parts:

- A context that describes when a pattern is applicable
- The problem (or “system conflicting forces”) that the pattern resolves in that context
- A configuration that describes the physical relationships that solve the problem.

The principle to have a set of pattern based designs that, in a given context, will behave in an appropriate way in order to satisfy a requirement. The pattern approach offers a “design” that is proven in one or more contexts, to be assessed for application in a new context. However, although a full design could be assessed as to whether it is fit for a given purpose, the fitness of the pattern will usually be approximate in one of three ways:

- an incomplete knowledge of or partial fit with the new context
- a different set of requirements to be met
- a design which is often at least slightly different from the original pattern.

There is a benefit of not starting from scratch in the design process and taking advantages of the proven qualities of the pattern. However the methods needed to extract patterns from existing designs in a way that captures their benefits need to be less onerous than methods involved in designing from scratch and the pattern has to be shown to be valid to the new context.

Pattern design has received considerable interest in the field of software design (Coplien, 1997), (Price, 1999), (Gamma, Helm, Johnson, & Vlissides, 1993) where it is used in support of the Object Oriented Design approach. Examples can also be seen in the field of systems thinking where patterns in the form of Systems Archetypes have been identified (P. Senge, 1990; P. M. Senge et al., 1994). In the latter, these archetypes are used as ways of diagnosing the behaviour of the system and its behaviour in context as indicative of imbalances in the implementation.

2.4.2 System Architecting Strategies

Fricke and Schultz identify three fundamental architecting strategies (Fricke & Schulz, 2005): simplicity, independence and modularity

2.4.2.1 **Simplicity**

It is intuitive that reducing the complication of a system by concentrating purely on what is needed is a good strategy as it avoids the design, verification and maintenance associated with unnecessary artefacts, thereby reducing effort and cost. Fricke and Schultz identify the concept of design streamlining as a means of simplifying by minimising interfaces and secondary functions, and focusing on already existing functions where possible.

Suh (Suh, 1990), takes this further by suggesting in his Information Axiom, that an architecture can be optimised by minimising the information content of the design, thereby making it as simple as possible. The principle is that simpler designs involve less information and Suh suggests seven Corollaries to his Axioms, six of which relate to achieving these aims which are:

- Minimisation of functional requirements
- Integration of Physical parts
- Use of standardisation
- Use of symmetry
- Largest tolerance (in stating requirements)
- Uncoupled design with less information

Again it is intuitive that reducing the information content can contribute to reducing the complication of a design, but it is perhaps harder to see how these could be used in calculations to provide objective support to the design process (Kim & Cochran, 2000). Two numerical methods are found in the literature. The first is Suh's calculation of information content and the second is a measure used in the methodology TRIZ, where Altshuller defines a closely related measure to Simplicity as Ideality (Altshuller, 2002), where:

> *Ideality = Sum of useful functions/[Sum of harmful functions + Sum of costs]*

Equation 1: Ideality equation

Gershenson and Prasad (Gershenson & Prasad, 1997) describe a related means of tackling complication in a manufacturing process, where "process similarity" is a way of grouping "components and sub-assemblies which undergo the same manufacturing processes". Here it can be seen that similarity is a strategy for tackling complication as it reduces the effective information content by re-using the same information throughout the design.

The concept of simplicity is addressed from a slightly broader perspective in Maeda's account (Maeda, 2006) defining 10 Laws and 3 Keys:

"Ten Laws

1. Reduce – the simplest way to achieve simplicity is through thoughtful reduction.
2. Organise – organisation makes a system of many appear fewer
3. Time – savings in time seem like simplicity
4. Learn – knowledge makes everything simpler
5. Differences – simplicity and complexity need each other
6. Context – what lies in the periphery of simplicity is definitely not peripheral
7. Emotion – more emotions are better than less
8. Trust – in simplicity we trust
9. Failure – some things can never be made simple
10. The one – simplicity is about subtracting the obvious, and adding the meaningful

Three Keys

1. Away – more appears like less by simply moving it far, far away
2. Open – openness simplifies complexity
3. Power – use less, gain more"

2.4.2.2 **Independence and modularity**

Suh (Suh, 1998) maintains that "there are two axioms that cover good design", the Information Axiom has just been described, but his first axiom is to "Maintain the independence of the Functional Requirements". Following the Independence principle helps to achieve minimal coupling in a system, which will simplify dependencies within the design – the benefits of this are clearly stated by Fricke and Schulz (Fricke & Schulz, 2005) "each system function or functional requirement has to be satisfied by an independent design parameter… changing a design parameter does not affect any related design parameters and thus not the proper operation of related functions". Furthermore, achieving Independence is key to creating a "unified description" between functional and formal descriptions that Alexander (Alexander, 1964) maintains is key to dealing with complex systems.

Simon (Simon, 1962) notes that the complexity of systems frequently takes the form of hierarchy, and that systems with hierarchy have the property of "near-

decomposability"; that is to say that there are configurations where intra-component linkages are generally stronger than inter-component ones. He argues that hierarchy:

- Favours the evolution of complex systems
- Exhibits relatively simple dynamic behaviour
- Makes a system easier to describe and to understand how it should develop and evolve.

Simon proves that hierarchy can help us to describe and understand the system in a way that enables us to both better manage its behaviour and minimise the effort required to do so; the fact that hierarchic systems can evolve far more quickly than non-hierarchic systems of comparable size is an important finding in terms of their development. A further important distinction of hierarchies is how they are described; Simon identifies that physical hierarchies are described primarily in spatial terms, whereas organisational hierarchies are defined primarily in terms of interactions. He asserts that these can be reconciled by defining hierarchy in terms of intensity of interaction, which is theme we will return to. It is noticeable that much of the literature on this and related subjects is to be found in management journals, which tends to reflect the lack of application to engineering problems.

Simon maintains that the recognition of hierarchy and the near-decomposability of systems allows the analytical benefits of a reductionist approach whist encouraging a holistic view; "In the face of complexity, an in-principle reductionist may be at the same time a pragmatic holist". So what represents a "near-decomposable" boundary between component subsystems? Simon asserts that "(a) in a nearly decomposable system the short-run behaviour of each of the component subsystems is approximately independent of the short-run behaviour of the other components; (b) in the long run, the behaviour of any one of the components depends in only aggregate way on the behaviour of the other components". This has resonance in a paper (Orton & Weick, 1990), where the authors identify that decomposition should be along the lines of "loose coupling". Glassman, (Glassman, 1973) defines Loose Coupling as being present when systems have few variables in common or the variables they have in common are weak. In an attempt to qualify "weak" Weick, (Weick, 1982) suggested that this was when elements effect each other "suddenly (rather than constantly), negligibly (rather than significantly), indirectly (rather than directly) and eventually (rather than immediately). A key finding of Orton's paper, however, was that "loose coupling" was a misunderstood concept. He maintained that systems should be designed as loosely coupled as possible and that practitioners saw coupling on a scale of tight- to loose- coupling. If loose coupling is considered

desirable then total decoupling would represent the optimal case, but this would no longer represent a system and presumably any benefit of synergies between the subsystems will have been lost. Orton preferred to establish a spectrum of coupling ranging from decoupled through loosely coupled to tightly coupled, where these represent varying attributes of responsiveness (representing determinacy and interdependence) and distinctiveness (representing spontaneity, independence and indeterminacy).

Modularity in practice has similarities with the principle of Independence. According to their book on Product Design and Development (K. Ulrich & Eppinger, 2008), a modular architecture is composed of "chunks", where modularity is achieved where the architecture "has the following two properties:

- Chunks implement one or a few functional elements in their entirety
- The interactions between chunks are well defined and are generally fundamental to the primary functions of the product".

This definition implies that strict independence is not required for modularity, with chunks potentially relating to more than one functional requirement. However, an independent design would also be a modular one.

Literature reviews (Campagnolo & Camuffo, 2009) and (Gershenson, Prasad, & Zhang, 2003) shed some light on the modularity; Campagnolo and Camuffo find that there are different types of modularity dealing with the *product*, the *production system* and the *organisation* and suggest that different drivers, be they technical or commercial, might drive different strategies in each dimension. Within *product architecture* there are also differing perspectives of function and lifecycle, of which Sako (Sako, 2003) observes that there can be difficulty in identifying a single optimal decomposition as different phases of the lifecycle have different objectives and each would potentially drive the architecture into different configurations. Another classification (Baldwin & Clark, 2004) identifies modularity in design, production and use.

Bayliss and Clark, (Bayliss & Clark, 1997) maintain that the practice of modularity in design is well known in the computer industry where designers achieve modularity by dividing their designs into visible and hidden information. The visible information represent "design rules" that the designer of the module has to comply with, but leaving them free to implement the rest of the design (a process known as 'information hiding' (Parnas, 1972) in any way that they wish. The visible design rules consist of architecture (the modules and what their functions will be), interfaces (how they will interact, fit together, connect and communicate) and standards for testing conformity and measurement of performance. The authors however readily admit that the determination of the rules is a difficult task as it requires "the designers of the modular systems [to]

know a great deal about the inner workings of the overall product or process in order to develop the visible design rules" and "they have to specify those rules in advance". The cases that Baldwin and Clark examine were also mainly related to appropriate work-share, allocation of tasks and reuse of components. These have clear potential benefits from a project management perspective, but it is as likely that modularity can have a beneficial effect on more technical areas such as system performance and effectiveness. The same authors point out that *modularisation* requires that *"every important cross-module dependency must be understood and addressed via a design rule" and that the "density of dependencies matters"*. They discuss that modularity showing promise in the one and two dimensional world of computers would seem an easier proposition than for most mechanical systems that have complicated, 3-dimensional designs to deal with (Baldwin & Clark, 2004)

Ericsson and Erixon (Ericsson & Erixon, 1999) describe a modular product platform design approach, where the product platform is a set of products "built from a common structure, consisting of a set of modules and interfaces. It produces company–specific deliverables that can be efficiently developed, marketed and produced…". They assign metrics to various characteristics of the modular design as shown in Table 3 (for which some evaluation methods are provided).

Table 3: Metrics for characteristics of modular design

Product characteristic	**Effect**
Interface complexity	Lead time in development
Share of carryover	Development costs
Share of purchased modules	Development capacity
Assortment complexity	Product costs
Share of purchased modules	System costs
Number of modules in product	Lead time
Share of separately tested modules	Quality
Multiple use	Variant flexibility
Functional purity in modules	Service/upgrading
Material purity in modules	Recyclability

These metrics cover aspects of design, production, use and lifecycle with the implication that a modular design has implications on them all and these should be addressed.

2.4.2.3 **Lifecycle Modularity & Similarity**

A number of researchers have referred to the concept of architecting for the lifecycle. An example of this is Lifecycle Modularity (Gershenson & Prasad, 1997), which in addition to ensuring that there is independence throughout the

entire product lifecycle, requires that each module is also processed in the same manner during each lifecycle stage – this they term Similarity. Thus they define three facets of Lifecycle Modularity:

- Attribute Independence
- Process Independence
- Process Similarity

Here the independence discussed earlier (referred to here as Attribute Independence) has been addressed by the System Designer taking account of the measures required to sustain the product over its life e.g. use of components with adequate and compatible lifetimes. An example of process independence would be that if a component is chosen that may fail in the lifetime, then the corresponding processes of test, diagnosis, replacement and disposal should display both independence and similarity. Whilst this is easy to appreciate, in finding the reasons why we should embark on such an analysis, we need to examine the potential benefits i.e. asking the question "why should we improve Lifecycle Modularity".

Gu and Sosale (Gu & Sosale, 1999)[2]describe the following reasons for modular design across the lifecycle (against each is an indication of strategies that may be employed to achieve the modular design)

- It enables parallel development - this requires independence in organisation, which can be achieved by functional independence and loose coupling of system components thereby facilitating individual teams to perform development in parallel
- Efficient and flexible production – this is another form of organisational independence, where loose coupling of system components can allow flexibility in production and assembly
- Increased standardisation – common can be enabled by functional independence, allowing economies of scale
- Common services allowing more efficient maintenance action – benefits are provided by grouping of components based on frequency of failure, level of diagnosis, required maintenance action, and required line of maintenance (e.g. *Line Replaceable Units)*
- Easier upgrade – functional independence and loose coupling between system components can be used to reduce impact on the rest of the system and cause minimal disruption during updates

[2] Also a similar work by (Huang, 2000)

- Easier reconfiguration – functional independence allows additional components or module to be easily added, increasing the utility of the product
- Better recycling - identification and grouping of reusable components as well as grouping of components by material types from the perspective of recycling or disposal
- Increased variety and customisation – functional independence and loose coupling of system components allows variants of the design for different needs/purposes

Agreeing with the previous authors in a number of areas, Erickson and Erixon (Ericsson & Erixon, 1999) identify the following additional benefits of a modular approach to design, although some of these are as a result of sharing resources across the variants of a product enabled through a modular approach:

- Shorter lead time in development and assembly due to reduction in interface complexity and concurrent assembly processes respectively
- Reduced defects due to increased opportunity for testing at module level
- Increased interchangeability due to reduced functional interfaces for independent designs

The following being for products with variants only:

- Greater development capacity and reduced system costs due to sharing of modules
- Reduced development costs due to carryover from other programmes
- Reduced product costs due to the potential sharing or production tools across product ranges
- Increased flexibility if there are alternate module options.

2.4.3 Architectural Perspectives

There is a small body of work that looks into architectural approaches that are designed in order to promote certain system attributes. Whilst this is apparently in its infancy, two authors talk about system perspectives (Woods & Rozanski, 2005) and system aspects (Wijnstra, 2001) based around quality attributes. These consider that an architecture comprises both entities and relationships of a system and therefore, any aspect of the design that involves entities and relationships can be viewed as architectural. In a traditional systems design, functions are identified and the relationships between these functions include interfaces of information, resource and control flow; this represents an abstract architecture for which the quality attribute is the performance of the functionality. Using a similar line of argument, a case can be made for a safety architecture where the entities and relationships are those that are safety critical.

Architectures to address different quality attributes are likely to identify and favour different architectural dependencies and therefore it is unlikely that a single architectural solution will respect the potentially conflicting needs; an argument supported by Sako (Sako, 2003). Indeed, it is not clear that all potential strategies can result in a single "best" architecture, where for instance the optimal safety architecture can align with the optimal architecture for performance. Chung (Chung & Leite, 2009) promotes the idea of "soft goals" that indicates the subjective nature of quality and that implicitly recognises the trade-offs between non-functional parameters; that there cannot be a single "best" answer. Never the less, he maintains that identification of how different architectures contribute to these soft-goals is important.

An indication of the difficulty in achieving a practical architectural strategy with the aim of optimising quality attributes is discussed by Alexander (Alexander, 1964). Alexander contends that the concept of a quality attribute such as safety *"…is convenient and helps hammer home the very general importance of keeping designs danger-free, but it is used in the statement of such dissimilar problems as the design of a tea kettle and the design of a highway interchange. As far as its meaning is concerned it is relevant to both. But as far as the individual structure of the two problems goes, it seems unlikely that the one word should successfully identify a principal component subsystem in each of these two very dissimilar problems".* It seems that although it is reasonably straightforward to identify architectures associated with Quality Attributes after a design is produced i.e. once a design is created we can determine if an item is safety critical, it is not so easy in the early stages where we are looking for direction on how to produce a safe design. Perhaps this shouldn't be too much of a surprise as Quality Attributes are often termed as 'Emergent Properties'. Alexander also goes on to say that a concept of creating a design for individual attributes will not help the designer unless it happens to correspond to the system's subsystems. He states that *"No complex adaptive system will succeed in adapting in a reasonable amount of time unless the adaptation can proceed subsystem by subsystem, each subsystem relatively independent of each other… the chances are small because the number of factors which must fall into simultaneously into place is enormous".*

Many observers suggest that the way to design for Emergent Properties is to apply certain rules or heuristics developed on the basis of experience (Rechtin, 1992). However these are often no more than broad statements that are difficult to interpret by the designer (for instance "keep it simple"). What we would hope for is a means of interpreting the structure of the design in a way that gave a view of its benefit to an emergent property. Klein et al (Klein et al., 1999) propose Attribute-Based Architectural Styles, which attempt to analyse existing designs from a preferred architectural standpoint.

It is evident that certain design strategies can lead to architectures that provide benefits from a design feature or quality attribute standpoint. There is however, not universal agreement as to what these quality attributes are. One view of this is provided by the international standard ISO25010 (ISO 25010:2011, 2011) and talks about Quality in Use and Product Quality. It is the latter that is of interest here, though the former Quality in Use will be relevant to later discussions on evaluation. Quality attributes defined are:

- Functional suitability
- Performance efficiency
- Compatibility
- Usability
- Reliability
- Security
- Maintainability
- Portability

Further classifications can be found (Chung & Leite, 2009) including those by Roman (Roman, 1985) and Boehm (Boehm, Brown, & Lipow, 1976). The author also published a work (Mackley, 2005). For a complete set of quality attributes is seems that a combination of classifications is required. In the table presented earlier, precedence could be given to the international definition of quality attributes (ISO/IEC/IEEE, 2011) in creating a combined list, but where potentially important attributes are present in other definitions, these are added (Table 4).

A drawback of any of these attribute classifications is that no relationships are shown between the attributes (Chung & Leite, 2009) and there is no obvious linkage to system architecture frameworks. This could make it difficult to identify the effect of architectural change or to identify the trade-offs or “side effects” (Bass, Klein, & Bachmann, 2002) involved in making such a change. As Alexander noted (Alexander, 1964), this can make it difficult for the systems designer to conceive solutions. Bachmann noted in his abstract (Bachmann, Bass, Klein, & Shelton, 2005):

“First there must be some way to specify quality attribute requirements so that it can be determined whether the design architecture can achieve them. Secondly, there must be some way for modularising the knowledge associated with quality attributes so that the design method does not need to know how to reason about all the multiplicity of quality attributes that exist. Finally, there must be some way

for managing the interactions among the quality attributes so that either the requirements can be satisfied or the ones that cannot be satisfied are identified."

Klein (Klein et al., 1999) reasons that adopting Attribute-Based Architectural Styles can help address the performance of system quality attributes. Chung (Chung, Gross, & Yu, 1999) and Harrison (Harrison & Avgeriou, 2007) expand on this, but there is not agreement in the methods, with Chung showing links of Architectural Styles to attributes such as Modifiability, Interactivity, Reusability and Performance, and Harrison with a different set of attributes including Usability, Security, Maintainability, Efficiency, Reliability, Portability and Implementability. Klein identifies quality attribute measures for performance, which is arguably the most direct non-functional requirement to defining the functional behaviour of a system. This work identifies *latency*, *throughput*, *nature of arrival of stimuli* and *resources required* as being key parameters to achievement of performance, suggesting that an architecture should take this into account.

Table 4: A definition of system quality attributes

ISO 25010 (ISO/IEC, 2011)	**Roman** (Roman, 1985)	**Boehm** (Boehm, 1978)	**Mackley** (Mackley, 2005)	**Combined List**
Functional suitability		Functionality	Effect	Functional suitability
Performance efficiency	Performance requirements	Performance		Performance efficiency
Compatibility		Compatibility	Compatibility	Compatibility
Usability	Operating requirements	Usability	Operability	Usability
Reliability	Reliability	Reliability	Reliability	Reliability
Security	Security	Security		Security
Maintainability	Maintainability	Maintainability	Maintainability	Maintainability
Portability	Portability	Portability		Portability
		Supportability	Supportability	Supportability
	Survivability		Survivability	Survivability
Interoperability			Interoperability	Interoperability
Availability	Availability		Availability	Availability
Adaptability	Enhanceability	Adaptability	Adaptability	Adaptability
		Predictability	Predictability	Predictability
			Producibility	Producibility
			Safety	Safety
	Economic and political requirements		Acceptability (PESTLE)	Acceptability (PESTLE)
		Serviceability	Serviceability	Serviceability

2.4.4 Observations on architecture principles

There are three basic principles for architecting (Fricke & Schulz, 2005). These are:

- Simplicity

- Independence
- Modularity.

They are seen as “fundamental” as they offer the benefits of using order to tackle the challenges of complexity and complication. They are not a panacea in the sense that they can be used to help solve every problem – for instance a solution can only be made simpler if it is still able to meet its purpose. They also require means of application that focus on improving desired outcomes.

Intuitively, appropriate *simplifying* of the design should reduce the difficulty of the design and development process. Closely related concepts of simplicity, streamlining, ideality and Suh’s information axiom, aim to provide guidance on how this can be achieved, but there is little help available to provide objective support to the system designer. Suh’s information content parameter and Altshuller’s Ideality calculation (Altshuller, 2002) are two candidate measures, and general heuristics, such as those of Maeda (Maeda, 2006) may be useful in conceiving ways of achieving simplicity.

Independence is a specific means of decoupling functional elements of the design, as they are represented in the physical design. The potential advantage in terms of analysis of design development and modification is then obvious, but Simon and Orton counsel that total independence is effectively decoupling the design and this can exclude benefits of synergy that loose coupling can provide. Weick and Klien suggest characteristics of the interface that are desirable for loose coupling.

Modularity attempts to create modules in the design that reduce the degree of interfaces or coupling between those modules. This makes modules easier to incorporate or remove/replace which provides benefits in design, production and use. It is a principle that is complemented by independence as both seek to reduce interaction between elements of the design. The degree to which modularity is desirable will depend upon the context and this needs to be assessed on a case by case basis.

Notably, the literature on system architecting focuses on creating an ordered structure and simplifying where possible. Alexander and Simon convincingly argue that architectural strategies that aim to decouple the design are essential. However, a grossly simplified solution is unlikely to meet a complex need and, as Orton observes, a completely modular design to the extent that it is uncoupled is no longer a system and so it is unlikely to benefit from any resulting synergies. For instance, a strategy of functional independence is not necessarily going to align with an architecture designed from a safety or maintenance perspective and even if this was possible, in Alexander’s terms this will still not help unless the

designer can see the implication that his design decisions have on the high level systems attributes; Sako believes this unlikely (Sako, 2003). This is supported by Bachman who observes “there must be some way for managing the interactions among the quality attributes so that either the requirements can be satisfied or the ones that cannot be satisfied are identified.” Architectural perspectives are found to offer a way of viewing and designing the system in different ways according to the needs of different quality attributes. A difficulty here is that there is not a universal agreement on what the complete list of quality attributes is. However, a combination can be made from the various taxonomies. The abstract nature of Architectural Perspectives might be a source of concern for the designer, but in practice system design practitioners are already comfortable with the concept of the abstract functional architecture. More of an issue might be Alexander’s argument suggesting that a concept of different architectures for each attribute should be rejected if it doesn’t align closely with the physical design. However, Alexander states this in the belief that overlapping architectures cannot help the designer to make rational decisions; if it can be shown that the designer can act upon the analysis with a clear view of how it will improve the design, this can be acceptable.

Architectural decisions (Tyree & Akerman, 2005) reflects on the fact that not all decisions of a design should be considered as architectural; some interactions should be considered as more important than others from an architectural point of view and as long as a joint architecture respects these then that should be considered a positive design. There is a need to identify where functions can easily be partitioned needs to be determined. It is proposed to address the latter point by the characterising functions and their implied interfaces using a concept called ‘*Functional interaction types*’. Langlois (Langlois, 2002) commented that:

> “In a world of change, modularity is generally worth the costs. The real issue is normally not whether to be modular, but *how* to be modular” and “how do we find the ‘natural’ encapsulation boundaries?”

This is a fundamental question that needs to be addressed; how to produce a modular design, ensuring that it uses ‘natural’ boundaries to encapsulate the design in the most appropriate way.

Finally, the principles of Fricke and Shultz refer to the system product, but other work (Gershenson & Prasad, 1997), Gu (Gu & Sosale, 1999) identifies further Lifecycle Modularity principles of:

- Attribute independence
- Process independence
- Process similarity.

There is overlap between these principles and if these are to be properly understood and applied, then a common terminology should be sought; this will be addressed.

2.5 Methods for System Architecting

There are few methods that have been developed to aid with the process of system architecting with many systems engineering methodologies treating it as a creative activity, based on experience. An example is Total Design (Pugh, 1991) where the mark of a good architecture is established by evaluating the system outcome and comparing it with either a requirement or other possible solutions; the result doesn't demonstrate a good architecture, but only that the design (and by inference, the architecture) is good enough or better than the others proposed. Whilst this might be satisfactory in some situations it can be wasteful in that it requires a number of concepts to demonstrate and only when a solution is developed can there be confidence in the assessment. The method and as it doesn't incorporate means to actively optimise the design its product may not be competitive against those that did.

Two methods that look at improving the architecture by design are Axiomatic design and Design Structure Matrices.

2.5.1 Axiomatic Design (Suh)

Suh presents a theory for systems design that applies to systems in general, be they machines, software, large systems or organisations (Suh, 1998). Two axioms are presented that are used to create a top down design.

"Axiom 1: The Independence Axiom: Maintain the independence of the Functional Requirements (FRs).

Axiom 2: The Information Axiom: Minimise the information context of the design."

These axioms are used to produce an architecture composed of three hierarchies; which in the case of systems represents (Suh, 1995):

- Functional Requirement – function requirements of the system
- Design Parameters – machines, components, subcomponents
- Process Variables – resources (human, financial, materials etc.)

The method is prescriptive, describing rules that must be followed in the determination of a "best" design. In specifying a prescriptive approach the author achieves clarity in application, but in the process of achieving an optimal solution there is a significant amount of calculation based on probabilities of satisfying requirements, which would be difficult to evaluate and validate (Frey & Dym,

2006). In terms of validation, Suh offers his method as a set of axioms and corollaries and therefore claims that “there are general principles or self-evident truths that cannot be derived or proven to be true, but for which there are no counterexamples or exceptions” (Suh, 2001). In developing the system design Suh uses three types of interface or “junctions”. These are Control (which is in the sense of command or demand), Summation and Feedback. The latter represents an unacceptable situation as feedback between modules violates the Independence axiom and therefore is not permitted.

By the author’s own admission the method has certain issues in the design of systems, largely down to its strict adherence to its axioms. These are issues with:

- Addressing large and flexible systems
- Situations where reuse is expected,
- Unstable systems (where stability is termed as not being able to meet the independence axiom)
- Human interaction, which can introduce unpredictable effects outside the design analysis

A further issue is that, in pursuit of the “best” design, the method does not address the potential conflicting demands of quality attributes e.g. is a highly safe, modestly performing solution better than a highly performing moderately safe solution? By its definition only one “best” solution can exist, which in the reality of systems evaluation is not the case (as discussed in section 2.7.2). The issue of quality is addressed by Suh (Suh, 1995), but this is in the narrow confines of a predictable design that is robust and contains redundancy.

According to Orton and Weick a system that is decoupled is no longer a system, but rather a collection of independent items (Orton & Weick, 1990). However, a “loosely” coupled system that approximates to and behaves as if it were an independent system is appropriate. Therefore a definition of what is an acceptable level of independence is required. Suh terms this as a designer specified tolerance, but there is no guidance on how this can be defined.

2.5.2 Design Structure Matrices

Tools have been developed that examine the elements of a product or organisation and group them in terms of the degree of coupling of the mutual interactions. Steward (Steward, 1981) was perhaps one of the first proponents of the Design Structure Matrix which takes a mathematical approach to the determination of a modular structure, based on the number of interactions between the elements. This might be reasonably straightforward if all interfaces are equal, but this is rarely the case for real systems. Others (Steven D Eppinger

& Pimmler, 1994) and (Sosa, 2003) propose that, for a physical system, the strength of interaction should depend on whether there are significant interactions of the following certain types (i.e. energy, material, information, spatial, structural) and that values can be assigned to reflect this importance. Sharman takes this further, and uses the significance of the interaction in each case to derive an overall value of strength of dependency. Yassine et al (Yassine, Falkenburg, & Chelst, 1999) investigate various methods that have been devised based upon a single dependency measure:

"Steward (Steward, 1981) discussed the use of numbers instead of marks in the DSM to represent the difficulty level of using an estimate. Smith and Eppinger (Smith & Eppinger, 1997) extended the basic representation of a DSM to accommodate numerical values that reflected the difficulty of performing tasks in the absence of predecessor information. Krishnan et al. (Krishnan, Eppinger, & Whitney, 1991) introduced the notion of a quality loss function to capture the decrease in quality of task results due to constraints imposed by a certain sequence of tasks. Sobieszczanski-Sobieski (Sobieszczanski-Sobieski, 1988) devised an algorithm that calculates the system sensitivity derivatives from a set of equations derived from the implicit function theorem. In a more recent paper, Smith and Eppinger (Smith & Eppinger, 1997) used methods of feedback control theory to analyse and identify controlling features of the iteration process. The method called for the determination of the eigenvalues and eigenvectors for the Work Transformation Matrix (WTM) which is a DSM containing the strength of dependency between the tasks."

These authors then propose that dependencies are better represented by a two dimensional variable representing the sensitivity and variability of the interface. However, certain difficulties appear to remain: firstly that assignment of values appears to be subjective and secondly that the distinguishing characteristics have widely differing properties in different units that cannot be obviously combined. Further difficulties are in developing an automated algorithm (Sharman 2002), but this also reflects the difficulty encountered in representing complex relationships on a two dimensional matrix structure. Attempts to progress have focussed upon geometries within the matrix, but these geometries do not generally reflect the nature of the interfaces which is likely to be the most important aspect.

2.5.3 Literature update

To ensure the currency of this research, a further search of the electronic databases at Cranfield University was performed subsequent to viva. This was performed on August 2nd 2018 using the terms "system", "design", "modular" and "architecture". This yielded 495 results of which 244 were from 2010 to the present. This yielded four relevant publications. Three of these related to methods

to create a modular system design based upon weighted techniques using DSM or Quality Function Deployment methods and so should be seen as supplementing methods already reviewed (Bayrak, Collopy, Papalambros, & Epureanu, 2018; Francalanza, Mercieca, & Fenech, 2018; Wong, Qaisar, & Ryan, 2016). Sillitto's book has an interesting and detailed view of the concepts, principles and practices surrounding the architecting of systems, which again refers to the use of N^2 matrices (a method similar to that of DSM) as the main tool for clustering and choosing the architecture (Sillitto, 2014).

2.6 The impact of context on systems design

2.6.1 Systems design in context

The context of a system arguably determines its requirement; its needs, its constraints, its conditions, its influences and its lifecycle. The requirements will affect the system design in the way that these requirements can be partitioned and allocated within the system and the system architecture in terms of the boundaries and corresponding interfaces that are chosen for the elements of that system. There are a number of concepts that can be found in the literature that can help to describe the influential aspects of the problem context.

A good place to start is with Problem Types (Obeng, 1995), where Obeng identifies four problem types depending on the uncertainty in both objective and solution. His contention is that in managing projects there is a temptation to address them as if timescale, cost and performance can be accurately determined in advance. In reality, if there is uncertainty in the objective and/or solution, the approach needs to adapt and reflect that estimating these project management parameters is not obvious. Obeng's work provides impetus for questioning whether all problems are of the same type; are there other potential influences on the approach that should be taken other than uncertainty in objective and solution.

The starting point of the system design is also important in determining the approach. Formal, structured design techniques often start from a "clean sheet", but Jackson argues that *real* engineers don't start from clean sheets, with designs evolving conservatively over many generations*; "Only once in a thousand car designs does the designer depart from the accepted structures by an innovation like front-wheel drive or a transversely positioned engine."* Jackson points out that this diverts attention away from "large structural" (or architectural) issues and leads him to suggest that more effort should be placed into developing and using lower level formal methods rather than methods for dealing with structural design (Jackson, 1998). Jackson's view is reinforced by others with Cambridge academics Jarrett, Clarkson and Ekhert (Jarratt, Eckert, Caldwell, & Clarkson, 2011) quoting (Cross & Roy, 1989):

> *"…most designing is actually a variation from or modification to an already-existing product or machine."*

and (Bucciarelli, 1994).

> *"History matters – no design begins with an absolutely clean sheet of paper."*

A 'clean sheet' approach can be applicable for a number of reasons. Whilst Jackson argues that a clean sheet approach is rare in traditional engineering, it is an approach that can be justified in situations where there is rapid growth in technology and a design can take little benefit from existing designs where the components and techniques are obsolete – this is often the case for computer design (Bell, 1991). In considering flight path design for aircraft, a clean sheet design allows the designer to get away from a design that has evolved to meet the need at specific times and does not therefore necessarily address the need in an optimal way (Conker, Moch-Mooney, Niedringhaus, & Simmons, 2007). The need to sometimes remove measures that have been designed as evolutions to a system in order to make a fresh start is reinforced by Wolstenholme:

> *"to get the best out of systemic policies it is necessary first to remove institutionalised, emergency coping mechanisms (fixes), created because of time delays and difficulties in cross boundary working" (Wolstenholme, 2004b).*

Examination of system development in the military domain shows that systems are frequently developed from near clean sheets. This may be because of an imperative to obtain a capability that is superior to a would-be adversary, or because systems are designed and operated for long life-times and replacement of an obsolete system requires a complete rethink.

Problems addressed by Soft Systems Analysis seem to be of an entirely different sort. Here the starting point is one of a feeling of unrest or dissatisfaction with the status quo. Instead of addressing a clear capability gap or need the problem is to address an unsatisfactory situation. In the early 1960s Jay Forrester applied the concept of System Dynamics to industrial organisation (Forrester & Cambridge, 1961). This provided the basis for work (P. Senge, 1990), which used System Archetypes to describe the dynamics of business situations, by analysing a business as a system. The basic premise was that dynamic situations could be described in terms of reinforcing and balancing loops of cause and effect; archetypes were able to replicate the behaviour of ailing organisations and thus a diagnosis of the reasons for the problem can be achieved. The original set of around ten systems archetypes can be expressed as a reduced set of four archetypes (Wolstenholme, 2003), which themselves can be expressed as two

generic archetypes, the “Underachievement Archetype” and the “Out of Control Archetype” (Wolstenholme, 2004b). Wolstenholme maintains that when faced with underachievement problem archetype the solution requires remedial activity outside of the system boundary, but when faced with the out-of-control archetype problem, the expedient action is to find a fix within the system sphere of control (i.e. within its boundaries). Responding to a problem situation either by an internal modification to the system or by requiring additional activity externally would seem to cover all cases, but System Dynamics relies on the problem being predictable in order to correctly analyse it.

In the Early 1970s there was a change in the approach to Systems Thinking. Midgley (Midgley, 2006) described this as a “second wave” in Systems Thinking, which criticised the previous attempts to model the system as if it were deterministic and not reflecting the needs and views of involved stakeholders. The second wave emphasised dialogue, mutual appreciation, subjective understandings and accommodation between perspectives. Key protagonists were Churchman (Churchman, 1968) and Checkland (Checkland, 1981a). This is an important move architecturally, because it emphasises the point that some interactions within the system are not “hard” and deterministic, but “soft” and subjective. These sorts of influences on interactions and interfaces within a system are likely to introduce additional factors that need to be considered in devising an architecture. In his account Midgely also recognises a “third wave” of Systems Thinking; Critical Systems Thinking. In particular this maintained that not everyone within a system was at liberty to participate within the system according to their will. Jackson and Keyes (Jackson & Keys, 1984), (Jackson, 1994) expressed the difference by categorising Systems thinking as unitary, Pluralist and Coercive.

Other types of situation that have attracted significant attention in the systems field are those of varying complexity and different levels of control and ownership. Both of these relate as much to type of solution as to the type of problem. Peter Senge (P. Senge, 1990) distinguishes between modular and integrated solutions, defining them as cases of detailed and dynamic complexity. This view of complexity has subsequently been updated by Snowden (Snowden & Boone, 2007).

Maier recognised that the way a system can and should be designed will depend on ownership of the elements of the system in both development and operation. Where systems are independent in both of these cases it can be termed a system of systems (Maier, 1998) and in such cases relationships are defined by service agreements and standardised interfaces.

2.6.2 Observations on systems context

Analysis of the context is key to identify the need that a system is expected to address. System design is responsible for partitioning the requirements to the components or subsystems of the design, but to do this a clear view is needed of both what is needed and what a system design needs to address.

In this section it has been shown that a number of eminent systems thinkers have identified that there are different types of problem and that these need to be addressed in different ways. Some existing 'types' of problem have a bearing on the stakeholders, their needs and the way that a system should be organised. According to Midgely (Midgely & In, 2006), there are different "waves" of systems thinking, each applying to different types of situation; system dynamics, soft system methodologies and critical systems thinking. From this example we can see that different situations or context require different approaches. Identification of common factors that influence the choice of approach would help suggest how these different practices may be unified into a common approach. Such an approach should first identifies the type of problem to be addressed from an examination of the context, and then use this type to direct the line of systems activity and the requirements of an architecture. This research will develop the concept of "context type", where different aspects of the context will require different approaches to and measures within the design.

2.7 System and architecture evaluation

In proposing a process for developing an architecture, it will be necessary to evaluate the systems architecture and the systems design that is produced; that is evaluating the "goodness" of the architecture, and how the system design approach incorporates evaluation of the designed end product.

2.7.1 Evaluation of Architecture

2.7.1.1 Visualising and quantifying hierarchy and architecture

Early techniques to illustrate architecture in a graphical format were developed by Warfield as binary matrices (Warfield, 1973). The N^2 was then developed by Robert Lano to provide a mathematical technique that could be automated. Hitchins describes a method by which a single parameter score can be provided, which represents the distance of interfaces from the leading diagonal of the N^2 matrix (Hitchins, 1992). This provides a simple if crude indication of the "goodness" of the architecture.

It is possible to assign different values to the matrix and Steward (Steward, 1981) considered this for the similar Design Structure Matrix (DSM). Yassine (Yassine et al., 1999) advocates expert assigned values for sensitivity and variability to

clarify the strength of coupling and dependency between two elements in the matrix, but this is meant as an aid to a more definite clustering of the resulting DSM rather than a better evaluation.

2.7.1.2 Dependency/Visibility Ratio

Sharman (Sharman, 2004) develops two important aspects of modularity which are directly obtained from Design Structure Matrices. Visibility is the degree to which a component is visible to the rest of the system and Dependency is the degree to which it is dependent on other parts of the system; the lower the visibility and dependency are the more modular the component is.

2.7.1.3 Types of Design Dependency

Sosa, Pimmler and Eppinger suggest five types of design dependency that feature in systems, which are the Spatial type and four further Transfer types comprising Structural, Energy, Material and Information (Sosa, 2003), (Pimmler, 1994). These are used to help determine the modularity or otherwise of a design. Independent experts are asked to score the criticality of the interface on a five point scale ranging from -2 to +2 for each category; a negative score showing an undesirable dependency. This technique of scoring, albeit based on expert opinion is very subjective and when subjective parameters are combined then arguably the experience of the experts can no longer be used to validate the output. This is a common issue of the subjective nature of weighting techniques that will be returned to in the next section on evaluation.

Such an assessment is then used to distinguish a modular architecture from an integral architecture by performing a Chi square procedure to compare statistical similarity with a known integral or modular structure, there is no explicit guidance on what represents a model example of modular or integral structure, but in their research based upon amount of external interfaces.

2.7.1.4 Components of Modularity

The same authors propose an alternative view of modularity of an architecture by taking a network approach (Sosa, Eppinger, & Rowles, 2007). They propose that Component modularity equals Actual component disconnectivity minus Maximum possible component disconnectivity. They propose three components of modularity as:

- Degree Modularity – a normalised value that relates to the number of dependencies in and out that the component has i.e. the larger the number of components that affect or are affected by the design, the less modular it is

- Distance Modularity – a normalised value that relates to the distance from other components i.e. the more distant the more modular
- Bridge Modularity - again a normalised value, but that relates to the number of times a component lies on the path between two other components i.e. the more design dependencies that propagate through them, the less modular the component.

Suh's axioms provide another means of evaluation in terms of both the correct coupling criteria for the Independence Axiom (i.e. is it uncoupled or appropriately decoupled) and level of information obtained using the Information Axiom. TRIZ (Altshuller, 2002) offers the further metric of Ideality to ensure that unnecessary functionality is incorporated in the design. Only the Information content from Suh's axiom attempts to express goodness of design resulting from the architecture, although this is often requires considerable mathematical calculation as is based on estimate of probability that may be difficult to support. The other measures attempt to define prerequisites for an architecture to support a good design, but not to evaluate whether this has been achieved. In the case of the Information Axiom, this evaluation is made for functional requirements only and there is no consideration of the non-functional requirements. Architectural evaluation is therefore often made on subjective expert opinion and a combination of factors that cannot be validated.

A further method is also proposed for the evaluation of module concepts from a lifecycle context (Ericsson & Erixon, 1999). Only aspects that can benefit a single system product (rather than gaining advantages due to a family of variants) are considered and these are:

- Lead time in development is reduced by less complex interfaces
- Assembly times are reduced for modular designs
- Quality is increased by number of parts separately tested and is inversely dependent on assembly time
- Recyclability is improved by reducing the number of materials
- Interchangeability is inversely dependent on number of functional interconnections

Key complicating parameters emerging from this are assembly time, interface complexity (number and complexity of interfaces), for which strategies might be reducing number of different modules, reducing number of and simplifying of interfaces.

Studies into design for assembly have shown that defect rate increases significantly with the assembly time. It is reasonable to assume that longer

assembly times are associated with more integrated designs and more complex interfaces. Whilst it is not possible to assess the expected assembly times in early concept design, it is reasonable to assume that an increase in modularity would in general reduce assembly times and therefore reduce the level of defects (Barkan & Hinckley, 1994).

2.7.2 Evaluation of systems

A system design is difficult to evaluate as systems engineering deals with multi-criteria problems seeking to take explicit account of multiple, conflicting criteria. Multi Criteria Decision Analyses come in three technique categories (Belton & Stewart, 2002):

- Value Measurement models
- Goal Level models
- Outranking models

Value measurement assigns values or weighting to criteria and a preference for a concept is formed from an aggregation of the values. It relies on a strong set of axioms to form a preference in order to impose “some form of discipline” in the building of preference models, help decision makers to understand their values and be able to justify final decisions and to include statements of acceptable trade-offs between criteria (Mendoza & Martins, 2006). However, for independent criteria, relative value will always be subjective and (Arrow, 1951), “there is no method of aggregating individual preferences over three or more alternatives that would satisfy several conditions for fairness and always produce a logical result”.

Goal oriented methods are based on being able define outcome scenarios and requires the designer to specify goals for each criterion. To be able to achieve this, the designer often requires deeper understanding of the solution domain to understand trade-offs, which he typically achieves through past experience or feasibility studies. These requirements enable a systematic elimination of alternatives to leave only compliant solutions and there is an overriding principle of “satisfying” rather than optimising, allowing a down-selection of alternatives.

Outranking is a method that relies on pairwise comparisons and perhaps the most frequently use method is the concordance-discordance principle (Belton and Stewart, 2006). This declares that an alternative x is at least as good as an alternative y if:

- a majority of the attributes supports this assertion (concordance condition) and if

- the opposition of the other attributes (the minority) is not too “strong“ (non-discordance condition)

It is essentially a voting technique and a weakness compared to Value techniques is that this principle can allow contradictions that need addressing. For instance, it is possible that there is opposition to x being better than y at the same time as opposition of y being better than x, or that x is better than y is better than z, but that x is not better than z.

2.7.3 Observations on evaluation of system and architecture

Evaluating a system is a multi-criteria problem and Arrow’s impossibility theorem states that there can be no ‘best’ solution. It is difficult to evaluate a design confidently at the concept stage, but by selecting an appropriate architecture it is possible to lay down a favourable structure for building the design.

There are methods available that address the modularity of an architecture, but universally these methods are unable to judge the complication of interactions within the architecture and therefore it is difficult to evaluate whether a particular modular architecture is addressing complication as well as it could. A means of identifying the complication of interactions is required so that this can be fed back into the overall assessment of architecture.

Whilst it is not possible to produce a detailed evaluation of a system from its architecture, there is evidence to suggest that certain designs of architecture will promote certain quality attributes and so such traits, if identified, could also help in assessing the quality of the architecture.

3 RESEARCH QUESTIONS, METHODOLOGY AND APPROACH

The literature search has demonstrated that a system architecture is developed from the system design process and is a key element in terms of managing the behaviour and quality of a system. Surprisingly, there are few methodologies available to develop a system architecture and of these, no methods have achieved general acceptance in industry. It is this gap in knowledge that this research is designed to address. A methodology will be developed for the system design process that can apply modular architecture concepts in order to positively contribute to the effectiveness of the final design.

System design must address systems of many types, having many diverse purposes, of many different sizes and organised and behaving in many different ways. This variety provides a significant challenge to any generic systems process and, just as with any systems problem, a boundary and scope need to be defined for this research. This research will concentrate on the process of system design as applied in the generation of concepts, rather than the process through the complete systems engineering lifecycle. The concept phase is arguably the most critical part of the engineering lifecycle, as decisions made here will determine the majority of the lifecycle costs of the future solution (Ehrlenspiel et al., 2007). The prime focus will be on the architectural considerations of systems design process i.e. the organisation of subsystems, their boundaries and interactions. Other parts of the system design process, such as requirements capture and evaluation are addressed, but primarily to allow generation of case examples and for their evaluation.

The approach to systems design in this research is based on a premise that a system architecture, at a concept stage, can have a positive influence on the quality and behaviour of the final design. In particular, a modular architecture can reduce the complication of the design process in order to reduce the risks involved in system development. The aim will be to produce a compliant concept rather than seeking to optimise one or more performance parameters. However, a practitioner could use solutions generated to select an ‘optimum’ design. As a result, it is not guaranteed that this approach will come up with a “best” solution, though it is doubtful that a single best solution ever exists to a systems problem (section 2.7.2). The original concept of the research was that it might be possible to consider a system as a set of abstract design architectures each favouring a given quality attribute; choosing a solution that allows these architectures to align could simplify the system design problem. In reality, directly relating the features of a design to quality attributes, such as safety, in a coherent architectural strategy is unlikely to be achievable (Alexander, 1964). Instead the research proposes a set of fundamental functional elements or *functional blocks* that can

be managed by the designer to achieve the desired outcome and it is in maintaining the integrity of these *functional blocks* that an appropriate system design will emerge.

For this thesis therefore, the hypothesis is that there is a means of positively influencing quality attributes and development risk associated with a design through the application of modular architectural principles. This research will therefore address the question:

"How can modular architectural principles be applied to early system concept design to manage system effectiveness and reduce lifecycle risk?"

Ideally it would be possible to demonstrate the efficacy of the proposed methodology by applying it to a problem and demonstrating that the developed methodology performed better than other available methodologies. In reality this approach has a number of issues:

- Case studies of complicated systems design are complicated in themselves, often requiring multiple teams and many thousands of man-hours of effort to address the design at a level to determine the attributes of the design. Therefore, performing such case study is not feasible in the context of this doctoral study
- An individual system problem cannot be shown to be representative of all problems and therefore numerous studies would be required
- Evidence would be needed that the design developed using this methodology is better than would have been provided with existing methods. This would require the detailed study of designs derived using other methodologies for comparison. Furthermore, objective evidence would be required that one system design was indeed better than another, but the nature of multi-criteria problems is that comparisons are always going to be subjective and therefore difficult to validate (Arrow, 1951).

There is also an issue of scope; within a PhD study there is a limited time available. Even without developing a methodology from first principles, the application and assessment of such a methodology might form a research project in itself. Therefore the methodology of this research will rely on:

- Employing principles that can be shown, by induction, to benefit from well-established modular principles
- Illustrating that the approach can be applied to a variety of design concepts to verify the application of the process to real-life situations
- Comparing the utility and performance of the approach with similar and currently available methods.

For the latter, the developed system design methodology will be compared against two currently established methods applied to the same problem. For this comparison, the design of a central heating system has been chosen as it both relatively simple to facilitate the comparison, and of sufficient complication in terms of control, service and human factors to exercise important elements needed in a method.

In developing a methodology a research design should address certain elements (Zehra, 2015). Accordingly aspects of study design, study population, data collection and variables considered are addressed below:

a) Study design: combines observation in terms of identifying relevant architecture principles from current literature, with analytical activity of how these can be applied depending on the problem and the desired outcome. Finally, the utility of the approach will be demonstrated by case study and in comparison with other methodologies.
b) Study population: in this research the study population is composed of the three different problem situations to which the proposed methodology can be applied and two other methodologies that this methodology can be compared against.
c) How will population be identified?: the case studies have been chosen to represent different levels of complication, whilst accepting that their scope is limited by the time available in a doctoral study. Methodologies have been selected on having the same objectives as the proposed methodology and according to their level of acceptance in the systems engineering community.
d) What data will be collected?: data will be in the form of qualitative evidence that relevant quality attributes are being addressed by proposed designs.
e) Variables: the independent variables of this research are the methodologies used for comparison; the dependent or effect variables are those of system effectiveness (or quality attributes) and lifecycle risk

Literature searches have been performed on systems design and architecture. This has been broadened to include an analysis of literature about the system context, as the contextual setting of a system problem is key to understanding the needs of the design. There are many existing methodologies, but no particular methodology has been adopted widely within industry or academia (refer to table). Tomiyama et al (Tomiyama et al., 2009), identify the most prominent ones as:

- Systematic design (SAPD) by Pahl and Beitz (Pahl et al., 2007) as representative of design focused methods (Roozenburg & Cross, 1991)

- Axiomatic design by Suh (Suh, 1997) as representing of attribute focused design
- TRIZ by Altshuller due to its relative popularity in industry
- Product design by Ulrich and Eppinger (K. Ulrich & Eppinger, 2008) due to its popularity in the US
- Total design by Pugh due to its higher than average use in industry.

Uhlmann (Ullman, 2003) has not been considered here as it is essentially a systematic design method along the lines of that of Pahl and Beitz. Within the scope of the title of this thesis, a methodology should be expected to:

a. Address system design within a lifecycle context – that is the identification of both functional and non-functional requirements and ensuring appropriate systems architecture to address them in a cost effective and manageable way, considering the whole lifecycle. Activity models focus on a particular stage and can be incorporated within a framework provided by the stage model as long as it considers the implications of other stages of the lifecycle.

b. Apply to different starting points of the process – not all problems start from a clean sheet of paper and a methodology will need to be able to work within the constraints of established designs and context.

c. Be prescriptive – if such a process is to be adopted, it needs to be a clear and applicable process that does not require tacit knowledge. Descriptive models can identify good practice, but rely on the engineer to extract analogous concepts and apply them appropriately. If this method is to be widely adopted such capability should not be assumed.

d. Incorporate method, tool and notation support where possible – adoption of the process will require it to be a clear and efficient way of developing and communicating a solution. If this requires support methods, notations and tools, then these need to be defined or at least conceptualised in a way that assures that they are realistically achievable.

e. Focus on both actively improving design and reaching an assured level – in addition to an assurance that the design will meet its requirements, an engineer also requires that any given design intervention will achieve the desired improvements. Stage models focus on confirming that the necessary assurance is in place, but activity models tend to provide the measures that can be employed to ensure an improved design.

Therefore against the classification method developed in

Table 2 of section 2.3, the criteria that should be expected to be met are given in Table 5. This will be returned to in making a comparison of methodologies in section 9.6.

Table 5: Required characteristics of system design methodologies

	Required characteristics of methodology
Scope	Concept phase
Starting point	Both problem and solution based starting points
Approach	Concrete, prescriptive (procedural and analytical)
Models	Either activity or stage based
Aim	Design improvement
Support (to concept design):	**Desirable support**
Methods (relevant to concept stage)	Yes
Means	Yes
Notation	Yes

4 THE SYSTEM DESIGN IN CONTEXT

Characterising the context is important in identifying the required engineering activities including the architectural strategy. System design is about taking a problem or need and creating a system to satisfy it, where the “problem space” is largely determined by the system context of stakeholders, related systems and environmental conditions. Obeng suggests that there can be a tendency to address problems in the same way (Obeng, 1995), but that an approach should be tailored to the problem to be solved or the solution to be engineered. It is clear from the literature that there are a wide variety of problems that need to be addressed and this will require a systems design approach will that can to be tailored to this variety. To achieve an understanding of how to approach this, we should have a concept of:

- A framework which identifies and orders the context of a problem and its solution
- A notation to capture the contextual situation for the designer to respond to
- An approach that allows architectural strategies to be applied according to type of problem or context to be dealt with

4.1 Frameworks and Notations

All systems are open and therefore at any level of system design there needs to be consideration of how the system interacts with its higher level system or system of systems and therefore a need for a framework that provides this information (this may be an evolving context that is influenced by other developing solutions). Various ‘architecture frameworks’ have been developed and one such example is the Enterprise Architecture Framework, which is a framework that “can describe the underlying infrastructure, thus provide the groundwork for the hardware, software and networks to work together” (Urbaczewski & Mrdalj, 2006). There are a variety of frameworks that have been developed for different domains and attempts have been made to compare them to distil similarities in some of the most widely recognised examples (Urbaczewski & Mrdalj, 2006).

At the highest level they each have a definition of views (or perspectives) that represent the stakeholders of the system and also a definition of required abstraction. Urbaczewski attempts to make a mapping between the views and abstraction for each architectural framework, but concludes that such a mapping cannot be confidently made. Despite the commonalities claimed by systems theory (Von Bertalanffy, 1950), architectural frameworks have been developed to reflect the immediate and practical realities of a given domain.

An architectural framework is intended to structure information pertinent to a system development in a way that facilitates its design and management. As with any architecture it should indicate key elements of information and how they relate to each other. In terms of a system, key elements are the *capability* that the system is intended to contribute to, the definition of the *problem* to be solved, the *solution* proposed and the *lifecycle*. The as-is *capability* can be used to define a *capability gap* that is then used to frames a new *problem* to be solved and system *solution* is the proposed to the *problem*. Creating or developing the solution may have *impact* which modify the problem situation, which requires the *problem* and *solution* to be considered together. Once decided upon, the *solution* will be realised and operated over a *lifecycle*, with progressing *maturity*. At suitable points in its lifecycle the enterprise will be at a suitable state of *readiness* to then benefit from the improvement in *capability*. This is represented pictorially in Figure 3.

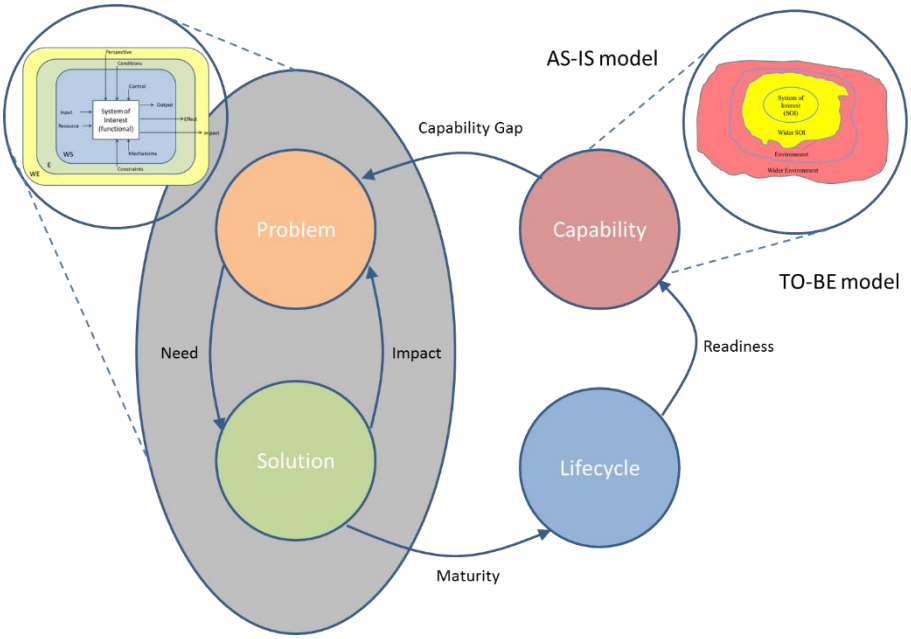


Figure 3: Proposed schematic of a generic architectural framework

Key points from this model are:

- The capability environment may itself be a system of systems and therefore, in order to maintain a clear baseline an attempt needs to be made to separate the need or *capability gap* from the system requirement
- As the conception of a system solution may itself impact upon the problem space, these need to be developed together
- The information about the system will evolve and need managing through the lifecycle, including contributing to design assurance

- In maturing, the system will go through a lifecycle and at various stages it will have different degrees of readiness (e.g. initial operating capability, final operating capability, mid-life update)

Flood and Carson (Flood & Carson, 1983) provide a convenient concept for helping to define the context of a system which can be used to define the current capability and identify capability gaps (Figure 4). Influences and constraints on the system of interest (SOI) come from:

- Wider System of Interest (WSOI)
    - Elements that interact or co-operate with the SOI and that are essential for the operation/ support of the SOI
- Environment
    - Elements that involve direct non-cooperative action or influence on or by the SOI
- Wider Environment
    - Elements with an indirect influence on the SOI through the environment

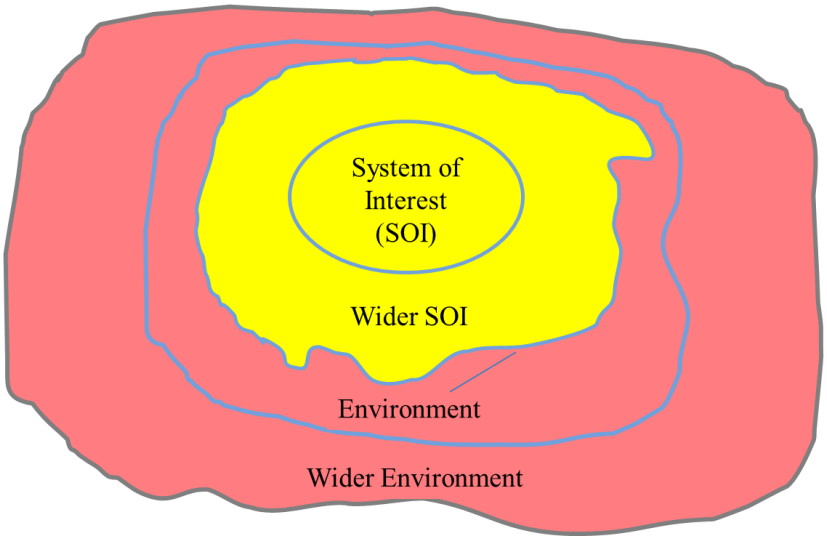


Figure 4: System context diagram (Flood and Carson)

This is convenient for a single system within the context, but in the case of a systems of systems, there may be two or more concurrently running problems and their solutions within a systems of systems would potentially have the same wider system or environment. This might involve a shared service whose capacity would be jointly influenced, or they might impact the same environment and a joint assessment of the consequence would then be needed. These potential issues will need to be recognised as potential evolutions of the capability gap defined for each individual system.

In order to elicit the requirements of the context, stakeholders can be idenitifed from Checkland's CATWOE acronym; Client, Actor, Transformation, Weltenshauung, Owner and Environment (Checkland, 1981b). Weltenshauung stakeholders can be elaborated by the PESTLE acronym for Political, Economic, Social, Technological, Legal and Environmental (Johnson, Scholes, & Whittington, 2008). Further stakeholders, from a capability provision perspective, can be derived from the UK MoD acronym TEPIDOIL; Training, Equipment, Personnel, Information, Doctrine, Organisation, Infrastructure, Logistics.

This leads to three interconnected, but loosely coupled views, with a one to one mapping with the required abstractions Table 6.

Table 6: Generic architectural framework views and abstractions

View	Abstraction	Perspective	Interaction issues
Capability provision	As-is capability and capability gap	User/ Client	Other systems will share services and have impact
Systems design	Problem statement, solution design and its maturity	System Designer	Requirement and solution will requiring validation against the capability gap
System lifecycle management	Lifecycle of the system solution and its readiness	Owner	Assessment of maturity will determine readiness

This research will focus on the system design only and it will involve itself with an identification of a capability gap and address an early maturity of system solution consistent with a concept architecture design. It will therefore focus on a specific definition of the context. However, in doing so it should address the full context (associated with a complete set of stakeholders across the layers of Flood and Carson's context model) and be aware of the issues relating to the broader capability and how it can change. Such issues are reduced if the system can be made more independent of its wider system to reduce the reliance on joint resources/services and by reducing the impact of its own activity.

4.2 Context and the functional requirement

The requirement starts with the functional view as this defines the mission (Hitchins, 2008). However functional methods have a narrow perspective on a problem and non-functional requirements and the accompanying physical structure are also important. Arguably the purely functional view can be applied with some success for software development. Software engineering practices have had a significant driving influence on formalised techniques of systems design, but in software it is much easier to work in functional terms and produce an acceptable end product. In systems where the function is attributed to more than the software, the non-functional requirements become more important with

the system being subject to many more environmental conditions that may impact the effectiveness of the solution. Alexander asserted that there is an intrinsic relationship between the context, requirement and the solution (Alexander, 1964), and techniques for defining requirements that ignore their dependency on the context of the problem and the likely solutions, are liable to make unjustified simplifications. Therefore a vehicle for specifying functionality is needed and, according to Alexander, it should meet the following criteria:

1. Has a necessary and sufficient definition of parameters to define a functional as a “black box” with inputs, outputs, resources and controls
2. It is able to define the system context in a way that enables scenarios to be developed that allow non-functional performance to be evaluated
3. It reflects the necessary aspects of the solution that enable elaboration of the solution architecture; see also (K. Ulrich, 1995)

The SADT IDEF0 representation is preferred for this research, because it is widely recognised and it contains the necessary and sufficient “black box” parameters for functional definition. However, as it is just a functional notation, additional context needs to be provided. Flood and Carson’s concept of systems context (Flood & Carson, 1983) divides entities of the context into four levels according to the relationship each has with the system of interest. This representation is again preferred for its simplicity and by combining these two concepts, both points 1 and 2 above are addressed. The inclusion of “mechanisms” in the IDEF0 definition, and the identification of the system of interest boundary in Flood and Carson’s context model, provide a means of addressing point 3. As a combination of the ideas of SADT and the concepts of Flood and Carson’s Context Model, this diagram will be defined as a Functional Context Diagram (Figure 5). To ensure completeness, transformation, management and resource reflect the different functional drivers defined by Hitchins Generic Reference Model of complete functionality, and Checkland’s CATWOE is used for a complete set of stakeholders (Checkland & Poulter, 2006).

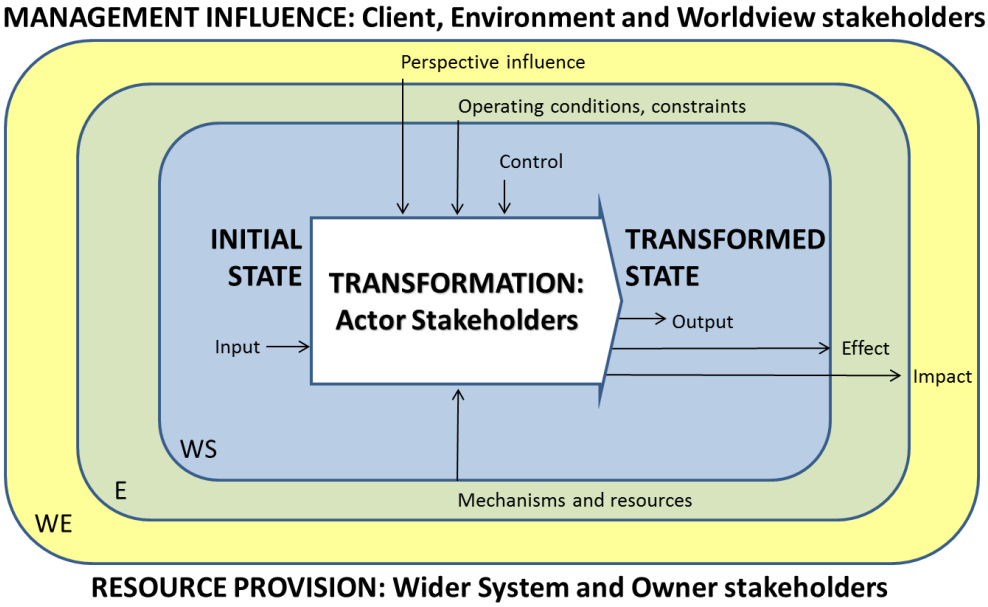


Figure 5: Functional Context Diagram

4.3 Context types related to definition of lifecycle approach

The types identified below represent a combination of academic concepts reported in the literature and a collation of real-life engineering projects. Whilst they refer to types of problem, a problem is not exclusively of one type. In most cases it will be necessary to characterise a problem as being of many types, each of these influencing the system design approach that is required. They have been grouped in terms of the particular aspect of the System design process that they most closely relate to i.e. requirements, design or the overall lifecycle. A table summarising the way that this might influence the approach taken is then provided.

Many shortfalls in problem solving can with hindsight be attributed to applying the wrong approach for the specific problem and its situation or context. Having identified a problem it can then be both a challenge to determine strategies that will succeed in its solution and also to communicate the value of what is proposed to gain acceptance of the way forward. The challenge here is to select an approach based upon both an understanding of the problem context and an identification of the severity of the problem in terms of the risk.

The method described proposes *Context types* to help analyze a problem context (Mackley, 2015). The analysis of each type is reduced to a four-quadrant matrix, where a particular quadrant can be used to define the appropriate system thinking or systems engineering approach. Each quadrant is also related to the likely level of risk or difficulty in addressing the problem. The resulting level of risk is then

expressed graphically as a Kiviat diagram in order to present it in a way that can facilitate communication and understanding by a wider audience.

First, a review is made of the current theory that can be used to develop an understanding of the types of context and lead to suggested approaches. Where appropriate these are expanded to generate a set of generic *Context type*s described as four-quadrant matrices. New and complementary *Context type*s are then proposed with the aim of providing a more complete analysis of the problem context. Finally an outline is given of how the *Context type*s can be used to suggest problem solving strategies and indicate the level of complexity and risk involved. By describing how to address problem contexts of different types, the method presents a unification of existing systems thinking approaches to provide a problem solving approach that can be tailored to specific circumstances.

4.3.1 Existing concepts

There are a number of existing concepts that allow a distinction to be made between different types of context and these are outlined below.

4.3.1.1 **Problem types**

Problem types characterize a problem in terms of uncertainty in requirement and in solution (Obeng, 1995).

- Painting-by-numbers (PBN) – clear objective and clear solution
- Foggy – uncertain objective, uncertain solution
- Movie – uncertain objective, clear solution
- Quest – clear objective, uncertain solution

Obeng defines a *Painting-by-numbers* problem as one where “you and most stakeholders are sure of both what to do and how it is done” based on similar experience. The fact that the problem is well defined and there is a clearly defined solution, means that technical, cost and timescale risk can be well identified; the challenge is perhaps to do it better.

A *Foggy* type of problem is very different in that “you and most of your stakeholders are unsure of what is to be done and unsure of how it is to be achieved”. The secret of success here, according to Obeng is to “proceed very carefully, to proceed one step at a time”.

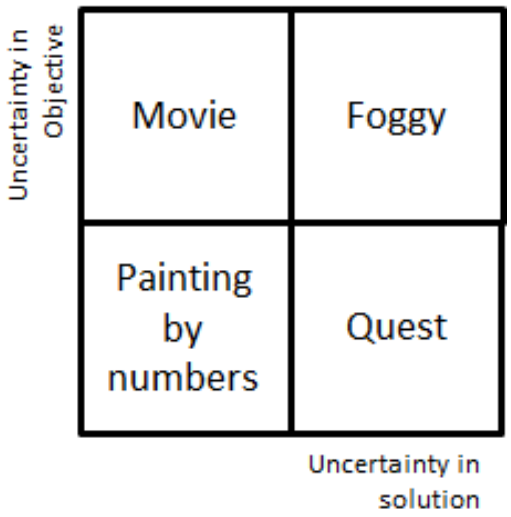


Figure 6: Problem type

In the *Movie* type “you and most of your stakeholders are very sure about how the project should be conducted but not what is to be done”. Typical expertise and facilities are in place, either looking or waiting for the problem to be tackled. In a Movie, Obeng says that concentration should be on “finding yourself a good script and the movie will write itself”.

For a *Quest*, “you and most of your stakeholders are sure of what should be done…however, you are unsure of how to achieve this”. The secret here, Obeng says, is to “get your knights fired up and send them off to seek [a solution] in parallel”.

Obeng’s aim is to identify that not all problems are of the same type and used characteristics of uncertainty in both objective and solution to categorize them. In doing so he emphasizes that a single approach was not appropriate for all. His four types are already arranged as a four quadrant matrix as shown in Figure 6.

Whilst this type frames the problem, strategies need to be evolved that address them and architectures designed in support of them. Painting-by-numbers is probably the easiest to address as it involves a clear requirement and a solution that has been established to work. In this case there is benefit around keeping the architecture the same as before with all the experience that provides into system properties and behaviour. If there is any change to be applied it is in improving performance, cost or timescale of development. Such a situation would seem to have a good fit with “lean" techniques.

Whilst the Movie type reflects a solution that is known, or at least familiar it also has to address uncertainty in the objective. This will requires knowledge/assurance of how the proposed solution will suit potential scenarios that may arise. Therefore analysis should focus on devising configurations that

will perform best in the variety of potential scenarios. As there will always be uncertainty in how the design can cope with requirements that are not fully known, an architecture that can allow some adaptability or flexibility would be an advantage; service oriented architectures are examples of this

A “quest” problem has a clear objective, but without a clear solution. In such an instance an approach that examines a number of different possible solutions will enable a better indication of what solutions and solution features will provide the right effectiveness. Different architectures are likely to be required in order to provide a useful variety of options to be considered. It is however conceivable that having a clear view of the objective will infer particular behaviour requirements from any solution that can lead to a favouring of certain archetypes that are known to provide these.

A “foggy” problem on the other hand removes that clarity of requirement. In this case iteration is normally required between a progressive maturing of the objective requirements and the potential for their achievement through design. As understanding of the problem becomes clearer it should be expected that architectural strategies will be the first aspect of system design to be developed. There can be no presumption of architecture in the Foggy problem, only there expectation that it will evolve hand-in-hand with the requirement.

In defining his problem types, Obeng has emphasised that a single approach is not appropriate for all and that an approach based on exploration would be required. Here it has been argued that more detailed strategies can be applied depending on the problem type and that the type also infers the architectural approach to be employed.

4.3.1.2 **Management type**

In the early 1960s Jay Forrester applied the concept of System Dynamics to the industrial organization (Forrester & Cambridge, 1961). This provided the basis for further work (P. M. Senge et al., 1994) using System Archetypes to analyse a business as a system. The basis of his contention was that dynamic situations could be described in terms of reinforcing and balancing loops of cause and effect and that simulation using archetypes is able to replicate the behaviour of ailing organizations thus providing a diagnosis of the reasons for their malaise. The original set of around ten systems archetypes can be expressed as a reduced set of four; Underachievement, Relative Underachievement, Out-of-control and Relative Control archetypes (Wolstenholme, 2003). These represent situations where there is either a problem in terms of availability of resource or in terms of an inappropriate control action being applied (Wolstenholme, 2004a). In this context type an assumption is made that inappropriate control is applied unintentionally and therefore as a result of a lack of situational awareness.

Using axes of “lack of situational awareness” and “inadequacy of resource” the four quadrant matrix of Figure 7 can be identified.

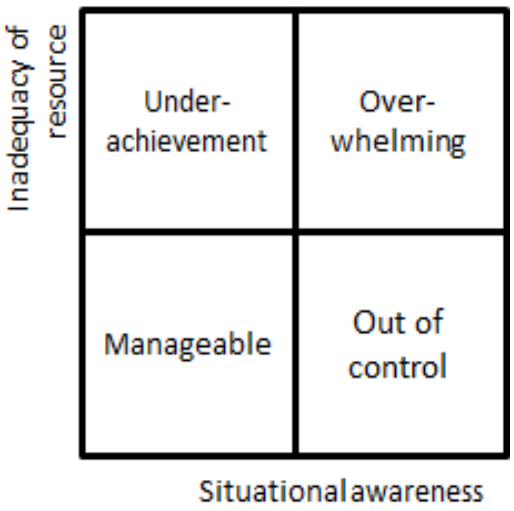


Figure 7: Management type

4.3.1.3 **Values type**

The concept of Divergence of values (Jackson, 1994) consists of unitary, pluralist and conflicting/coercive situations:

- Unitary - in that they all have a common goal and view of what is to be achieved and ultimately how.
- Pluralist - in that stakeholders cannot agree on goals and tend to pursue their own objectives, but that there is mutual benefit in the collaboration.
- Conflicting/Coercive - in that goals and objectives diverge, but that some group or groups get their way at the expense of others.

These situations are interpreted as distinguishing between the number of different viewpoints, and the degree of conflict that exists between stakeholders. In a collaborative environment an increasing number of viewpoints change a situation from *unitary* to *pluralist*. However, where there are conflicting priorities increasing the number of viewpoints will turn a situation from a *coercive* or simple conflict into *anarchy*. The resulting *context type* is shown in Figure 8.

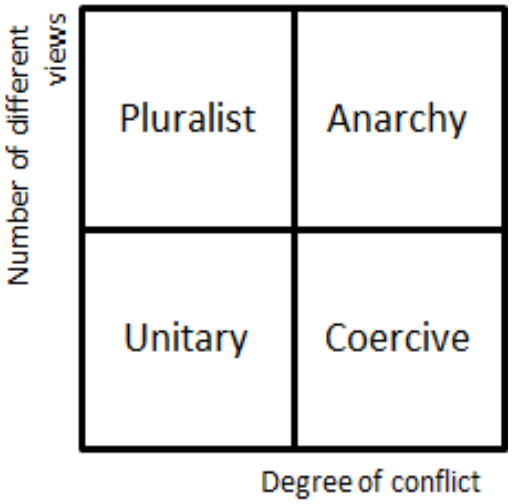


Figure 8: Values type

4.3.1.4 **Complexity type**

This concept makes the distinction of what problems are complex (Snowden & Boone, 2007), defining four quadrants:

- Simple – the relationship between cause and effect is obvious to all
- Complicated – the relationship between cause and effect requires analysis or some other form of investigation and/or the application of expert knowledge
- Complex – the relationship between cause and effect can only be perceived in retrospect
- Chaotic – no relationship between cause and effect at systems level

Snowden's definitions make the distinction between difficulty in analysis which creates complicated problems and unpredictability of outcome that results in complex problems; the combination of the two resulting in a chaotic situation (Figure 9).

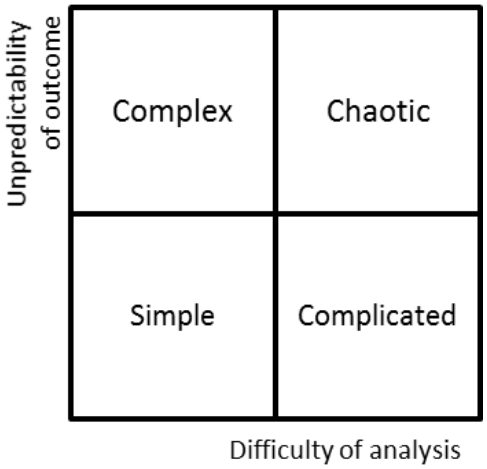


Figure 9: Complexity type

4.3.1.5 **Co-ordination type**

Finally Meier (Maier, 1998) distinguishes between types of organization of a system from a unitary system to a system of systems, on the basis of operational and development independence of its components. His definition for a system-of-systems is:

> "an assemblage of components which individually may be regarded as systems, and which possesses two additional properties:
>
> Operational Independence of the components: if the system-of-systems is disassembled into its component systems the component systems must be able to usefully operate independently. That is, the components fulfill customer-operator purposes on their own.
>
> Managerial Independence of the components: the component systems not only can operate independently, they do operate independently. The component systems are separately acquired and integrated, but maintain a continuing operational existence independent of the system-of-systems."

Maier's concept of *system-of-systems* contrasts with a unitary or *centralized* system; a system-of-systems displays both development and operational independence whereas the *centralized* system has neither of these. Considering solely development independence will lead to an *off the shelf* solution (i.e. assembled from separately developed components), whereas solely operational independence implies an *asset management* case (see Figure 10).

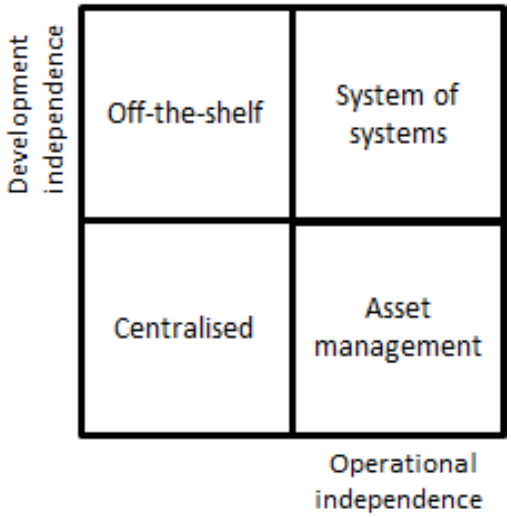


Figure 10: Coordination type

4.3.2 Further context types

This section, additional context types have been developed to complement the five generated from current theory. To cover the variety of problem situations a total of eleven *context type*s are described. In each case it is useful to keep in mind the question “how critical could this *context type* be to influencing the required approach of the problem solver?”

4.3.2.1 **Evolution types**

Obeng’s Problem Type concept is an established way of addressing a particular problem at a given time, but often the challenge comes from how the problem changes over time. Important considerations are: how much has the requirement changed; what is the uncertainty of the requirement or in the solution as a result; when and how often does the problem need to be addressed to ensure continuous capability provision?

The rate of change of requirement is important as this will tend to erode any spare capacity built into the system or may expose areas where the system currently has no inherent capability. This will determine how long it will be before the system is in capability deficit and will drive the time at which modification is required as well as the duration of modification activity that can be tolerated. For instance, in a rapidly changing environment, capability may need to be updated on a regular basis and the time taken to perform the update must be consistent with those challenging timescales in order to converge upon a solution before further updates are required. Equally the uncertainty in requirement is important as this will drive the type of approach needed to address the capability update and indicate the time that the activity is likely to take. Effectively this is predicting

the Problem type (Obeng, 1995) that is likely to be encountered at the time in the future when the modification will be required (Mackley, Deane, & John, 2010).

For this type the axes of the four quadrant matrix are uncertainty in future objectives and uncertainty in future solution. If the future objectives and solution are clear, then the situation will be one of routine *obsolescence management*. This could be the situation for road vehicle rental firms; vehicle design has remained fairly invariant over many years and the users expectations are very much in line with what a current road vehicle can provide. However what if the future objectives or possible solutions were not known? Imagine that current vehicle solutions based on oil based fuels were becoming less economic and vehicles using alternative energy become more attractive – broadening the business to consider these would be seen as *opportunity development*. Conversely, if we imagined that the technologies of cars in the future are to become expensive and cars or their components become leased then this is more an area of *service development* (such as leasing of batteries for electric cars). A rapidly changing environment with novel and emerging solutions could be termed as represents *capability development*, resembling the approach often taken in military development, but would arguably fit well with mobile computing and communication solutions. Evolution types can thus be identified as in Figure 11.

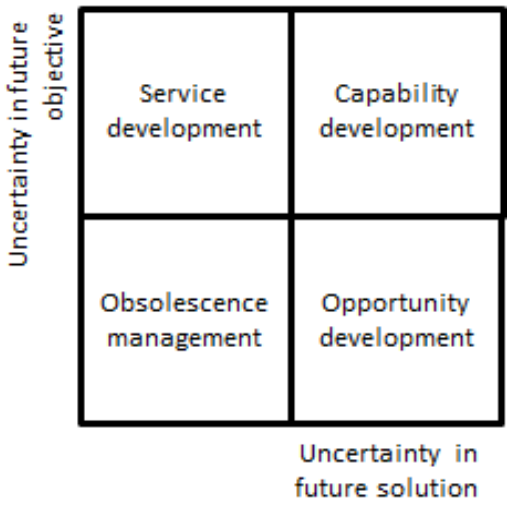


Figure 11: Evolution type

4.3.2.2 **Response type**

The focus for this type is the urgency of the need. Depending on the complexity of the problem, a more or less urgent need will have a bearing on the approach taken. To characterize urgency a distinction is made between developing a solution under normal commercial conditions i.e. working in a viable and competitive situation, and an emergency situation where corners are allowed to be cut or significant extra resource is justified. Urgent but non-complex situations

can be addressed by cutting corners as the consequences of this can be evaluated. If a situation is both urgent and complex then simple measures are often not appropriate as they may have consequences that in themselves can have serious implications. In the matrix below the distinction is made between the former, similar to the Urgent operational requirement process employed by military organizations and the latter being a *systemic emergency*. An example of a systemic emergency might be an outbreak of a highly virulence strain of flu and its effect on a countries health service and economy. *Routine* and *systemic development* make up the four quadrants of Figure 12.

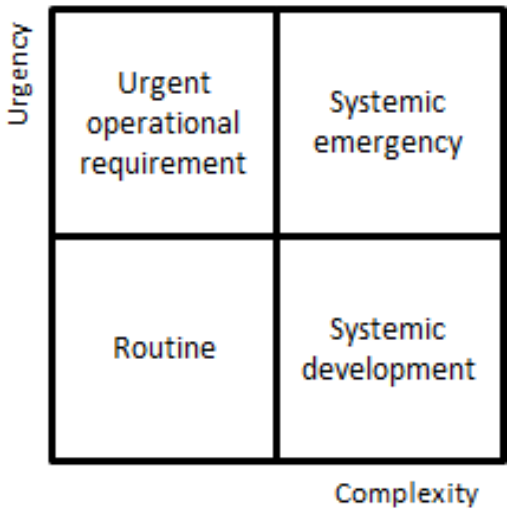


Figure 12: Response type

4.3.2.3 **Situation type**

It is clear that the starting point will have a significant bearing on the solution and so this aspect will be key in determining the approach required. The following situations might be encountered based on differences in uncertainty of design baseline and the degree of change required.

The Situation Type involves consideration of what the starting point of the activity is. For instance this may be:

- Design starting from a *clean sheet*, with little or no previously defined concept of design or legacy constraint (e.g. new capability acquisition). The truly clean sheet is not a common situation for the system designer, although it is perhaps more prevalent in some domains than others (e.g. defense).
- An *upgrade* of capability, where the starting point is going to have a considerable bearing on the solution that might be chosen (e.g. mid-life

update). In this situation it will be normal to identify the “capability gap” that needs to be met.

- A need for *system review*, to identify changes required to the system baseline to be fit for the existing purpose, rather than from the definition from stakeholders of a required change in capability.
- Simply a *reconfiguration* of what is already in place, but used in a different way to solve the problem. In isolation this is a relatively simple case, but it can also describe a system-of-systems which provides challenges of its own (see Coordination type).

The four quadrant matrix for Situation type is given in Figure 13.

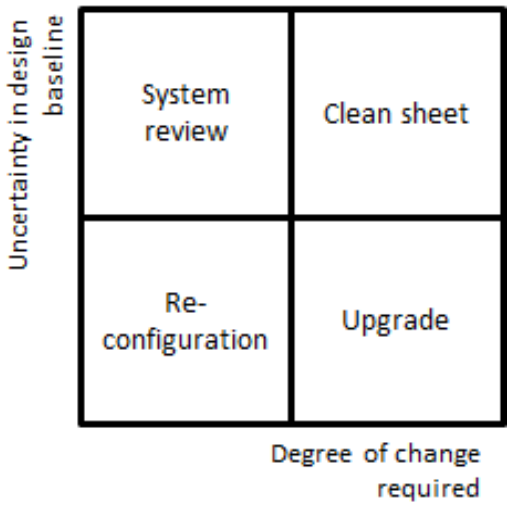


Figure 13: Situation type

4.3.2.4 **Risk type**

Risk and maturity are key elements of a system development that should be considered together. With an immature system, achieving the desired system outcome without clear knowledge of probability of success or related consequences represents a risk. Equally, a relatively mature system can be a risk if there are severe consequences should it fail. Engineers work at trying to find a suitable balance between risk and maturity of a system design. The preference is a mature/low risk combination or a *no brainer*, but for higher risk situations a project may choose a mature solution to *play it safe*. If there is solution immaturity then low risk solutions represent *calculated risks*, with a high risk/immature solution being a *gamble*.

The four quadrant matrix for Risk type could be drawn as in Figure 14.

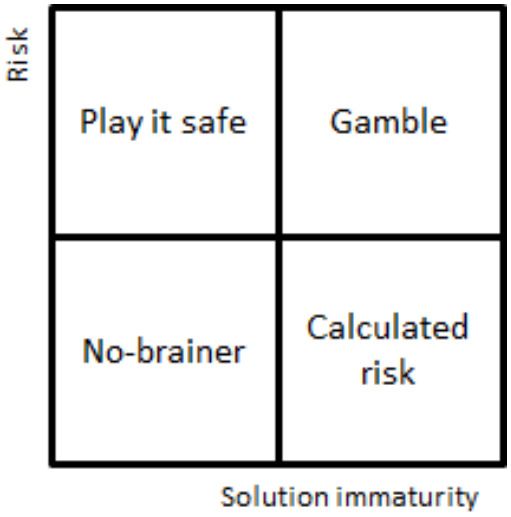


Figure 14: Risk type

4.3.2.5 **Target type**

Enterprises will often find themselves facing different types of target. Some enterprises are required to deliver to strict timescales and others might have a reputation based on the quality of their product or service. As shown in Figure 15, these represent orthogonal axes, where a high quality challenging target situation can be seen as an O*lympic sprint* compared with a relaxed timescale at a familiar and achievable quality being the *stock in trade*. *Critical path* and *gold standard* provide the remaining quadrants.

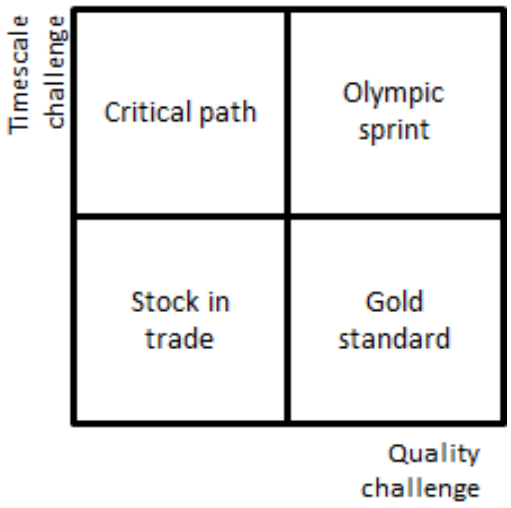


Figure 15: Target type

4.3.2.6 **Business area type**

A particular challenge for a business is to ensure it has the capability to deal with a problem, and in particular that it has a properly trained and prepared workforce.

A distinction can be made between the requirements that a given context places on expertise that is gained with professional qualifications on the one hand, compared with experience on the other. Whereas expertise might be acquired quickly, experience has to be accumulated over time: in some areas, expertise is in short supply and that introduces challenges of its own. Types of work are often referred to as “collar workers”, but the different “collars” do not always reflect the distinction of education and experience, so categories of *low skill*, *professional*, *trades*, *gold collar* have been chosen as in Figure 16.

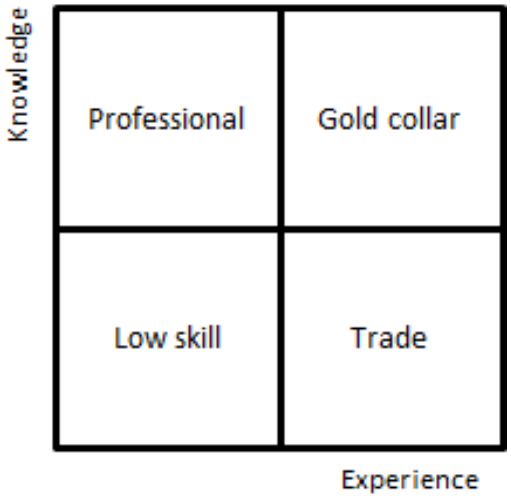


Figure 16: Business area

4.3.3 Combination types

An analysis of the identified types shows that there are common axes. For instance, risk type compares risk against solution immaturity whereas Obeng’s problem types compare solution immaturity to objective uncertainty. This allows the combination to be described as a 3D matrix introducing types of; *play it safe PBN*, *surefire success movie*, *critical quest,* and *freezing fog*. This combination can be described as “Problem risk type” (Figure 17).

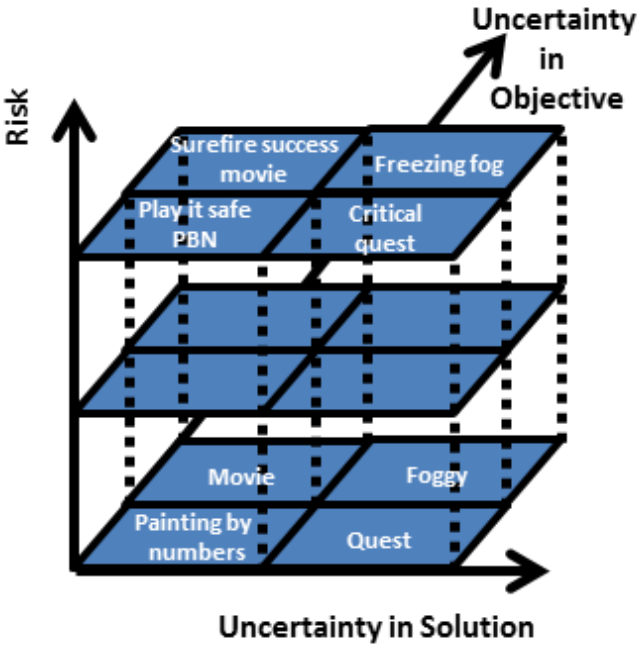


Figure 17: Problem risk type

Also the types of response and complexity share an axis of complexity, which leads to urgency being compared with both unpredictability of outcome and difficulty of analysis. This introduces types of *urgent operational requirement*, *balanced scorecard*, *tiger team*, *systemic development* and *systemic emergency*. This is described as “Urgent complexity types” (Figure 18).

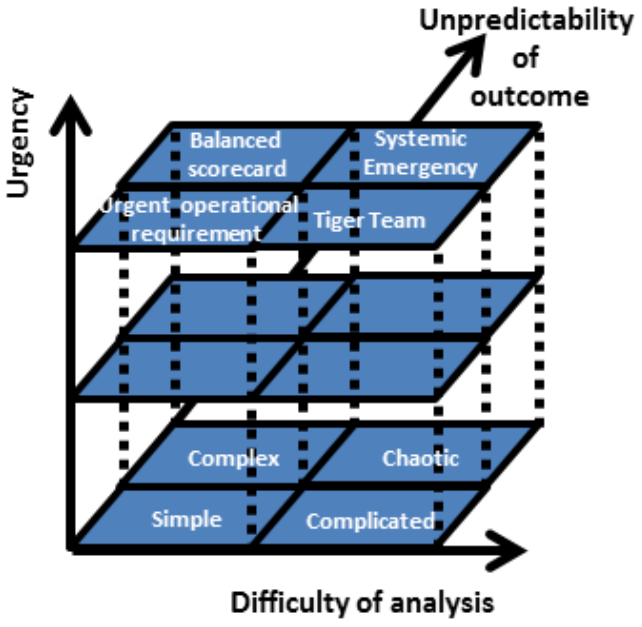


Figure 18: Urgent complexity type

4.3.4 Problem solving approach and risk evaluation

The use of the four quadrant matrix for describing each *context type*, allows a spectrum of context to be identified. Each matrix is structured in such a way that risk increases as the value of any single axis increases. Figure 19 is numbered to provide a reference and, in coarse terms we might conclude that quadrant 1 represents low risk, quadrant 4 represents high risk and quadrant 2 and 3 indicating a medium risk. Thus an overall context risk might be evaluated by identifying where a given context falls for each of the types.

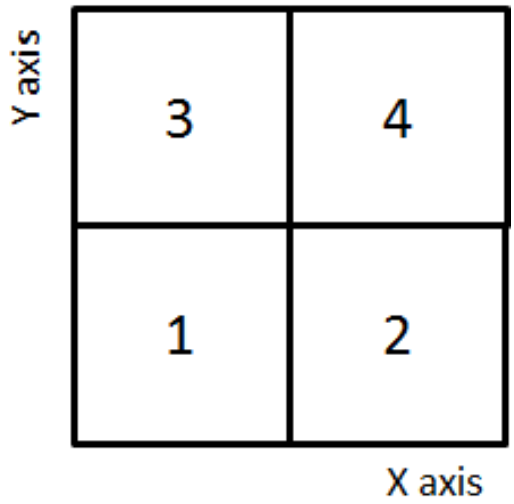


Figure 19: Risk evaluation matrix

For identifying risk it is important to ensure that all potential contributors are considered. There is perhaps no guaranteed way of determining that the list of *context type*s addresses all elements of potential risk in a system solution to a problem and this is an area which deserves further analysis against more traditional risk indices. However, it is possible to identify key domains of a system's problem and solution space that should be considered. Key domains of a system have been described as: product and producing domains (Mackley, 2008); product, process and organization (S D Eppinger & Salminen, 2001); customer, functional, physical and process (Suh, 1990). These can be combined to give domains of *requirement*, *solution*, *process* and *organization*. Mapping the eleven *context type*s to these four domains there is coverage in each domain with either two or three types each.

The division is shown in Table 7. Table 7 also shows a simple illustration of the approach for two example problems. Imagine being asked to address problems facing the UK National Health Service, or being asked to work out a strategy for developing a new concept of airplane based on a new distributed propulsion concept. The table shows an analysis of both problems; the crosses represent

the problem of developing the new aircraft and the ticks represent the problem of addressing the challenges of the UK National Health Service (NHS). The risk profile for the distributed propulsion problem is analyzed as 4,5,2 and so seems to represent a medium risk, with a tendency to areas that are manageable rather than risky: the situation for the problem of the NHS shows a risk "profile" of 0,5,6 which indicates no easy areas, with risk in almost half the areas being high. For distributed propulsion the risk is reasonably well distributed across the system domains and therefore requires a balanced approach: for the NHS there are significant organizational risks to overcome and these stand-out compared to risks of process, requirement and solution.

Table 7: Characterizing risk: Examples

Type	Quadrant 1	Quadrant 2,3	Quadrant 4
Process			
Problem Evolution Response		X √ X √ X	 √
Requirement			
Situation Divergence of values Management	 X X	√ √	X √
Solution			
Risk Complexity		X √ X	 √
Organization			
Coordination Target Business area	X X		√ √ X √
Summary risk	**X (4) √(0)**	X (5) √ (**5**)	X (2) √ (**6**)

This can be effectively visualized using the Kiviat diagram (Figure 20), which gives an immediate pictorial view of what areas represent the greatest risk (with 1, 2 and 3 being low, medium and high risk respectively).

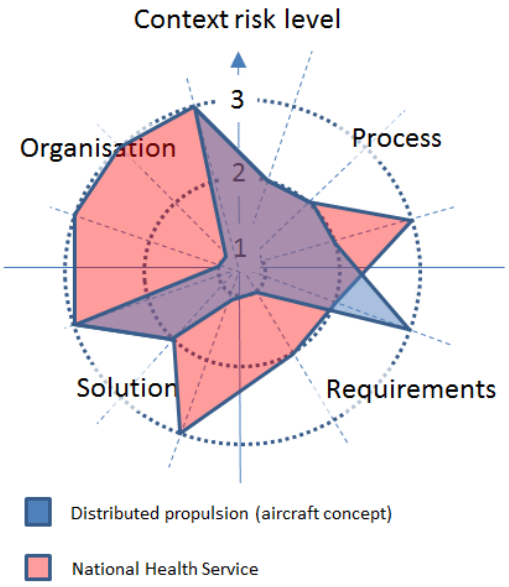


Figure 20: Kiveat diagram of example risk scores

The four quadrant matrices can be used to gain an idea of overall difficulty by considering each type individually and assessing the combination of the outcomes. However, this four quadrant notation has the risk of dividing up the problem without considering the interactions. As this is a qualitative tool to inform a strategic approach, these overlaps are considered small and are expected to be addressed in ensuring a coherent strategy for the whole problem. Some overlap in the *context type*s can readily be identified by the Combination types, which reflect combinations of issues that should be addressed to identify their impact on the approach taken. In the examples given the *Problem risk type* results in a *critical quest* for both the UK NHS and new aircraft concept, whereas the *Urgent complexity type* emphasizes a *systemic emergency* for the NHS rather than a *systemic development* for the aircraft.

Consideration of context type can have a bearing on the approach used; some examples are identified in Table 8.

Table 8: Architectural approach according to context type (sub-type numbers are as Figure 19)

Type	Sub-type	Approach
Problem types	1. Painting by numbers	Follow existing tried and tested process and retain existing architecture
	2. Quest	Examination of solutions using an incremental approach; new architectures are to be evolved, but may take benefits from established patterns or archetypes
	3. Movie	Scenarios need to be examined to establish use of existing assets. An existing architecture will be in place, but use of a Service Oriented Architecture will facilitate more flexibility asset management (Russell & Xu, 2007)
	4. Foggy	Iterative and exploratory approach; a new architecture is to be evolved, but may take benefits from established patterns or archetypes
Management type	1. Manageable	No particular action is required as the system solution is in functioning well in its context.
	2. Out of control	The out of control situation means that there may be insufficient variety in the solution to control the variety of influence in the systems context. The solution involves increasing the variety in solution, as identified by Ashby in his theory of Requisite Variety (Ashby, 1991)
	3. Under-achievement	The under-achievement situation is generated in situations where there is inappropriate resources for the system to perform. The solution is to either establish an increase in available resources or to increase the system variety in a way that makes more efficient use of the available resources (Wolstenholme, 2004b). Architecturally the solution may benefit from widening the system boundary to enable a better policy on use of available resources.
	4. Overwhelming	Solution requires both an increase in resource and variety to provide a system in balance with its context and avoid the out of control and under resourced outcomes (see strategies for out-of-control and underachievement above)
Values type	1. Unitary	Approach can be based on Consensus, with a clear definition of boundary and architecture. It is perhaps the simplest case for systems engineering, where there is a clear overriding client objective and other stakeholder requirements are defined purely as constraints on the design. Trade-offs and architectural definition will generally be at the design level.
	2. Coercive	Stakeholder views may appear unitary, but mask coercion. Ulrich`s Critical Heuristics (W. Ulrich, 1987) can be used to establish where the system boundaries ought to be. Regulation may subsequently be required to enforce an appropriate architecture.
	3. Pluralist	In contrast to the Unitary case, there will be different driving perspectives on the objectives, and priorities will differ. The approach will be subject to agreement based on compromise. Discussions will need to be informed by trade-off studies at the requirements level and therefore require consideration of both the functional and physical architecture of the system. Soft systems

Type	Sub-type	Approach
		methodologies (Checkland, 1981a) can be used to establish suitable compromises.
	4. Anarchy	There is no sense of centralized objectives and responsibility, and therefore no coordinated strategies for achieving outcomes. No meaningful structure exists. Architectural rules and structure need to be established and enforced, addressing stakeholder views, but also establishing a view of social norms (such as law and order)
Complexity type	1. Simple	The relationship between cause and effect is obvious to all
	2. Complicated	The relationship between cause and effect requires analysis or some other form of investigation and/or the application of expert knowledge
	3. Complex	The relationship between cause and effect can only be perceived in retrospect
	4. Chaotic	No relationship between cause and effect at systems level
Coordination type	1. Centralised	Encourages the use of a standard product lifecycle using a bespoke architecture. Both operational and managerial control over the system allows complete control of the system and its development. Whilst it is possible to mismanage there will not be independently generated influences from elements within the system architecture itself.
	2. Asset management	Operational independence is provided by establishing a service agreement as the system requirement. The development of the service system and provision of the service should follow a service based lifecycle and employ a service oriented architecture.
	3. Off-the-shelf	To cope with ownership/managerial independence the strategy should be to delegate development responsibility with clear guidelines. A top down modular approach using open standardised architecture is called for.
	4. Systems of systems	Service oriented architecture can be employed. Agile processes are desirable in order to maximize flexibility in operation (Mackley et al., 2010)
Evolution Type	1. Obsolescence management	Assumes an existing architecture, although a modular architecture if chosen in the original concept, will reduce the burden associated with obsolescence management.
	2. Opportunity development	Benefits from a service oriented architecture and required an analysis of the likely scenarios to be examined
	3. Service development	Benefits from a service oriented architecture and required an analysis of the likely scenarios to be examined
	4. Capability development	Examination of solutions to meet a capability gap. There should be no assumption of solution (MOD, 2005)
Response type	1. Routine	Refer to complexity type for guidance. Dealing with the urgency dimension requires agile methods (Mackley et al., 2010)
	2. Systemic development	
	3.Urgent operational requirement	

Type	Sub-type	Approach
	4. Systemic emergency	
Situation type	1. Reconfiguration	Design is largely unchanged, but requalification is required for any new operational requirements. This requires examination of the use of the systems and the conditions involved to determine if there has been an extension to the performance envelope that will need to be re-qualified.
	2. Upgrade	Upgrades will usually reflect a need to modify the system as elements have become obsolete, or because an insertion of new technology is desired. A modular design, with standardized interfaces will enable replacement of affected modules and result in a minimal requalification for the upgraded build standard. Upgrade is facilitated if it part of a pre-envisaged Incremental Development Lifecycle model.
	3. System review	In cases where there is no scheduled system upgrade, but there are clear symptoms of the system performing beneath the desired performance levels the first step will be to diagnose an agree the appropriate way forward. In order to identify, analysis and diagnose the root causes of underperformance, it is appropriate to employ an issue or soft systems analysis such as the Rigorous Soft Method (Hitchins, 2008) or Soft Systems Methodology (Checkland, 1981a)
	4. Clean sheet	Design from new using an exploratory, capability based approach. Suited to an initial approach of implementing a spiral based lifecycle and verified for completeness against methods such as complete systems methodologies such as Hitchins' Generic Reference Model (Hitchins, 2008).
Risk type	1. No-brainer	Solutions to problem have relatively little risk exposure due to experience. This requires following the established process gained by experience. In cases of an established process, then "Lean" techniques can be considered to improve time, cost or quality.
	2. Calculated risk	Despite lack of maturity, the relatively low level of risk means that a trial based approach is both acceptable and desirable as it can provide validated outcomes to converge on the solution.
	3. Play it safe	Risk consequence requires an established, tried and tested approach. Tends to be highly procedural based on previous experience and changes to established architectures and design are resisted due to the effort and cost of requalification.
	4. Gamble	This applies for situations where there is tangible risk to the system or its context and typically this would be a safety or security issue or other situations with significant implications on an enterprise. With the lack of confidence, fail safe measures need to be incorporated to limit the damage in case of unforeseen behaviour.
Target type and Business area type	Refer to the enterprise rather than the product system and so not considered	

5 A MODULAR APPROACH TO SYSTEMS DESIGN

5.1 Basic principles of the System Design or Architecting process

Architecture concentrates on arrangements of entities or elements rather than their detailed design; the intention is to put in place a framework for the detailed design activity in order to ensure a good solution. A basic principle of the system design or is one of order. By applying order to a problem it is possible to:

- understand how it works
- provide benefit from the structure achieved
- manage concurrent achievement of desired outcomes (Crawley et al., 2004).

The understanding of systems becomes more difficult as systems become more complex and hence the role of architecture becomes more important. Bar Yam (Bar-Yam, 1997) defines characteristics of a complex system as:

- Elements (and their number)
- Interactions (and their strength)
- Formation/Operation (activities and their objectives/ timescales)
- Diversity/Variability
- Environment (and its demands)

Here, Bar Yam identifies that the complexity of a system can be both in product and its producing systems, both of which need to be examined by the system designer for an effective solution (Mackley, 2008). In Bar Yam's terms, the product architecture consists of elements, interactions and operational activities, whereas the producing system is similarly described by elements and interactions, but associated with development rather than operation. Whilst Bar Yam's characteristics are instructive in highlighting that complexity is both in the system and its associated development processes, his definition does not make a distinction between them. Gershenson and Prasad (Gershenson & Prasad, 1997) describe "attribute independence" and "process independence", where the "attributes" refer to the physical attributes of a coffee maker and the "process" refers to the process of making it. Here "process similarity" is a means of grouping "components and sub-assemblies which undergo the same manufacturing processes". The consideration of the product and its production as distinct systems allows architecting principles to be applied to the design of both. Therefore, in this research it is proposed to reduce the characteristics so that the

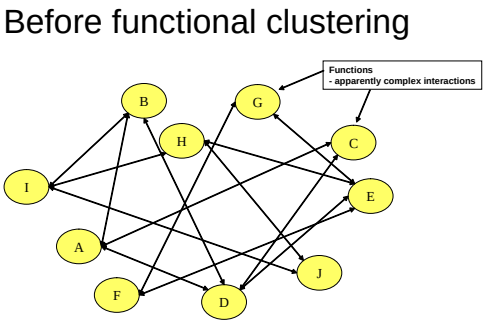
definitions can be employed identically to either product or process i.e. complexity of elements/activities, complexity of interactions, complexity due to variety (diversity/variability). Complexity due to environment is external to the system.

The analysis of Table 9 allows a direct mapping between the complex system characteristics identified and the systems architecture principles discussed in Chapter 2 (simplicity, modularity and similarity, where independence is considered a strategy that promotes modularity).

Table 9: The influence of architectural principles on Bar Yam's system characteristics (product and process)

	Simplicity	Modularity/ independence	Similarity
Complexity of Elements	**Simple elements reduce complexity**	Modularity does not reduce element complexity	Similar elements do not reduce their individual element complexity
Complexity of Interactions	Similar elements do not in themselves reduce interaction	**Modularity reduces interaction**	Similar elements do not in themselves reduce interaction
Complexity due to Variety	Simplicity is not a guarantee of less variety	Modularity is not a guarantee of less variety	**Similarity reduces variety**

The table indicates that the complex or complicated characteristics of both product and process can be addressed by clear and understood principles of architecting. The complexity of elements and their variety are often determined by suppliers and are not under the direct control of the system designer and it therefore might be argued that the greatest influence that the system designer can have is to influence the complexity or complication of the interactions. It is possible to see how interactions can be used to manage this by reference to the following figures, which depict functional coupling.



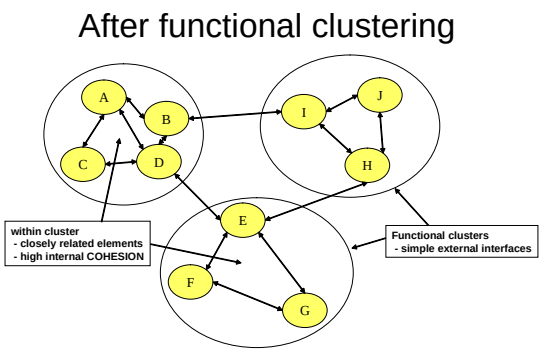


Figure 21: Functional grouping without order

Figure 22: Functional grouping with order

Figure 21 and Figure 22 show a diagrammatic representation of a group of functions and their interactions. The first shows an apparently complicated network of interactions, but the second shows the same set of functions and interactions arranged in a much more orderly way. This is functional clustering, which is a key element of ensuring a modular design. This arranges functions into groups that minimise the amount of interaction outside of the group, which is an advantage both for the system in operation and also in development where one could imagine that the design activities associated with each group could be managed relatively independently, whilst checking the simple interfaces outside the group on a more occasional basis. Whilst the functional architecture is an abstract notion, it has a physical implication in that it describes an efficient way of organising the system design and development activity.

When functions are subsequently allocated to subsystems then often the clustering is not preserved as in Figure 23. The misalignment between functional and physical elements then threatens to remove the benefits of order in the functional architecture by partitioning it in the physical design. Inspection will show that the number of interfaces between the physical elements is now increased, representing an increase in complication that will make the system design harder to manage.

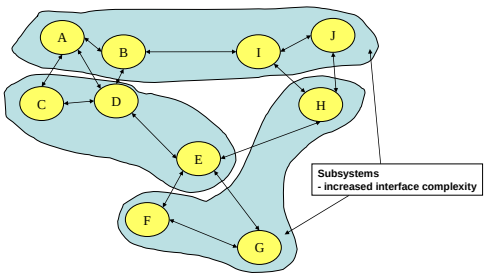


Figure 23: Reduced order after allocation to subsystems

If the functional and physical architectures were made to align, the benefits of a well-structured functional architecture could be returned, which is the principle behind functional independence. There is however no guarantee that other influences will allow such an alignment (Sako, 2003); for instance, a favourable solution for clustering for functional reasons might not be good for others. Whilst functional grouping is important, other influences can be drivers such as non-functional, organisational and lifecycle influences:

- Physical influence: groupings in terms of subsystems, units or modules of hardware or software that are developed independently as part of a system hierarchy
- Non-functional influence: groupings of elements of the design that are associated with improving a specific quality attribute
- Organisational influence: where the groupings of elements are such that analysis and development of elements of the design are facilitated
- Lifecycle influence: an arrangement of elements of a design that suit different lifecycle stages; for instance, an arrangement for development may not suit the maintenance policy or disposal policy

Put in another way, a design that is ordered according to function expedience, may not suit its non-functional management, its physical allocation, its organisational ownership or its lifecycle management. A way of arranging the various architectures with respect to each other is needed and as function reflects what the stakeholders require from the system then this is the most logical place to start. A method for addressing the functional partitioning problem is proposed in the next section: when this is addressed, the other domains can also be considered.

5.2 System Design: Functional

5.2.1 Functional Interaction Types

As identified in the previous section, a key aspect of architectural design that the system designer can influence, is the management of the complication of the interfaces. Attention should be given to this as early as possible in the system design process, but to do this requires sufficient understanding of how functions will behave, which is often not known at an early stage. Functional analysis is often described in terms of data flow diagrams that describe the data interface and sequence of activities (e.g. Yourdon, SysML), but these methods do not describe the *nature* of the function that is key to a system's behaviour and to the way it can be analysed and managed. The earlier literature search has identified that many analysts view architecture as a matter of coupling and that this can be used to some benefit in using the method of Design Structure Matrices to produce a modular design Section 2.5.2. However, this doesn't take into account the criticality of the interfaces or their behaviour; greater independence is expected to be achieved by virtue of reducing the number of interfaces without recognising that some interfaces are more important, or critical, than others. It is a contention of this thesis that functional interactions can be classified in a way that indicates their importance in architectural terms.

The literature search has shown that interfaces have been characterised by what is transferred across the interface (energy, material, information, forces) or by spatial dependency (Steven D Eppinger & Pimmler, 1994)(Sosa, 2003). In the majority of cases researchers have attempted to assign a somewhat subjective numerical importance to each (Sosa, 2003) (Yassine et al., 1999). Importance of an interface is made without knowledge of its intricacies and hence challenges to the system design. Suh's stipulation that a feedback relationship is not considered acceptable for an external interface (Suh, 1997) is an attempt to understand such functional intricacies in terms of the effect it should have on system architecture. Different types of system provide candidates for the potential intricacies of interface. This research has identified that control systems, service systems, decision systems, command systems, critical systems and soft systems all have functional constructs types that need to be considered or prioritised. In reality a system design often incorporates more than one of these functional constructs, which will be defined as '*functional interaction types*'. They represent types that help to understand how the required functionality will be achieved, how it will behave and how it can be analysed and managed. They are important to the systems designer in order that he/she can formulate logical system concepts and differentiate between them on the basis of their interaction difficulty or importance.

The following types have been identified:

- Chain interaction type – the appropriate execution and performance of a sequence of bespoke activities
- Loop interaction type - control of a parameter or property according to a demanded value
- Service interaction type – external provision of activity or resource according to agreement to potentially more than one client
- Judgement type - determination of course of action based on various sources of available information
- Human issue – issues of human interaction that influence the way that a system should be designed
- Physical Interface type - functionality associated with interconnection and often transfer of energy

For all of these types, an architectural partitioning of the system needs to avoid the inappropriate division of functionality.

A *chain interaction type* is defined as the appropriate execution and performance of a sequence of bespoke activities. In a chain, it is important that the integrity of the chain is maintained, which could be viewed as a required reliability or trust between individual links in the chain. Threats to the c*hain interaction type* would be a fault, interruption or unacceptable delay. Two subtypes have been identified and these are *critical chains* and *functional chains.* A *critical chain* requires an automatic and immediate response between an activity and its predecessor, regardless of other events, requiring singular purpose and priority in design. Creating an architectural interface along a *critical chain* would introduce complication to its behaviour: what potential delay might be introduced; what impact would this have; what would be the impact of a failure? If chosen in the wrong place it can lead to the creation of a complicated interface, making the specification of the required performance difficult to manage. Without appropriate safeguards, these chains will be mission or safety critical and therefore affect the reliability and safety characteristics of the system. A *functional chain* on the other hand consists of a set of functions where the dependency is ‘pull’ in nature (i.e. on demand), rather than the ‘push’ of a *sequence chain*. The pull nature reduces the degree of coupling and makes it amenable to system partitioning, but the functional association is one that should discourage partitioning across organisational boundaries, as the resulting functional allocation can be difficult to manage between organisations. Examples of chains in systems design are mission chains, supply chains, failure chains and safety chains.

A *loop interaction type* is a set of activities that control a sequence, parameter or property according to a set criterion or demanded value. Poor loop performance can lead to instability, inadequate response to events or residual errors. There are two sub-types that require different considerations, the first being the *control loop* where it is inadvisable to create an architectural interface within a highly performing control loop as delays introduced are likely to cause instability and affect performance. In missile design, this issue can be encountered when considering the centralisation of inertial measurement in a single Inertial Measurement Unit. This often requires communication of measurements across a databus to 'distant parts' of the missile and this process can introduce unacceptable latency in the data. As a result, certain inertial instruments may be duplicated in the distant location to "shorten" the loop, improve response and reduce latency. The second sub-type is the *on-condition loop*; functions that are logically connected by the need to fulfil a condition before an activity can or should progress. The need to consider them together is one of considering overall loop performance. Meeting the loop performance is normally one of adequate budgeting rather than continuous monitoring and changing loop parameters (as is the case with the *control loop* type). This makes it amenable to system partitioning, but the need to budget would favour constraining the design within a single organisation. Examples of loops within system include feedback loops such as guidance loops and control of system environmental factors.

The *service interaction type* is an external provision of resource provided in accordance with an agreement and has potentially more than one client. A service is designed to comply with a predefined and agreed level specified by another organisation and therefore this is logically amenable to both system and organisational partitioning. However, where there is more than one client, this requires planning to ensure sufficient capacity. Considerations for the service interaction type are timeliness, availability, capability to provide the service and flexibility to interface tolerances. Threats to the type are shortage of resource, untimely provision and difficulties in planning. Examples of services in systems are in resource management, maintenance and 'handover basket' strategies; the latter is one where a set of tolerances around a required value are provided in order that a critical dependency is avoided i.e. any value within tolerance is acceptable, which is then more amenable to non-bespoke solutions.

The *human issue* type includes issues of human interaction that influence the way that a system should be designed. Human issues require different approaches such as that of Soft Systems Methodology (Checkland, 1981a), which represents a different approach to the more traditional engineering methods. However, soft issues still manifest themselves as interface challenges. Waring identified several common issues (Waring, 1996) of which conflict, pressure, solidarity and knotty problem all represent different interface problems.

In architectural terms, parties that are in conflict should normally be kept apart to avoid the conflict escalating. Both pressure and knotty problems represent those that are difficult to resolve – complex/complicated interfaces that require scrutiny and cooperation; architecturally these are entities where it would be inadvisable to design or organise them apart from each other. However, solidarity is an indicator of single purpose and agreement – often parties in solidarity and with strong agreement allow them to be easily considered across organisational boundaries.

The *judgement type* recognises the decision making element of a system (most likely human) and critically includes not just the decision-making element, but also the availability of suitable information required in order come to an appropriate decision. Such judgements are encountered in command and control structures, where appropriate situational awareness will be required to make the decision. Appropriately accurate and timely information will be required to make good decisions and so this is an issue that should be addressed architecturally.

The issues raised in the consideration of these *functional interaction types* are undoubtedly important in the architectural decisions that a system designer has to make. In a system design process, it is not sufficient to identify functions in terms of their sequence and dataflow, but there is also a need to consider them in terms of their *type* and the challenges that this can infer. Such a determination will help to guide the choice of system architecture by preserving elements that require close coupling, whilst allowing more freedom in the remaining cases where close coupling is not needed.

5.2.2 Partitioning by functional interaction type

Morris (Morris & Parnas, 1971) recognised the need to keep certain interfaces internal to subsystems (encapsulation), which requires an identification of what these interfaces are. Table 10 takes each of these types and summarises the ones that are amenable to partitioning in a systems design, and which ones aren't, based upon their behavioural intricacies. As identified above, there are some types which are amenable to partitioning from a behavioural perspective, but where the need to coordinate the design means that it is advisable to keep the design activity within the same organisation structure in order to provide appropriate management to provide a coherent design.

Whilst at this stage purely functional drivers are used to advise on partitioning in physical and organisational domains, appropriate management of functional drivers can also be envisaged to address non-functional performance, as the *functional interaction types* allow the appropriate management of behaviour across interfaces and hence promote benefits in terms of such quality attributes as performance, reliability and safety.

There is also legitimate concern over how different lifecycle stages should impact the system architecture, especially as organisational boundaries can be expected to vary over a system lifecycle. In industry, movement from one lifecycle stage to the next is often dealt with as a service type of relationship (for instance a transfer from an engineering organisation to a customer support organisation) which helps to ensure that it is amenable to partitioning, but it is in danger of ignoring benefits that might be achieved for instance by feedback of in-service experience into an evolving design baseline; this is often referred to as “over the wall” (Steven D Eppinger, 1991)(Loch & Terwiesch, 1998).

Table 10 summarises the *functional interaction types* and the advised rules to managing the interfaces according to the earlier discussion. Three columns are provided: a) interactions that require separation; b) interactions that should be kept in the same physical subsystem; and c) interactions that should be contained in the same organisation. When a functional design has been devised (in accordance with stakeholder requirements) then the system designer can use these rules to decide on appropriate subsystem boundaries and organisations in order to maintain appropriate functional interactions in the architecture.

As described, these definitions allow the following categories for characterising a system architecture with a view to functional partitioning:

- *Unsuitable interactions* – situations where functional interaction types suggest that a system interaction should be separated, which is applied to *human conflict issues* and the advised ‘un-sharing’ of *shared services*
- *Fundamental blocks* – functional interactions that suggest a critical dependency that should be kept together (push chains, control loops, shared services and complex/pressured human issues)
- *Organisational constructs* – coupled functions that benefit from organisational structure (functional chains, on-condition loops and judgements)
- *Partitioning points* – functions interactions that involve a natural break in cause and effect so are more amenable to partitioning (exclusive services, solidarity and agreement)

Table 10 Definition of Functional interaction types

Type		**Definition**	**a) Require separation ?**	**b) Keep within subsystem**	**c) Keep within organisn**	**Additional advice**	**Mitigation**
Service	Shared	*A non-exclusive functional relationship that is made available to others according to an agreement*	√	√	√	Ensure availability and capacity given conflicting demands of multiple users. Minimise sources	Overcapacity
	Exclusive	*Exclusive provision at any one time of an agreed function*	X	*x*	*x*		N/A
Chain	Critical (or Push)	*A prescribed and automatic sequence of functions*	x	√	√	Keep chains short, consider in parallel. Approach depends on failure probability and impact	Redundant failure mechanisms
	Functional (or pull)	*A functionally dependent association*	x	*x*	√	Keep chains short, consider in parallel	Establish performance budgets
Loop	Control	*Control of a parameter based on feedback based on the value of the output*	x	√	√	Ensure requisite variety, stability depends on response	Stability of system to be established
	On-condition	*Control of activity according to a set condition*	x	*x*	√	Balance loops to manage flows	Establish performance budgets
Human issues	Complex (knotty), pressure	*A function that is provided to address a human issue*	x	*x*	√		-
	Conflict		√	*x*	*x*	Establish precedence	Arbitration
	Agreement (+solidarity, Trustworthy)		x	*x*	*x*		N/A
Judgement		*A function where choice is made between options on a way forward*	x	*x*	√	Ensure adequate situational awareness, sufficient options and appropriate quality of information.	Clear procedure on available options

The definition of *functional interaction types* is aimed at providing a novel way to help the system designer to establish where the partitions and boundaries of the architecture should be. In the literature, a decoupled or independent architecture can help to deal with complication by simplifying its interactions. However, it is also clear that and uncoupled or fully independent system is often not desirable; a system is often looked on favourably as “more than the sum of its parts”, but full independence would entail a product that is *only* the sum of its parts. Orton describes the appropriate use of independence as the pursuit of the “loosely coupled” system. The system designer’s role is therefore to ensure that elements of the system are decoupled and independent where the benefit outweighs any corresponding loss of opportunity. Whilst current architectural techniques focus on determining where architectural boundaries should be placed, *functional interaction types* also enable the designer to decide where boundaries should not be placed. The approach of only constraining decisions that are of a key architectural significance is supported by Tyree and Akermann in their paper (Tyree & Akerman, 2005).

In the first step of systems design the system functions are identified and to create a modular architecture it is usually possible to identify closely related functions that can be analysed independently. If a critical dependency is allowed between groups of related functions, then the analysis and management of the function becomes more difficult. Subsequent allocation of functions to subsystems may cause critical dependencies across physical boundaries, which complicates the specification of the interfaces and subsequent integration. The partitioning of *fundamental blocks* and their *critical interactions* across functional boundaries (i.e. between *functional chains*) and across subsystem interfaces should therefore be avoided.

The above strategies based on *functional interaction types* should be considered by the system designer in creating an architecture. Having created a view of the functional interactions, *unsuitable interactions* should be examined first and *shared services* should be avoided or turned into individual *exclusive services* where possible. Then *fundamental blocks* should be identified so that they are not inappropriately partitioned in either the physical design or within the development organisation. *Organisational constructs* can then be identified as groupings of the functional design that need to be developed together. Finally, *partitioning points* can also be considered when deciding on the system architecture.

Coupling isn't just a static concept and there are dynamic and unpredictable dependencies that need to be considered at the interface that can provide both functional and non-functional benefits:

- Independence and failure: longer chains make the task of system design harder as it reduces the options available to partition the design. Benefits can therefore be achieved by reducing chain length where possible. Where *critical chains* have to be split it should be ensured that no single fault failure along the chain should result in a failure at a physical subsystem interface
- Parallel activities: dependence is created if activities are carried out in series. Therefore, activities that are not related in a chain should be considered to be in parallel where possible
- Balance activities: activities designed in isolation may operate at different rates which can result in a mismatch in flows that can cause either a build-up of stock or a failure to supply. Service agreements will need to be put in place, and the provider will need to allocate an appropriate stock level as part of the agreement

5.2.3 Identifying Functional interaction types in a system

In order to help the system designer to identify *functional interaction types* in a system, are there generic types that occur within systems in general? For instance, partitioning by the rules of the previous section we would expect that there should be external interfaces of the following types:

- Exclusive services
- Human relationships of agreement, solidarity and conflict

Within a given organisation interfaces for the following might be anticipated:

- On-condition loops
- Functional chains
- Complex human relationships
- Judgement

Terms such as "supply chain" and OODA loop (Observe, Orient, Decide, Act) suggest that there might be generic instances that a system designer could search for. In order to identify these instances, a generic view of a system could enable such interactions to be identified. The Generic Reference Model (GRM) identifies a generic and complete set of functions that any system requires, comprising Mission, Viability and Resource management functions. Stakeholder

needs are used to identify Mission functionality and further system functionality is generated for system viability and management of resources. Its focus on producing a complete functional picture means that it offers the prospect of forming a complete design, but can also be used to distinguish *functional interaction types*.

Hitchin's GRM, firstly can be used to distinguish between functionality driven by external influence and that generated from internal needs. These different functional areas are shown in Table 11.

Table 11: Internally and externally stimulated functionality of the GRM

	Internally stimulated	Externally stimulated
Mission	None	Inputs, cooperation
Viability	Synergy, maintenance, homeostasis, evolution	Evolution, survival
Resourcing	Store, distribute, convert	Acquire, dispose

Already, the division between internally and externally stimulated functionality helps to identify the service types, as an external interface to a system will be under the responsibility of a different design authority and should not be allowed as a critical interaction. To gain further benefit from the model, further interactions need to be identified - Hitchins has not explicitly identified all of the relationships and their specific nature as he uses the model as a model of functional completeness rather than one for analysing functional structure. For this research, functional relationships within the model are explored, building on Hitchins's work to represent the causal dependencies in Figure 24. The causal loop notation facilitates the visualisation of *loops* and *chains* in particular. The behavioural model, by its nature, involves decisions or *judgements* that are readily identifiable as dependencies to the rest of the model. Finally, as mentioned earlier, *services* will typically be identified at external interfaces of the model.

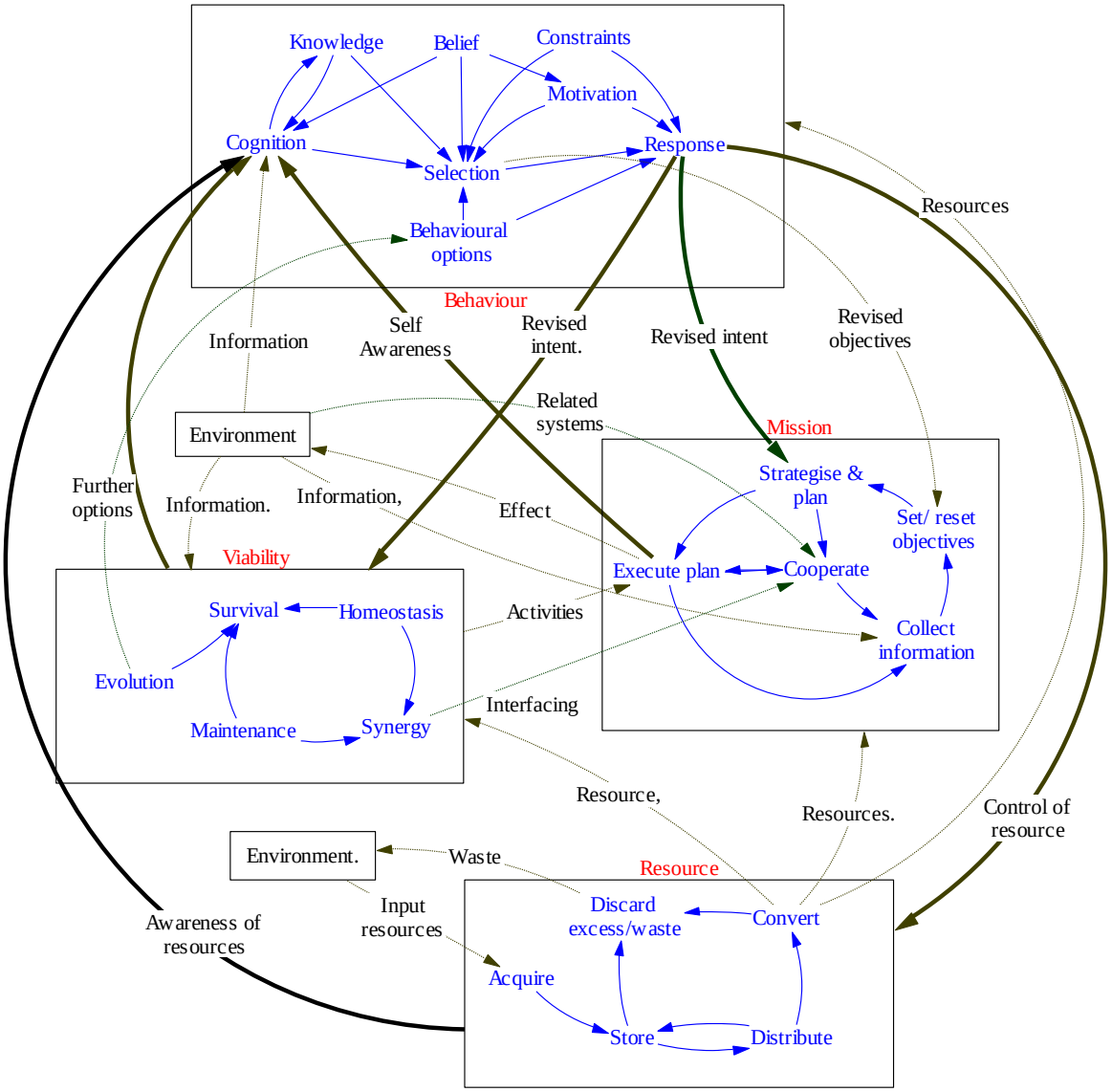


Figure 24: Influence diagram of relationships of Hitchins' Generic Reference Model

Some *functional interaction types* can be seen to relate very strongly to certain elements of the model as outlined in the Table 12.

Table 12: Relationships between functional interaction types and the GRM

GRM element	GRM sub-element	Corresponding Functional Type	Typical question to help identify functionality
Behaviour		Judgement	What decisions are made?
Mission	Information	Service with wider system	What are the information requirements?
	Cooperation	Service with wider system	What external service agreements are required?
	Objectives, strategy, execution	Functional Chain	What chain of activities needs to be performed to complete mission?
		Loops (OODA)	What decisions involve feedback of mission outcomes?
		Judgement	What decisions are made as part of mission?
Viability	Evolution	Service	What is lifecycle management or impact of environmental threats?
	Survival	Chain	How to respond to urgent threatening events?
	Homeostasis	Control loop	What internal conditions need to be controlled?
	Maintenance	Service	What demand is there for maintenance activity?
	Synergy	Functional Chain	What chains exist across subsystems?
		Control loop	What loops exist between subsystems?
		Judgement	What decisions are made for reasons of synergy between subsystems?
Resource Management	Acquisition, Disposal	Service with wider system	What is relationship/ constraints with wider system from supply?
	All	Chain	How is supply managed from acquisition to disposal
		Judgement	What decisions are made about resource management?

Such a table can therefore be used to help check that *functional interaction types* have been comprehensively identified within the design.

5.2.4 Applying function interaction types

The concept of f*unctional interaction types* provides rules that a system designer should follow in partitioning a system architecture. According to the discussion of section 5.1, the system architect should first create a meaningful architecture from a functional point of view and then, in the interest of maintaining functional independence, try to preserve this in the physical architecture. The development of the functional architecture is then achieved by the identification of *functional chains*, defined in Table 10 as a functionally dependent association. A *functional chain* will consist of tightly coupled functions identified by clustering techniques such as N^2 or DSM. Achieving the required system functions is key to meeting the system stakeholder needs and therefore to the system development. The functional architecture is also key to determining the behaviour of a system and the system designer should ensure that *unsuitable interactions* and *fundamental blocks* are preserved where possible by firstly the *functional chains* and subsequently by the choice of subsystem boundaries.

If it is not possible to manage the *unsuitable interactions* or observe the *fundamental blocks* in all cases, which may be anticipated in a real design, then guidance is needed for the system designer on what action to take. When using DSMs, guidance is implicit in the clustering process, but this performs a somewhat arbitrary and subjective assignment of weights to each interface based upon its complexity or criticality. Suh takes a different approach with an analytical technique for applying his Information axiom. The basis of his concept is that there can be a parameter associated with any interface, *information*, that can be calculated and that the greater the information the more critical that interface is. However, this is often not an intuitive measure and the threshold at which an interface conveys too much information to be decoupled is not, and cannot be objectively determined (Suh, 1990). Therefore, in the literature there seems to be no such guidance on when decoupling might be performed across a critical interaction. For this research, a *critical interaction* is one where the needs of managing *unsuitable interactions* and *fundamental blocks* cannot be respected; reference to the earlier Table 10 identifies these as *shared services*, *control loops*, *critical chains* and *complex and conflict human issues*. The different *function interaction types* exhibit different behaviour and therefore it should be expected that they will require different criteria to evaluate whether decoupling can be applied; decoupling should only be accepted where a clear and manageable solution exists. Suggestions of solutions for each type is dealt with individually below and the case presented for each:

- *Shared service*: The capacities of resources supplying the service should be sufficient to support a viable service in a worst case scenario.

- *Control loop*: Margin of stability is acceptable. Such a margin could be judged by sensitivity studies using initial simplified representations of the control systems employed.
- *Critical chain*: Redundant mechanisms need to be in place for the event that a chain is compromised by failure (thus removing the need for detailed failure investigation), with a priority interrupt functionality designed at the interface to ensure timely response.
- *Human conflict issue*: An agreement to an independent and binding, arbitrated solution is required.

In such cases, the issue with splitting a *fundamental block* is mitigated and this should be taken into account in an evaluation of the architecture, on a case by case basis.

5.3 System Design: Physical

5.3.1 Architectural approaches to Physical design

There are several factors that should be taken into account in influencing the physical architecture of the system. Firstly, the physical architecture will receive benefits from addressing the functional drivers outlined above; the benefits of which will be to greatly facilitate the physical design process by designing-out unnecessary complication. In doing this it may be possible to combine or modularise some of these functional elements; such benefits may include cost savings and increased reliability due to a reduction in parts. There may also be groupings of functionality that make sense from a physical rather than a functional perspective. For instance, if a number of chains call upon the same functionality, consideration can be given into creating that functionality as a *shared service* (with appropriate mitigation).

Benefit can also be gained from using principles discussed by Suh (N P Suh, 1990; see 3.2.4.1); the implications of these concepts and corollaries are restructured below in a way that is easier to reflect in design practice, as follows:

- Remove unnecessary functionality
- Specify largest acceptable functional tolerances
- Streamline both parts and interfaces of the system (a reduction in parts that as a result increases coupling and therefore interfaces, should be avoided)
- In evolving the design, focus on existing functions that are both useful and proven

- Choose design solutions that are simple to represent and make

These are very practical aspects of design and their implications are clear – the design will become less complicated and there will be less interactions that the architecture needs to address; the concept of simplicity is one of preparing the design to be partitioned. Emerging from analysis of all of these architecting principles are two types of approach; firstly the design of the artefact themselves and secondly the way that they are allowed to integrate.

Strategies that influence the design of artefacts are:

- Specify largest acceptable tolerances (artefacts are less reliant on interface quality)
- In evolving the design focus on existing functions and solutions that are both useful and proven (use mature and understood building blocks)
- Choose design solutions that are both simple to represent and make (less complicated by design)

Strategies that influence integration are:

- Independence (mapping of solution to function is simpler and so, therefore, is integration)
- Removal of unnecessary functionality (less functionality requires less integration)
- Streamlining parts and influences of the system (less parts require less integration)

Apart from the benefits from functional design there can also be benefits attached to strategies from the arrangement of internal elements and arrangement with respect to external elements. Whilst it is almost certainly not possible to determine an architecture that can guarantee that a design has particular quality attributes, architectural strategies might be employed to promote quality attributes and improved effectiveness of the design. This is addressed in the following section.

5.3.2 Architectural strategies for improving quality attributes and achieving effectiveness

Although formal architecting techniques are often not employed in practice, when they are, they focus upon the reduction of functional coupling when partitioning the design to the physical subsystem structure. Current approaches are of limited effectiveness as:

- Methods for identifying functional coupling tend to be simplistic (as discussed earlier)
- The design is considered only in its broadest sense as a functional to physical mapping; other 'architectures' could be used to control a systems performance in terms of safety, reliability, security, thermal properties etc.

The use of *function interaction types*, provides a means to avoid the oversimplification of the first of these points. The second point has been discussed by others (Wijnstra, 2001; Woods & Rozanski, 2005), the latter proposes an 'architectural perspective' as a collection of guidance on achieving a specific quality attribute in a system. The guidance however is a set of guidelines and best practice rather than a set of architecting principles. Klein (Klein et al., 1999) suggests an approach of developing attribute based architectural styles. To establish these styles, activities required to improve each quality attribute are examined in order to establish useful principles and architecting techniques. This approach, apart from identifying important principles and techniques for a given attribute, can also help identify those that are common across many attributes. In this way the system designer can ensure that all relevant impacts of techniques being performed in pursuit of a specific quality attribute can be recognised.

Various practitioners/researchers in this field have identified the need to "design for" certain design attributes. Suh talks about designing for manufacture (Suh, 1990) and Ulrich and Eppinger about design for production (K. Ulrich & Eppinger, 2008). Wasson takes a much more comprehensive view, suggesting that there are strategies that enable design for comfort, interoperability, reliability, availability, portability and more (Wasson, 2006). However, not all of these strategies relate to the design of the architecture. In order to understand the potential benefits, the nature of design and how it contributes to effectiveness need to be determined. A means of achieving this in practice was to ask four questions about design practice for each required attribute. These are summarised in Figure 25 as:

- Enabler: architectural strategies that will result in an improvement in effectiveness

- Constraint: constraints that may prevent an improvement in effectiveness
- Potential additional benefits: what other benefits might be derived from strategies employed
- Potential negative consequences: where design may reduce the effectiveness of the solution in other areas

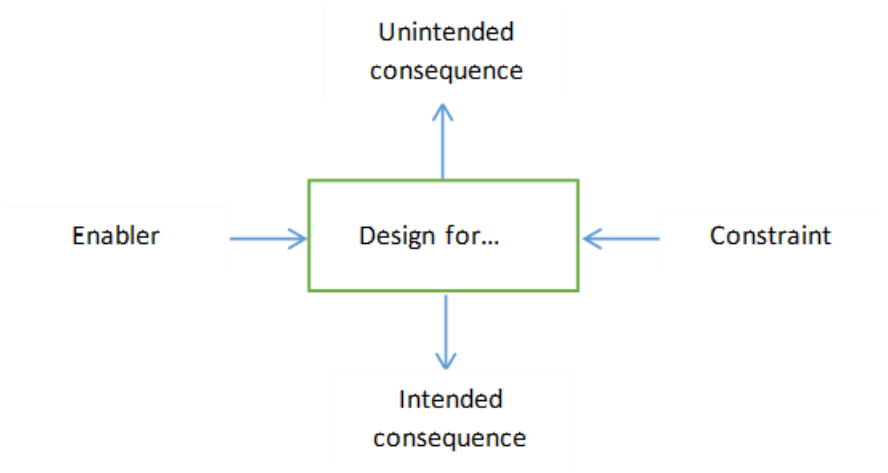


Figure 25: Generic "design for" influence diagram

This has been used to identify aspects of the design that can be used to improve or control the quality of the design, and which of these can be related in some way to architectural strategies. Examination of all quality attributes enables a synthesis of architectural areas/strategies that should be addressed to provide a balanced design. The list of quality attributes examined were those of Mackley (Mackley, 2005) as this was shown in the literature research to encompass all other methods. Where possible, in each case a detailed definition is provided from Wasson's book (Wasson, 2006) which in turn reflect definitions from the US Department of Defence (DoD).

As expected, there are some quality attributes that can be directly influenced by architectural influences, whereas other attributes cannot. It is recognised that such an approach cannot hope to capture all possible architectural strategies for improving system effectiveness, but is a structured and systematic method designed to identify as many as possible.

Design for Reliability “the ability of a system and its parts to perform its mission of a specific duration under specific operating conditions without failure, degradation, or demand on the support system” (Wasson, 2006) is as in

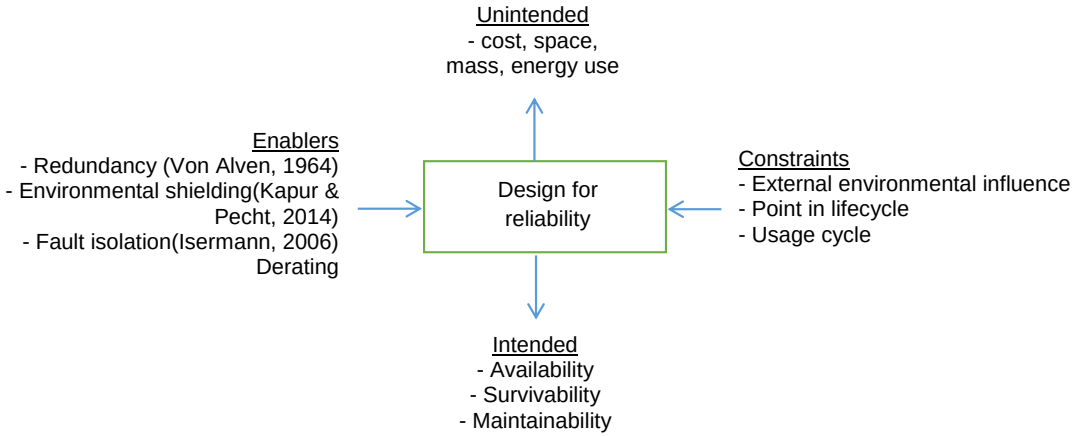


Figure 26.

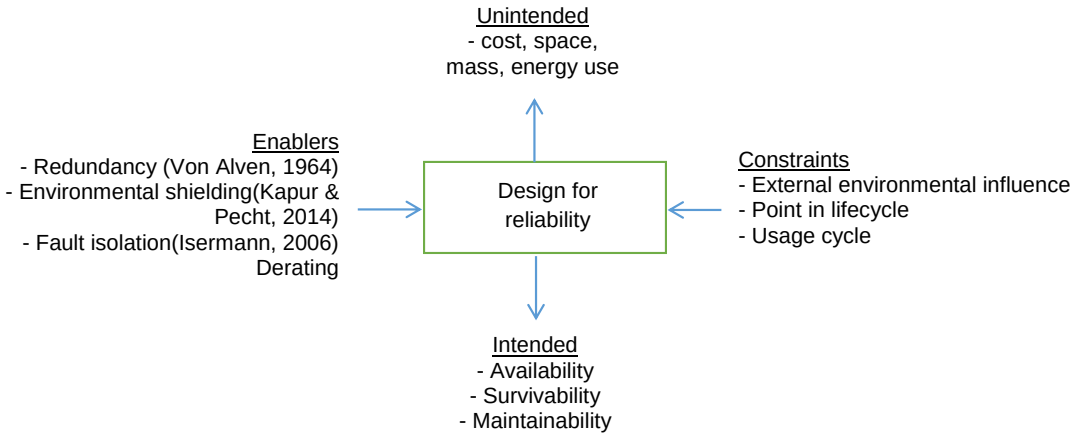


Figure 26: Aspects of design for reliability

From this, appropriate architectural strategies contributing to reliability, availability, maintainability and survivability are:

- Use of redundancy to minimise impact of faults
- Using environmental shielding to protect and extend life of components
- Minimising the potential fault chain to reduce overall impact of an event

Design for Maintainability "the ability of an item to be retained in, or restored to, a specified condition when maintenance is performed by personnel having specified skill levels, using prescribed procedures and resources, at each prescribed level of maintenance and repair" is as in Figure 27.

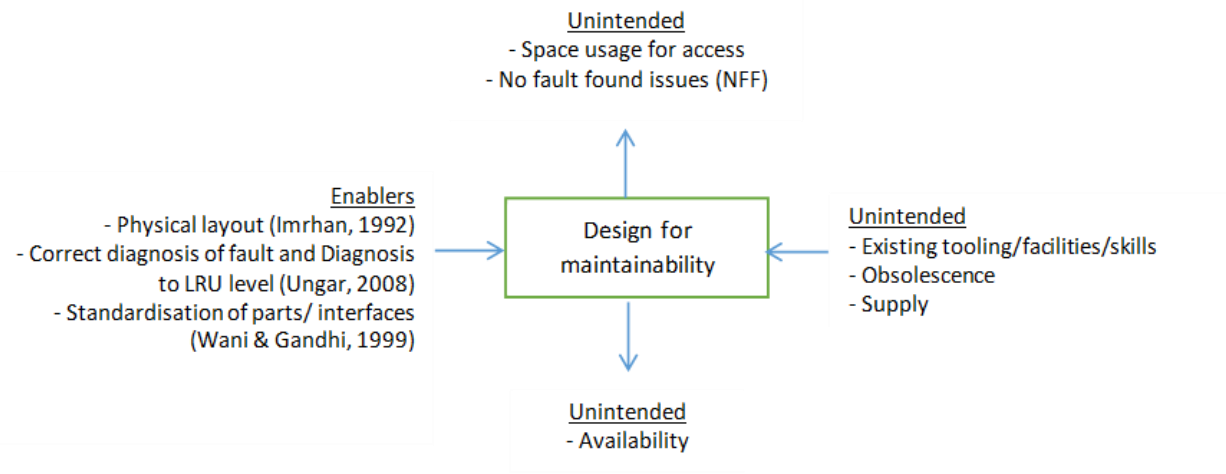


Figure 27: Aspects of design for Maintainability

From this, appropriate architectural strategies are:

- A suitable general physical layout
- Appropriate Line replaceable unit definition
- Use of a Standardised architecture/interface definition
- Supportive supply chain

Design for safety “the application of engineering and management principles, criteria, and techniques to optimise safety within the constraints of operational effectiveness, time, and cost throughout all phases of the system lifecycle” is as in Figure 28.

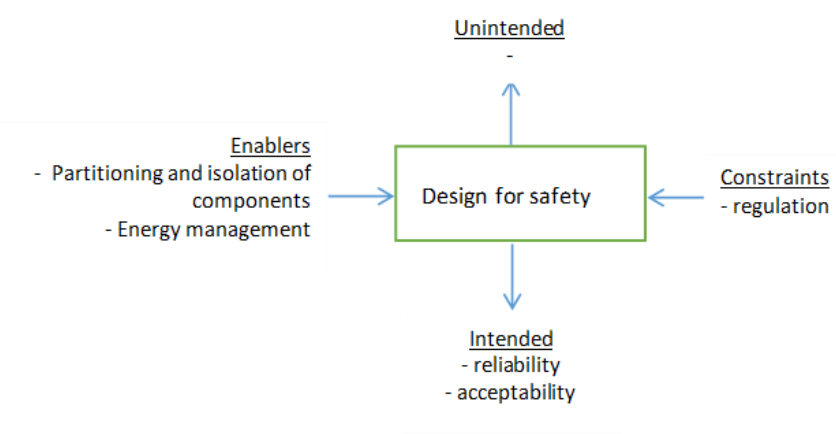


Figure 28: Aspects of design for safety

From this, appropriate architectural strategies contributing to safety are:

- Partitioning and isolation of components
- Energy containment

Design for operability "the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of a system or component (usability)" is as in Figure 29.

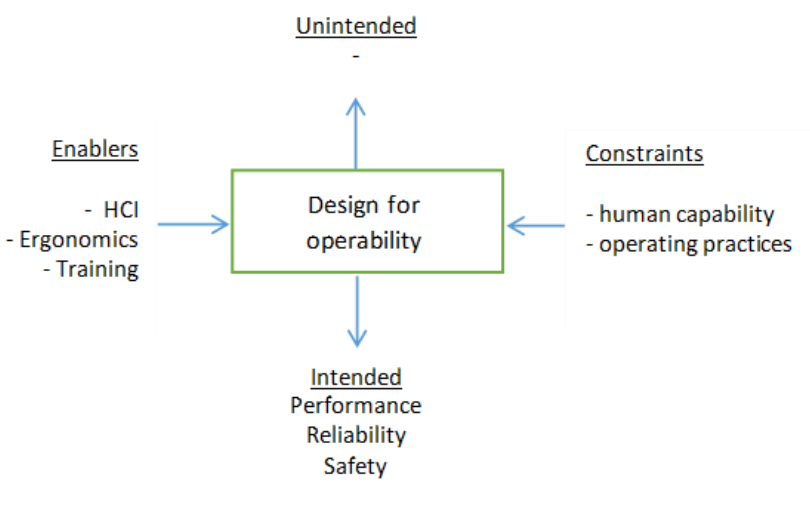


Figure 29: Aspects of design for operability

From this, appropriate architectural strategies contributing to operability are:

- Physical layout for human manipulation
- Presentation of information

Design for compatibility is as in Figure 30.

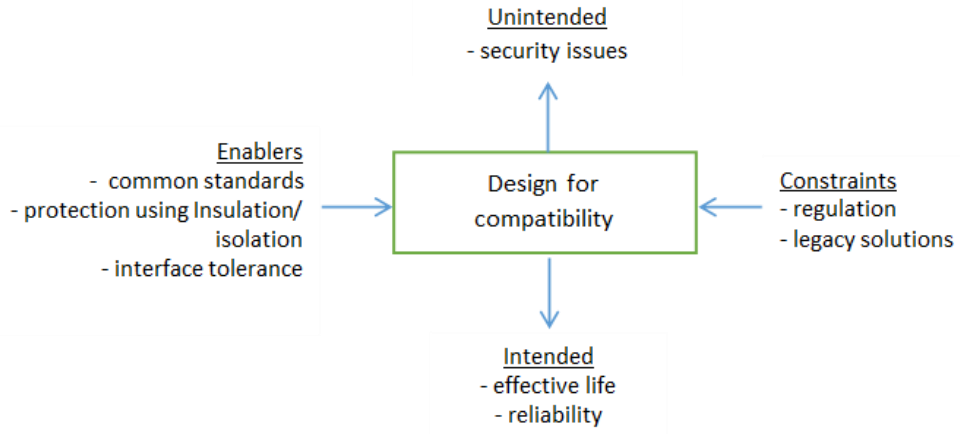


Figure 30: Aspects of design for compatibility

From this, appropriate architectural strategies contributing to compatibility are:

- Standardisation
- Tolerance
- Protection

Design for survivability “the capability of a system and its crew, if applicable, to avoid or withstand a hostile man-made, natural, and induced operating environment without suffering an abortive impairment of its ability to accomplish its designated mission” is as in Figure 31.

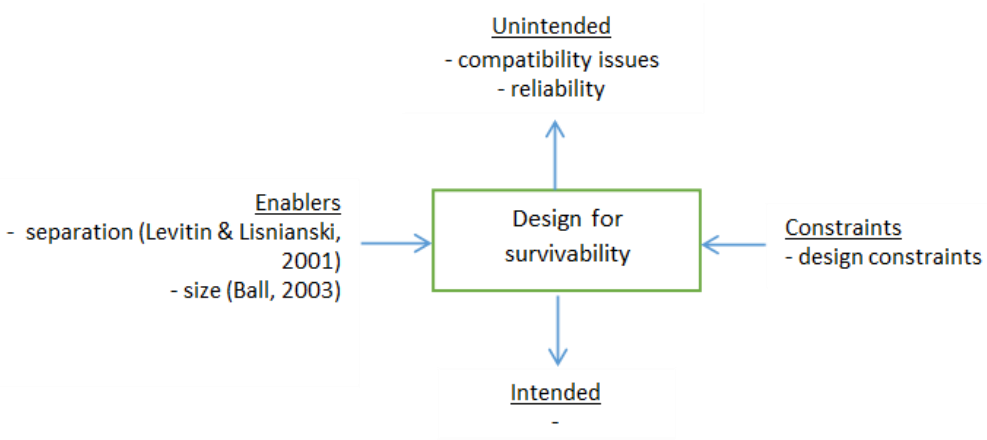


Figure 31: Aspects of design for survivability

From this, appropriate architectural strategies contributing to survivability are:

- Size
- Separation
- Protection

Whilst this is a useful and informative analysis for each attribute in isolation, the question should be asked as to how these architectural strategies can be applied together. There is doubt as to whether all quality attributes lend themselves to the analysis of an architecture. Alexander was concerned about this in his book (Alexander, 1964) as discussed earlier. However, this analysis has shown that there are architectural strategies that can be seen to contribute to various system quality attributes for which examples are identified below:

- Survivability by means of physical separation to protect redundant systems; by compression/size reduction to reduce probability of damage; by insulation against external sources of energy
- Reliability by managing internal sources of energy
- Safety by grouping safety critical items in one place for ease of management and by separation/insulation from sources of energy
- Security by grouping elements in at various levels of security and controlled coupling by access control
- Maintainability by providing sufficient spacing for access
- Environmental compatibility by creating physical separation/insulation or by managing adjacencies
- Operability by close physical design layout

These architectural strategies can be categorised as being of cohesive and dispersive influence[3] (Hitchins, 2008):

- Dispersive influence: system elements can need to be separated or insulated from each other
- Cohesive: system elements can need to be:
    - Close or associated with each other
    - Connected to each other (such as design for energy transfer interactions[4])

Table 13 describes how dispersive and cohesive strategies can be used to influence quality attributes.

[3] Note: N squared analysis may be used to produce a dispersive or spatial assessment matrix with a score of 9=close, 5=no preference, 1=apart/insulated. If the necessary separation cannot be realised then insulation measures will be required

[4] Gu (Gu & Sosale, 1999)maintains that this is a function of the system. This method makes a distinction between functions required for the mission of the system and secondary functionality from specific design features.

Table 13: Design for influences in system design

	Survival	Maintain	Operation	Ext comp	Safety	Reliability	Int comp
Dispersive (separated/ decoupled)	Distant	Distant		Distant or Insulated	Distant or insulated	Insulated	Distant or Insulated
Cohesive (encapsulate-ed)	Close		Close	Close or Conducting			Close or Conducting

To the system designer this indicates that, in considering groupings of components, quality attributes of survival, operation, external and internal compatibility can be influenced. Effective separation between components allows influence of survival, maintenance, safety, reliability, external and internal compatibility. The aspiration is to achieve an architecting process that can aid in increasing the effectiveness of systems design. Alexander suggested that Quality Attributes were too abstract to allow clear cause and effect between design and these attributes; the point he made was that, without a clear link to the benefit achieved, the designer is unable to improve the design. The method described here has focussed on aspects that the systems designer can influence, which will provide the framework for system designs that address desired quality attributes.

5.4 Lifecycle Architectural Influences

From the work of Gu (Gu & Sosale, 1999) we can also generate a set of desired architectural considerations from a Lifecycle perspective:

- Organisation independence (including ownership)
- Production Independence
- Standardisation
- Line/Lifecycle Replaceable Units
- Reconfiguration
- Recycling (including reuse and disposal).

Modularity can benefit in each of these areas, but the benefit is dependent on different, though not necessarily mutually exclusive, groupings as follows:

a) Organisation independence can be helped by grouping similar functions (functional independence) so that they can be developed independently

b) Production independence can be helped by grouping similar technologies so that similar technologies can be produced together

c) Standardisation allows reuse of modules with the associated benefits of scale – standard modules should be used where possible for similar functionality

d) Line replaceable units will be more effective if they contain components that need to be removed for maintenance at the same time

e) Systems can more easily be reconfigured if they are functionally independent as change has minimal impact to surrounding systems

f) Recycling is more easily achieved if similar components or materials can be grouped together.

Bullets a) and e) will be addressed by the incorporation of a modular functional design, which leaves the following that can be used to address the lifecycle dimension:

- Production independence – addressed by grouping similar technologies
- Line replaceable units – addressed by grouping components of similar maintenance policy
- Recycling – addressed by grouping components of similar materials
- Standardisation – addressed by using common modules where possible for similar functionality

However, it may be useful to consider all of these elements to ensure they have been addressed in earlier steps of the process, particularly if there were trade-offs earlier in the process that suggested a compromise to a principle. This would then include:

- Organisation independence – grouping of similar functions
- Reconfigurable – ensuring functional independence

These strategies can be considered to the extent possible at the concept stage and the possibilities for doing this are expected to depend on the particular system in question. However, care has to be taken that such considerations don't then invalidate the architecture already put in place based on functional and physical considerations.

6 EVALUATING THE SYSTEMS DESIGN

6.1 Overview

The appropriate use of modular and independent architectural principles has been shown to offer benefits in terms of operational effectiveness and lifecycle management. The literature search has not identified a method that allows a satisfactory evaluation of an architecture, although parameters have been defined that could be used to determine desirable attributes of such an architecture. This research suggests that important considerations in architecture should be addressed as part of a structured process and that adherence to this process therefore will be an indication of quality. An evaluation could then focus upon how well each step of the process was performed. Necessarily then such a method will be bespoke, but may draw upon suitable evaluation parameters from the literature; in this section such parameters are identified. This approach is one of addressing and improving the quality of the *system architecture*; such an evaluation will still be required. Alexander (Alexander, 1964) suggested that a good architecture will produce an effective design, but that an exact relationship between architecture and achieved effectiveness cannot be determined. It can be concluded that an evaluation of the 'quality' of a system design can be achieved by an evaluation of its architecture, but a further evaluation of achieved effectiveness is also required to establish whether it meets its goals and objectives. In this section I will address how candidate architectures can be evaluated so that different architectures can be compared with each other and establish a best architecture.

6.2 Evaluation of architecture design

The literature has shown various ways of evaluating an architecture in a quantitative sense (see section 2.7). There are methods that provide an overall score for "goodness" and methods that look at more detailed aspects of the architectural properties in an attempt to make a more detailed assessment.

Two measures at an overall system level that were identified in the literature search were Altshuler's *Ideality* (Altshuller, 2002) and Suh's *Information* (Suh, 1990), discussed in section 2.4.2. Both of these parameters attempt to provide a measure of how simple or complex a solution is. Altshuler's Ideality is useful in principle, in that it highlights the benefits of a solution that promotes useful functions over unnecessary or harmful functions and favours a solution where costs of functionality are minimised. However, the need to evaluate costs is difficult to satisfy in initial concept design. Suh's *Information* is equated to the sum of the probabilities of satisfying the functional requirements. Calculation of *information* involves the setting of an allowable tolerance in achieving a function

and calculating the probability of meeting it. Such a probability is often difficult to determine and the importance attached to achievement of all functional requirements cannot be considered as being equal.

N^2 and Design Structure Matrices are useful tools in the analysis of architecture and various techniques have been developed to judge a system based on the way components are arranged in clusters (Section 2.7.1). Establishing the "energy" of the matrix can be a very quick and easy parameter to calculate and gives an indication of the coupling of the system.

Further useful insight into the complication of an architecture can be achieved through the examination of "visibility" and "dependence" (Sharman, 2004). By looking at the incoming and outgoing flows of a component it is possible to gain some insight as to how modular it is. It is also able to identify system input and output components, whose contribution to the structure of the system is often difficult to influence. Whilst Design Structure Matrices can usefully show clusters within an architecture, it often become more difficult to interpret for large amounts of components. The visibility vs dependency diagram can provide a view of the contribution to a modular design of each component that is much easier to interpret. A modular system, will have components that are minimally visible and dependent internally, but may have input and output modules that are highly externally visible and dependent respectively.

A more sophisticated way of measuring the modularity of an architecture involves the calculation of three parameters; degree modularity, distance modularity and bridge modularity (Sosa, 2007):

- Degree modularity is defined as in-degree modularity (the number of components depended on) and out-degree modularity (the number of components that depend on it). These relate directly to visibility and dependence respectively and whereas the latter provide a useful visual evaluation, degree modularity provides a single overall modularity measure. *In-degree modularity* and *out-degree modularity,* appear to be equivalent concepts to Sharman's dependence and visibility. *Degree modularity* is useful in determining the complication of coupling within a system, but it does not easily deal with individual interfaces of varying complication/complexity; Sosa does propose the concept of applying weightings to interfaces, but the choice of weighting is arbitrary and left to the designer.

- Distance modularity is a measure of modularity from a separation perspective. It evaluates the number of steps there is between one interconnected subsystem and another and therefore records how many subsystems are in the interaction path. Interestingly, the greater the *distance disconnectivity* the more modular a system is meant to be, but equally the greater chance of a single event propagating through a system. Whilst d*istance modularity* recognises the importance of modules being separated from each other, modularity is as much about keeping the right elements together as it is about keeping others apart i.e. keeping certain components together in modules that are then separated. For instance, it is often desirable to separate redundant systems to avoid both being damaged in an attack, but improved survivability can also be achieved by reducing the presented area to a threat.
- Bridge modularity refers to the number of times that a component lies on the optimal path between two other components. This is important as the failure of or subsequent removal of the intermediate component can prevent the interaction. *Bridge modularity* is useful as in indication of a systems ability to accept change as it evaluates how many system components are likely to be affected in some way. Such a situation would occur if a unit was removed and replaced with another, functionally similar, but not physically identical component – would the flow still be allowed to pass through?

Complication may be driven by a few critical interfaces, as indicated by the concept of *fundamental blocks.* It is possible to assign values to particular interfaces to apply some indication of importance or priority and various proposals exist for this (Yassine et al., 1999), (Steward, 1981). However, the assignment of these weightings is at best by good judgement and therefore providing an objective analysis of the results is difficult. Various authors (Sosa, 2003) (Steven D Eppinger & Pimmler, 1994) discuss breaking down the problem to analyse the quality of the architecture in terms of its interaction type; spatial architecture, as well as structural, energy, information and material flows. Whilst these are without doubt important distinctions for the development of architectural properties, their individual merits cannot be considered comparable. Firstly, they are not independent (e.g. spatial separation will impact on structure as will energy impact on material flows), and they therefore cannot be combined to create a single order of merit. Secondly, there is no method proposed to allow an objective comparison of the benefits of the parameters, only a qualitative or subjective assessment can be made.

The existing candidates are compared with the steps of the Critical interaction modular design methodology to examine how they might be used to evaluate the effectiveness of a candidate architecture in Table 14.

Table 14: Existing measures of modularity compared with Critical interaction modular design methodology steps (elaborated in Chapter 7)

	Candidates	Comment
Step 1	-	No candidate
Step 2	Ideality, Information, Energy (N^2)	Ideality and Information are not tangible parameters at concept level, but Energy can give an overall system indication
Step 3	Degree modularity, Dependence, Visibility, Energy (N^2)	Can be applied separately to flows (spatial, structural, material, energy, information), but does not recognise relative importance of interactions. Energy can be an indication at system level
	Distance modularity	Does not address issues of physical distance
Step 4		Useful in identifying extent of impact, but not importance
	Bridge modularity	Identifies paths across subsystems, but not importance

Whilst there are no satisfactory options available for the first two steps, it is possible to relate potentially useful measures to steps 3 and 4. Sosa's measures are pertinent, but they do not distinguish between criticality of interactions in an objective way and the influence of spatial separation cannot be measured. The Critical interaction modular design methodology has a way of distinguishing between interactions as it determines those, associated with certain *functional interaction types that* are more difficult than others. This knowledge is used to influence the system design and the organisation associated with them to both reduce complication and manage this when it cannot be mitigated. Whilst it is difficult to compare the various *functional interaction types* associated with *fundamental blocks*, there is a clear increase in their complication compared with other *functional interaction types*. Making Juran's assumption (Juran, 1954), that requires separation of the "vital few" from the trivial, it is proposed that these critical interactions are considered as the "vital few" in terms of increased complication of the system. This provides the opportunity to perform calculations of degree, distance and bridge modularity for critical interactions only – values derived would then indicate the modularity from the perspective of the most challenging interfaces.

The definitions from Sosa (Sosa et al., 2007) are for a given component and here, a measure representing a system of components is required. Therefore taking an

average across all n components i of a system and considering only critical functional interaction types, Equation 2 and Equation 3 can be derived.

Equation 2: Critical degree modularity

$$\boldsymbol{Critical\ degree\ modularity, Cdm} = \overline{Cdm_i}$$

$$\boldsymbol{where}:$$

$$Cdm_i = 1 - \frac{\sum_{j=1,i\neq j}^{n} x_{ji} + \sum_{j=1,i\neq j}^{n} x_{ij}}{2x_{max}(n-1)}$$

$$\boldsymbol{as}: x_{max} = (n-1)$$

$$Cdm_i = 1 - \frac{\sum_{j=1,i\neq j}^{n} x_{ji} + \sum_{j=1,i\neq j}^{n} x_{ij}}{2(n-1)^2}$$

$$\boldsymbol{and}:$$

$n\ is\ the\ number\ of\ subsystems, i$

$\sum_{j=1,i\neq j}^{n} x_{ji}$ is the $number\ of\ subsystems\ that\ have\ critical\ inputs\ to\ i$

$\sum_{j=1,i\neq j}^{n} x_{ij}$ is the $number\ of\ subsystems\ that\ receive\ critical\ outputs\ from\ i$

Equation 3: Critical distance modularity

$$\boldsymbol{Critical\ distance\ modularity, Csm} = \overline{\boldsymbol{Csm_i}}$$

$$\boldsymbol{where}:$$

$$\overline{\boldsymbol{Csm_i}} = \frac{\sum_{j=1,j\neq i}^{n} d(i,j) + \sum_{j=1,j\neq i}^{n} d(j,i)}{2n(n-1)}$$

$$\boldsymbol{and}:$$

$d(i,j)\ is\ the\ geodesic\ distance\ between\ subsystems\ i\ and\ j$

$n\ is\ the\ number\ of\ subsystems$

However Distance modularity presents a difficulty when used purely for Critical interactions. In considering Degree modularity, the aim was to reduce the inputs and outputs to create a 'score' that is as low as possible; this is then subtracted from the value of 1. A value as close to 1 is preferred and the consideration of only Critical interfaces is consistent with this. However, for Distance modularity, a high 'score', as close to 1 is also preferred, but this is composed of scores from all interactions of all components. Consideration of purely Critical interactions in a modular design would create a low score, even for a modular design. Given that a further measure relating to separation of components will be proposed in Equation 5, it is proposed not to use Critical distance modularity as a parameter.

Bridge modularity requires more careful consideration. Here the calculation is identifying the degree to which a component is a bridge between other interacting components. In order to evaluate this it would need to be shown that there is a critical path existing across the entire length of each interaction. In fact, the critical interaction concept is less valid for consideration of the lifecycle maintenance perspective. The more a component is a bridge, the more its removal and replacement will require test involving other components regardless of whether it is a critical interaction or not, but there is not obviously going to be a difference in the level of disruption compared with other interactions. The equation is therefore for the average number of any interactions, across all subsystems n, in Equation 4.

Equation 4: Bridge modularity

$$\boldsymbol{Bridge\ modularity, Bm} = \overline{Bm_i}$$

$$\boldsymbol{Bm_i} = \frac{\sum_{i=1}^{n} 1 - \frac{\sum_{i\neq a, i\neq b, a\neq b}^{n} \frac{nd_{ab}(i)}{nd_{ab}}}{n[(n-1)(n-2)]}}{n}$$

$\boldsymbol{where}$:

$nd_{ab}(i)\ is\ the\ number\ of\ geodesics\ between\ a\ and\ b\ \ through\ subsystem\ i$

$n\ is\ the\ number\ of\ subsystems$

Measuring the benefits of spatial separation is not trivial as the degree of separation between components and the benefits derived from it will vary significantly when considering different quality attributes. In some cases, the benefit will be roughly proportional to distance and in others there may be a specific separation required for compliance (particularly in the case of safety and security). In this way the problem can be seen as a multi-criteria problem where some parameters are characterised by a goal (goal based approach) and some can be assigned a value (value measurement approach). In the goal based case an architecture can be described as compliant (acceptable) or non-compliant (not acceptable). For attributes that can have a value based approach then it is possible to calculate a value based on physical measurements of the system. For instance, in a later section a case study will be made of a central heating system (section 9.5.3). In this case, there is a need to separate the thermostat from the heat source to allow even full heating of the room and this should be in the furthest corner. It is possible then to assign and optimum value as the maximum distance and determine what proportion of that distance can be achieved in the design (as it may be limited by other factors such as built in wardrobes or windows). Similarly, a need to be adjacent can be recognised by an optimum separation of zero. Hence we have Equation 5.

Equation 5: Dispersion index

$$\boldsymbol{Dispersion\ index} = \frac{\sum_{i=1}^{a} \frac{actual\ dispersion\ distance}{maximum\ distance} + \sum_{i=1}^{b} 1 - \frac{actual\ cohesion\ distance}{maximum\ distance}}{a + b}$$

$\boldsymbol{where}$:

$a = number\ of\ dispersive\ actions\ between\ two\ system\ components\ required$

$b = number\ of\ cohesive\ actions\ between\ two\ system\ components\ required$

These additions give candidate measures for steps 3 and 4, but others are needed to evaluate the first two steps. The first step of the process is to choose a system design strategy that addresses the particular challenges presented by the *context type* of the system. For instance, an architecture that has been designed as if the system was a *unitary* type, when in reality it is a *coercive* type is not going to address all the necessary issues. In this instance, the system boundary assumed is likely to be wrong as the solution is liable to support the dominant stakeholders, but marginalise others; starting with the wrong system boundary would be a poor architecting decision. The correct identification of

problem type and application of a suitable architecting strategy is either observed or not and is given by the Boolean variable of Equation 6.

Equation 6: Suitability

$$\textbf{Suitability} = 1\ (\text{suitable})\ or\ 0\ (unsuitable)$$

The second step determines the functional architecture. It does so by identifying functional flows and then structuring *functional chains* so as to minimise the amount of critical functional dependencies between chains. In reality this can be used to structure the design organisation in terms of functional development, but it also becomes a measure of the complication of the functional design. If functional chains are seen as functional 'subsystems' then *degree modularity* can be calculated for functional chains, as the complication of the functional architecture can be linked to the proportion of interfaces that are critical. Therefore an indication of a well architected design could be:

Equation 7: Critical functional modularity

$$\boldsymbol{Critical\ functional\ modularity, Cfm} = \overline{Cfm_i}$$

$$\boldsymbol{where}:$$

$$Cfm_i = 1 - \frac{\sum_{j=1,i\neq j}^{n} x_{ji} + \sum_{j=1,i\neq j}^{n} x_{ij}}{2(n-1)^2}$$

$$\boldsymbol{and}:$$

$$n\ is\ the\ number\ of\ functional\ chains, i$$

$$\sum_{j=1,i\neq j}^{n} x_{ji}\ \ is\ the\ number\ of\ functional\ chains\ that\ have\ critical\ inputs\ to\ i$$

$$\sum_{j=1,i\neq j}^{n} x_{ij}\ \ is\ the\ number\ of\ functional\ chains\ that\ receive\ critical\ outputs\ from\ i$$

The above equations give different ways of evaluating the modularity of an architecture. Two established techniques are used (degree modularity and bridge modularity) are related to key stages of the methodology; where applicable, the concept of *critical interactions* is incorporated to help establish the level of goodness of the architecture. Further equations are added to evaluate how well an architecture addresses the needs of the context and the balance between cohesion and dispersion.

Measures of critical functional modularity, critical degree modularity and bridge modularity are based on a view that the modularity of an architecture depends on

its components. It is also important for the system designer to consider the external interfaces, especially in a system of systems, where the boundary inherently has more flexibility. Therefore a further measure is proposed which is the system boundary modularity, which determines the critical degree modularity at the boundary only; giving Equation 8.

Equation 8: System boundary modularity

$$\boldsymbol{System\ boundary\ modularity, Sbm} = \overline{Cfm_i}$$

$$\boldsymbol{where}:$$

$$\boldsymbol{System\ boundary\ modularity, Sbm} = 1 - \frac{\sum_{j=1, i\neq j}^{n} x_{ji} + \sum_{j=1, i\neq j}^{n} x_{ij}}{2(n-1)^2}$$

$$\boldsymbol{and}:$$

$$n\ is\ the\ number\ of\ related\ systems, i$$

$$\sum_{j=1, i\neq j}^{n} x_{ji}\ \ is\ the\ number\ of\ related\ systems\ that\ have\ critical\ inputs\ to\ i$$

$$\sum_{j=1, i\neq j}^{n} x_{ij}\ \ is\ the\ number\ of\ related\ systems\ that\ receive\ critical\ outputs\ from\ i$$

Arguably the benefit of the Critical interaction modular design methodology is in the application of a structured process, where the efficacy of each step can be evaluated using the associated parameters. However, if each parameter can be argued to be independent of each other then it would be possible to evaluate an aggregate score of all parameters to provide a best solution. The parameter for each step can be considered to be independent if the information given by one parameter does not give any information on the value of the other (Hyvärinen & Oja, 2000). Therefore, for each parameter in turn:

- *suitability* is only a qualifying parameter
- Critical degree modularity and System boundary modularity: the choice of functional chain boundaries is made independently of the boundaries of the system and subsystems and even with no critical dependencies between *functional chains* (low *critical functional modularity)* it is still possible to have many critical dependencies between system/subsystems or indeed none

- Bridge modularity: the choice of both functional and physical boundaries is independent of the needs for the lifecycle, and it is possible to have many critical interfaces between either functional or physical subsystems with either many or no ‘bridges’ through those subsystems
- Dispersion index: the physical distance between two components of a system is independent of functional or physical boundaries and the space between components is only limited by physical design constraints and the system physical boundary.

In summary, there are various levels of independent modularity measure:

- Problem based - suitability
- System level – system boundary modularity
- Functional chain level – critical functional modularity
- Subsystem level – dispersion index, bridge modularity and critical degree modularity

With six independent evaluation parameters there will be difficulty in formulating an evaluation of architecture for the following reasons:

- there is no way of determining a link between an architecture’s properties and the resulting system’s functional quality and non-functional performance; therefore goals for architectural ‘quality’ cannot be objectively set
- as with most complicated or complex multi-criteria problems, there is no way to objectively determine the relative value associated with any given architectural measure.

The method proposed is one that recognises these limitations, but has itself been validated in many different situations (Kahneman, 2011). The method relies on identifying key measures of goodness that are independent and assigning a score to each; each independent score is a assumed to be important and therefore no subjective weighting is assumed, but the individual scores are added to achieve an overall score. In this case, the relationship between the evaluation parameters and each step of the process means that the evaluation becomes an affirmation that that the approach has been applied, and applied well. Therefore a relative order of merit, the Relative Architectural Score or RAS can therefore be calculated as in Equation 9.

Equation 9: Relative Architectural Score

$$RAS = \{Suitability\}_{problem} + \{System\ boundary\ modularity\}_{system} + \{Critical\ functional\ modularity\}_{functional\ chain} + \{Critical\ degree\ modularity + Dispersion\ index + Bridge\ modularity\}_{subsystem}$$

One possible issue of using this method is that Kahnemann suggests that an evaluation requires an indication of what 'good' and 'bad' are. Examination of each parameter indicates that, apart from the Boolean, system configurations could easily be envisaged that have scores that range all the way from value 0 to 1. With no other information available, an assumption will be made that benefit is linear with the value of each parameter. This assertion can to some extent be tested in the latter case study sections. However, it should be noted that each case study is the result of applying the Critical interaction modular design methodology and therefore by definition should be modular and not displaying the full range of values that may be expected in a variety of designs. Valuable additional research would be to evaluate the scores of a range of integrated designs to establish what might be considered as 'bad' from a modular perspective; this may then be used to calibrate Kahnemann's method as he suggests.

7 THE CRITICAL INTERACTION MODULAR DESIGN METHODOLOGY

To incorporate the architecting strategies discussed previously, a methodology has been developed as part of this research; the Critical interaction modular design methodology. It is composed of five steps, which are shown in Figure 32. Boulding's concept of systems hierarchy (Boulding, 1956), explains that systems can be described as a hierarchy, and this methodology is at the system level, developing the system design concept in order that there is an architecture for the subsequent detailed design. It is specifically at the concept stage and therefore concentrates on the steps leading up to the development and assessment of the architecture, but stops short of the steps necessary to evaluate the design itself.

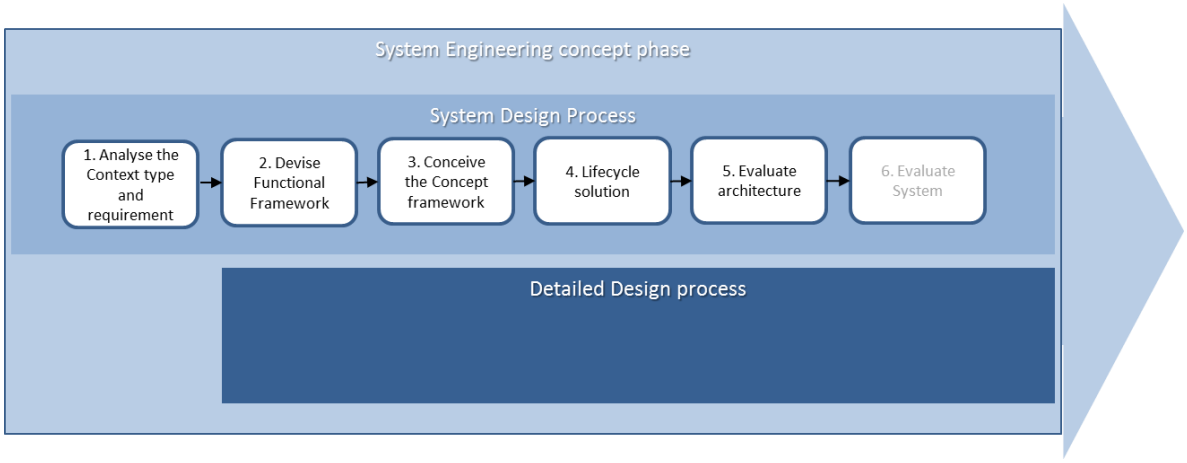


Figure 32 Critical interaction modular design methodology process steps

7.1 Step 1: Analyse the Context type and requirement:

a) Establish *context type* (in order to choose problem solving approach, architectural strategy and risk) (Section 4.3)

Situation type is particularly important as it will indicate the approach to be taken in each of the following steps. These are summarised in Table 15.

Table 15: Approach according to the Situation context type

Situation type	Steps to be followed
Clean sheet	In Step 3, Concept framework, creative options may be proposed providing that they each meet the fundamental blocks
Upgrade	Concept framework is available so reassess functional allocation within existing framework in Step 6
System review	Reassess context as an issue analysis to establish any gaps (Step 1). Considering any gaps, analyse existing functional design to identify any fundamental block violations (Step 2). Based on a modification of current framework as required review concept (Steps 4). Evaluate as before (Step 5)
Reconfiguration	If architecture is not changed then only requires system requalification.

b) Understand stakeholders and environment of the system in order identify all influences and capture requirements:
   - Record the needs of stakeholders and the nature of any human interaction e.g. using a rich picture format
   - Consider objects being acted on and systems interacted with; the latter to identify input, output, control and resources required
   - Consider the impact of the constraints and conditions of the location/environment
   - Record constraints imposed by system level design decisions (mechanisms)
   - Capture using Functional Context Diagrams (as Figure 33)

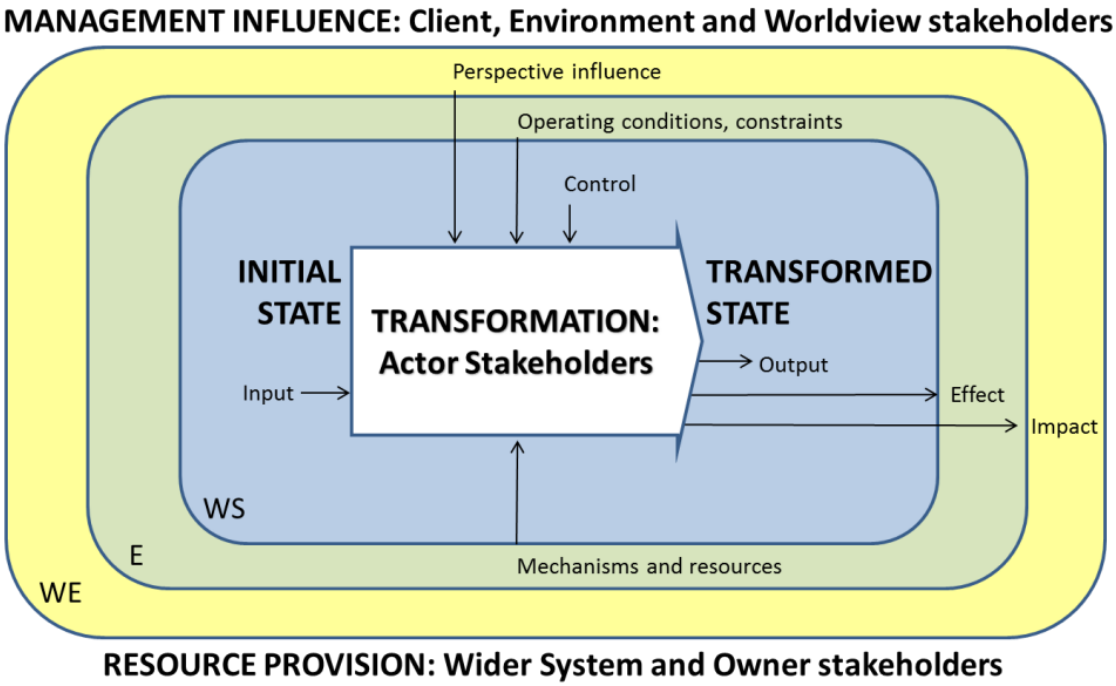


Figure 33: Functional context diagram

7.2 Step 2: Devise functional chain framework[5]

a) Determine functional requirements and flows from the needs of the contextual analysis of step 1.

b) Elaborate candidate mission functional chains according to Transformation viewpoints, starting with client functionality and observing the principle of *Simplicity* where possible

c) Identify *Function interaction types* to determine:

- Unsuitable iteractions
  - Shared services (SS)
- Fundamental blocks
  - Critical chains (C)
  - Control loops (CL)
  - Human issues of complexity (H_K)

[5] Exact process will be determined by Situation *context type*

- Partitioning opportunities
  - Exclusive services (ES)
  - Human conflict issues (H_C)
  - Human agreement (H_A)
- Structural constructs
  - Loose dependence functional chains (F)
  - On-condition loops(L)
  - Judgements (J)

d) Develop the functional architecture of *functional chains* according to Table 10 of section 5.2.2, minimising the partitioning of *fundamental blocks* and trying to achieve a functionally independent design. Clustering methods such as N^2 or DSM may be used to aid in the identification of candidate *functional chains*.

e) Repeat from a) whilst considering Viability, Resource and Management functions (Hitchins, 2008).

7.3 Step 3: Conceive the concept framework

a) Elaborate functions to achieve a level of definition of function that allows subsystems to be proposed, and make a mapping of function architecture to physical architecture observing constraints of the *functional interaction types*

b) Opportunities for similar functionality being performed by a common subsystem should be identified where possible

c) Consider cohesive and dispersive influences on the physical design:
- cohesive (association and conduction) for
  - survival, operation, external and internal compatibility
- dispersive influences for
  - Survival, maintenance, external and internal compatibility, safety and reliability
- any contradictions and resulting compromise trade-offs for
  - Survival, external and internal compatibility

d) Establish form appropriate to both *function interaction types* and other dispersive/cohesive drivers to devise subsystem boundaries

Note: It is possible that the analysis of this step will disqualify a solution – for instance, in considering safety, a given architecture may be determined as unsafe.

7.4 Step 4: Lifecycle solution

a) Mitigate any architectural conflicts across timeline, managing the effect of unavoidably 'compromised' architectural constructs by separation over time

b) Establish a lifecycle solution, whilst not compromising principles already applied in the previous steps, "design for" additional lifecycle related benefits by:

   - Ensuring further functional independence and loose coupling of system components where possible to improve organisational independence, upgradeability, allow variety, improve re-configurability and standardisation
   - Grouping components based on required maintenance action (e.g. Line Replaceable Units) to improve maintainability
   - Grouping reusable components as well as grouping of components by material types to improve recyclability

c) Standardisation enables common solution to achieving functionality, and the reduction of variety achieved reduced complication by similarity

d) Any conflict with previous steps will have to be addressed according to relative merits

7.5 Step 5: Evaluate architecture

a) Calculate the relative merit of the architecture, which is given by the Relative architectural score (Equation 9):

$$RAS = \{Suitability\}_{problem} + \{System\ boundary\ modularity\}_{system} + \{Critical\ functional\ modularity\}_{functional\ chain} + \{Critical\ degree\ modularity + Dispersion\ index + Bridge\ modularity\}_{subsystem}$$

Where:

Suitability ... ... **is given by** ... ... Equation 6

System boundary modularity ... . **is given by** ... ... Equation 8

Critical functional modularity ... ... **is given by** ... ... Equation 7

Critical degree modularity ... ... **is given by** ... ... Equation 2

Dispersion index ... ... **is given by** ... ... Equation 5

Bridge modularity ... ... **is given by** ... ... Equation 4

8 APPLICATION OF METHODOLOGY TO CASE STUDIES

In section 3 it has been proposed to apply the Critical interaction modular design methodology to two specific design problems to demonstrate its utility as a system concept design process (sections 8.1 and 8.2) and to compare its utility with other established methods (section 9). The application of the process to this varied set of cases has also helped to demonstrate areas where the developing process/approach could be improved and this has been fed back into the ongoing process design as part of the research. Two examples are conceived that range in their complication, the first being simple to clearly show the intent and progress of each step of the process, and the second an example to demonstrate the practicality of the methodology for a more complicated and typically encountered problem. The following subsections will therefore address:

- A simple Lego Mindstorms system concept developed for a short course
- A generic cruise missile design

8.1 A simple Lego Mindstorms system

The following is for a system design that is used as part of a lifecycle management system course at the university. The Lego Mindstorms kits are used by students to build simple systems that need to be developed against a high level need, demonstrated and validated. Lego Mindstorms is used as it is simple and intuitive to use, whilst having a degree of complication as it can represent systems with sensors, actuators and software programming to manage the control and information interfaces. Students are asked to design a system by selecting various optional modifications of existing and baseline subsystems. The fact that it is a design used for the course is somewhat incidental, but the design being well understood was a useful starting point for an analysis of the basic system design.

Blocks of Lego bricks consisting of 10 bricks of two sizes and four materials, are delivered by an external agent on a regular schedule for processing. These blocks are collected mechanically and transported to an operator for disassembly - some blocks are contaminated and need to be identified to the operator to be removed from further processing. A further transportation phase is anticipated to a place where the pallets are broken down into their constituent bricks and then mechanically sorted into piles according to material and size, before being removed by another external agent. Collection and sorting will be a continuous process, but during the operation there will need to be regular checks of battery levels and filter condition which will lead to the occasional need for replacement from the store.

The Pick-up vehicle (Figure 34) is designed to locate blocks of bricks, lift them and place them in a drop off area. It uses a sensor to identify contaminated blocks.

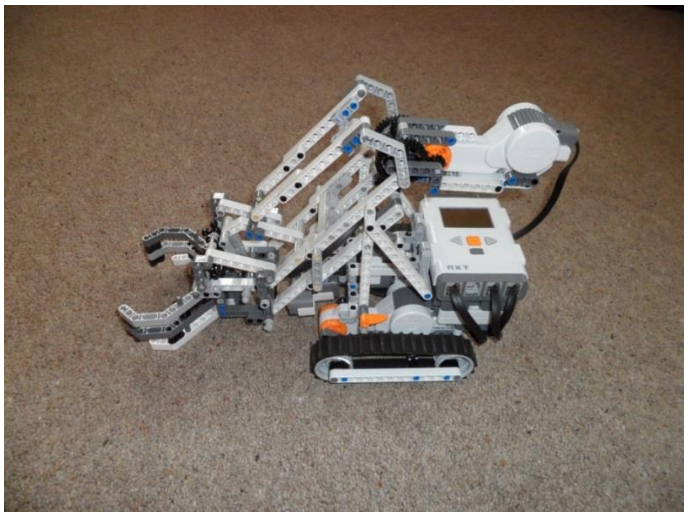


Figure 34: Pick-up vehicle

The pallets can then transported in a Transporter vehicle to a sorting area.

Bricks are then loaded by an operator into a Sorting vehicle (Figure 35) to sort bricks in terms of material and size. Using its sensor and depending on the material of a brick, it will move a specified distance before depositing it in a pile on the ground; depending on the size, it will either move to the left or to the right.

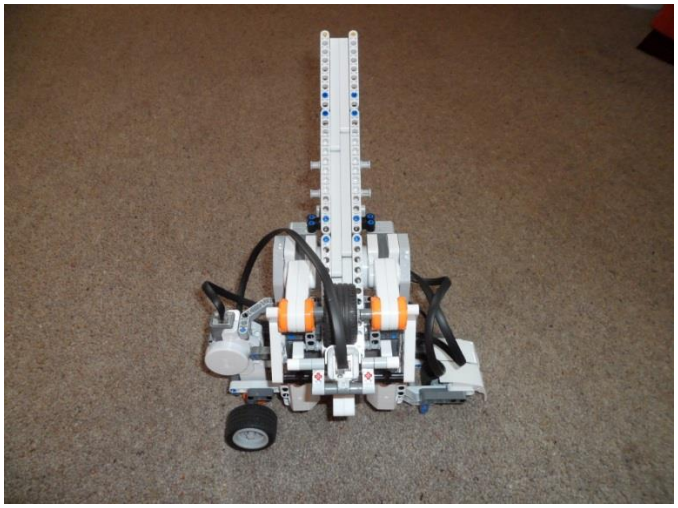


Figure 35: Sorter vehicle

Operator interaction is expected to take the blocks of bricks delivered by the Pick-up vehicle to the Transporter vehicle and then finally to take the transported block, break it into its individual parts for loading onto the Sorter to be mechanically sorted. Three operators are required, one to operate each machine.

8.1.1 Step 1: Analyse the Context type and requirement

a) Establish context type (in order to choose problem solving approach, architectural strategy and risk)

Examination of the Context Types yields Table 16, which shows this as a “Movie” problem as much of the equipment is already available and the task is how it should be used. In doing so, the architecture can be analysed to suggest areas where it is good and areas where it can be improved upon.

b) Understand stakeholders and environment of the system in order identify all influences and capture requirements

This step is to provide the important contextual information that will influence the architectural design:

- By object: pallet, brick (4 different colours/materials and 2 different sizes)
- By subject: Operator actions, Pick-up machine, Sorting machine
- By location/environment: Factory and conditions, Delivery of pallet (input), Removal of bricks (output)

These can be shown diagrammatically on the Functional Context Diagram of Figure 36.

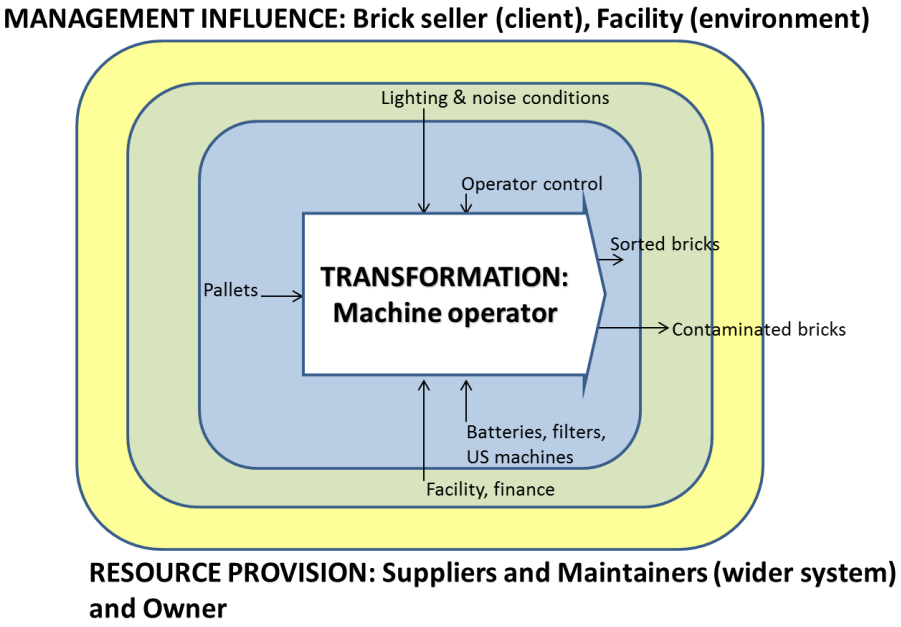


Figure 36: Lego Mindstorms project: Functional context diagram

Table 16: Lego Mindstorms project: Context types

Context Type	Quadrant	Approach	Architectural Strategy	Risk
Process				
Problem	Movie	Component equipment available; modifications might be required according to modified use	Number of scenarios can be explored to determine how system components might be best used	M
Evolution	Obsolescence management	Solution design is expected largely static over lifetime, with changes limited to replacement of parts and possibly extension.	Modular standardized parts to be employed for ease of replacement	L
Response	Routine	Standard project management	No special measures required of architecture	L
Requirement				
Situation	Upgrade	Upgrade required to component designs and their integration	Existing boundary of system and subsystems, with some modification at lower system levels and in integration of subsystems	M
Divergence of values	Unitary	Hard systems analysis	Clear and fixed requirements can be assumed	L
Management	Manageable	Can progress with clear ownership and definition of external boundaries	Can rely on clear definition and responsibilities at system boundary	L
Solution				
Risk	Tried and tested	Use of known solution, technology and process	Can retain existing architecture where possible	L
Complexity	Simple	Requires development and use of simple and decoupled models	Consistent with a clear boundary and modular design.	L
Organization				
Coordination	Centralised	Bespoke development is possible	Can have clarity of external interfaces with clear flow-down of requirements	L
Target	Critical path	Time constrained to replace existing capability		M
Business area	Professional	Skilled workforce, familiar with technologies involved (programming, sensors)		M

8.1.2 Step 2: Devise functional chain framework

a) Determine functional requirements and flows from the needs of the contextual analysis of step 1.

b) Elaborate candidate mission functional chains according to Transformation viewpoints, starting with client functionality and observing the principle of Simplicity where possible

A functional description can be developed from the client needs of this 'movie' problem, with the requirement to make use of existing machines. Starting with the transformation functionality, or what is (are) the primary purpose(s) of the system:

Transformation

- Blocks are delivered (five at a time)
- Pick-up vehicle:
    - used to approach and pick up blocks
    - transport blocks (one at a time)
    - identify contaminated blocks
    - set down block for operator
- Operator:
    - transfers block to transporter
- Transporter takes block to sorting location
- Operator:
    - unloads Transporter vehicle and disassembles block to constituent bricks
    - loads bricks to Sorting vehicle
- Sorting vehicle:
    - sorts bricks into piles according to colour and size
- Sorted bricks are collected

Mission chains can be recorded for the existing system design and these are shown in Figure A - 1.

c) Identify Function interaction types (Section 5.2.1)

d) Develop the functional architecture of functional chains according to Table 10 of section 5.2.2, minimising the partitioning of fundamental blocks and trying to achieve a functionally independent design. Clustering methods such as N^2 or DSM may be used to aid in the identification of candidate functional chains.

Analysis shows the following:

- There are two e*xclusive service* interactions “Deliver blocks” and “Collect bricks” functions (coloured red in diagram)
- There is a pick-up c*ritical chain* to pick up individual blocks (coloured blue in diagram)
- There is a sorting *critical chain* (coloured blue in diagram)
- There are three *judgements* associated with instigating/continuing loops based on power levels and service condition (coloured purple in diagram)
- There are additional o*n-condition loops* for transporting groups of 5 blocks and dissembling blocks and sorting the bricks into piles

Applying guidance of the developed method the following observations can be made:

- *Short critical chains*: the pick-up chain can be divided into 5 separate chains – if there is a failure then this can be recovered quicker and the impact of failure is minimised.
- *Parallel activity*: the critical chains, currently performed in sequence, can be performed in parallel to increase throughput
- *Balance of parallel operations*: the *critical chains* are known to take the following times:
    - Pallet collection chain: 40s
    - Disassembly and load: 20s
    - Brick sort chain: 80s

  The time determining chain is the Brick sort chain at 80s. The other chains will need to be set to respect the same intervals or an accumulation of pallets will occur at other points of the line.
- *Respect external service bandwidth*: the first and last chains form *on-condition loops* with input and output functions. For pallet delivery, this will be one every 80s, though if there were predictable contamination rates then a faster input with some buffering of pallets for collection might be considered. For brick removal, the removal will need to be at an average

rate of 10 bricks per 80s (buffering at this stage can be considered as piles can be allowed to build up).

- There is a possibility of using a common solution for transport, operator and brick recognition functionality.

An alternative functional diagram, using the pickup vehicle for transport, a single operator in one location to allow parallel activity would therefore improve the design, as in Figure A - 2.

e) Repeat from a) whilst considering Viability, Resource and Management functions (Hitchins, 2008).

Considering the resource, management and viability functions:
Resource

- Manage batteries

Management

- Operator control (already considered) including power-on and control

Viability

- Filter changes
- Refurbishment (outside scope of this example)
- Management of incident light conditions and noise levels (external action)

These have been added in Figure A - 3 where the replacement of batteries and filters have been added for both options. But the revised functional solution would be represented as Figure A - 4.

8.1.3 Step 3: Conceive the concept framework

a) Elaborate functions to achieve a level of definition of function that allows subsystems to be proposed, and make a mapping of function architecture to physical architecture observing constraints of the functional interaction types (Section 5.2.2)

b) Opportunities for similar functionality being performed by a common subsystem should be identified where possible

The existing subsystems are mapped onto the earlier functional descriptions as shown in Figure A - 5 and Figure A - 6 for each option.

c) Consider cohesive and dispersive influences on the physical design (Section 5.3)

d) Establish form appropriate to both function interaction types and other dispersive/cohesive drivers to devise subsystem boundaries

Considerations are for:

- Survivability: no hostile environment is envisaged and therefore no architectural strategy required
- Reliability: no strategy envisaged as internal environments not expected to be challenging
- Safety: automated machinery requires safety consideration. Possible safety issues therefore exist at pallet delivery, brick removal and at the operator to machine interface. Separation between operators and machines is necessary whilst the machines are moving and therefore remote operation (wireless option) for commands is essential. However, performance of operator *judgements* will require line of sight
- Maintainability: not considered an architectural issue at this system level (is expected to be an issue at subsystem level)
- Environmental compatibility: Consideration has to be given to spatial access for delivery of pallets and pick-up of bricks from external agencies. This is expected to be a line from input to output. The facility in which the machines are housed needs to allow line of sight for operators (i.e. no dividing walls). Potential contamination from contaminated pallets requires physical separation between good pallets and contaminated pallets/ equipment and decontamination procedures if necessary. If this is not possible to ensure by design, then management of these operations should not be separated i.e. they need to be managed together.
- Operability: ergonomics and HCI issues will be expected at subsystem level.

8.1.4 Step 4: Lifecycle solution

a) Mitigate any architectural conflicts across timeline, managing the effect of unavoidably 'compromised' architectural constructs by separation over time

b) Establish a lifecycle solution, whilst not compromising principles already applied in the previous steps, "design for" additional lifecycle related benefits

c) Standardisation enables common solution to achieving functionality, and the reduction of variety achieved reduced complication by similarity

d) Any conflict with previous steps will have to be addressed according to relative merits

A detailed analysis is difficult for a simple classroom example like this, but the following could be noted:

- Production independence – there is no justification for production independence in this example.
- Line replaceable units – at this level of the system, the individual systems can be considered as LRUs. Without a reliability analysis it will not be

possible to identify items that are more likely to need replacement than others (such an assessment might influence the architectural design). Therefore the only relevant replacement at the operational line level will be the replacement of the batteries and filters. Consideration could be given to whether these could be replaced at the same time, together or with common access.

- Recycling – Lego must be the ultimate recyclable technology! No particular advantage can be gained here.
- Standardisation – there are a number of elements in both candidate designs that promote standardisation. Firstly, the machines are made from the necessarily modular components of the Lego product. Common programmable control units and sensors will facilitate the functioning of the system. Components are standard when they need to be replaced. It should be noted however that Lego routinely subcontracts its components and these can have variations in build standard.
- Reconfiguration – in theory this should have been ensured by functional independence in Step 2. At the system level here, we might consider the machines. Combining multiple operations into a common platform is likely to reduce its desirability for more general use. The combination of pick-up and transport functions into a common vehicle is however not an issue this as the pick-up vehicle already had that capability.

8.1.5 Step 5: Evaluate architecture

a) Calculate the relative merit of the architecture is given by the Relative architectural score (RAS)

An evaluation of the architecture for both options is given in Table 17.

Table 17: Lego Mindstorms example: Architecture assessment

	Option 1	Option 2
Context suitability	Yes	Yes
Critical function modularity	1	1
Critical degree modularity	0.859	0.833
Dispersion index	Compliant[6]	Compliant[6]
Bridge modularity	0.923	0.933
System boundary modularity	1	1
Relative architectural score	3.935	3.905

[6] With wireless operation and physical isolation of the contaminated pallets

8.1.6 Simple Lego Mindstorms example: Summary

In this simple case, there appears to be a marginal advantage for Option 1. This is likely to be because Option 1 is divided into more components, with Option 2 combining tasks of pick-up and transportation for one rather than two operators. However, the scores are very similar, and given this the observations of section 6.2 should be considered and this would likely suggest, having created two designs that are intended to be modular, that architecture is not an important discriminator here compared with the lower wage costs of Option 2.

Using this example, it has been possible to run through all steps of the process. It is a simple example, which facilitates a view of what is actually going on in the process. This simple view however comes with limitations:

- It has only provided a limited exploration of critical interaction types. There are no examples of c*ontrol loop*s that would provide an extra level of complication. In fact it is *control loops* that often create complicated issues across architectural boundaries.
- It only treats the problem at one level of the system hierarchy. The next step would have been to take the process down to the subsystem level and apply it there and, at this level, *control loo*ps would be apparent

8.2 Application of approach to a generic cruise missile example

8.2.1 Step 1: Analyse Context type and requirements

a) Establish context type (in order to choose problem solving approach, architectural strategy and risk)

Examination of the *context type*s yields Table 18.

b) Understand stakeholders and environment of the system in order identify all influences and capture requirements

This step is to provide the important contextual information that will influence the architectural design:

- By object: air launched cruise missile, target, collateral
- By subject: mission planner, pilot, headquarters and politicians
- By location/environment: conditions, scenario, conventions and rules of engagement

These can be shown diagrammatically on the Functional Context Diagram of Figure 37.

Table 18: Missile example: context types

Context Type	Quadrant	Approach	Architectural Strategy	Risk
Process				
Problem	Quest	Explore options	Number of solutions need to be compared to see what is possible	M
Evolution	Obsolescence management	Solution design is expected largely static over lifetime, with changes limited to replacement of parts and possibly extension.	Modular standardized parts for replacement	L
Response	Routine	Standard project management	No special measures required of architecture	L
Requirement				
Situation	Clean sheet	New design concept	Need to define boundary and architecture from scratch	H
Divergence of values	Unitary	Hard systems analysis	Clear requirements can be assumed	L
Management	Manageable	Can progress with clear ownership and definition of external boundaries	Can rely on clear definition and responsibilities at system boundary	L
Solution				
Risk	Play it safe	Design according to safety and service related regulations	Assume regulated boundary and the need to consider critical items in architecture	M
Complexity	Complicated	Large predominantly decoupled models can be developed	Assume clear boundary and modular design.	M
Organization				
Coordination	Centralised	Bespoke development	Can have clarity of external interfaces with clear flow-down of requirements	L
Target	Critical path	Time constrained to replace existing capability		M
Business area	Gold collar	Highly skilled workforce		H

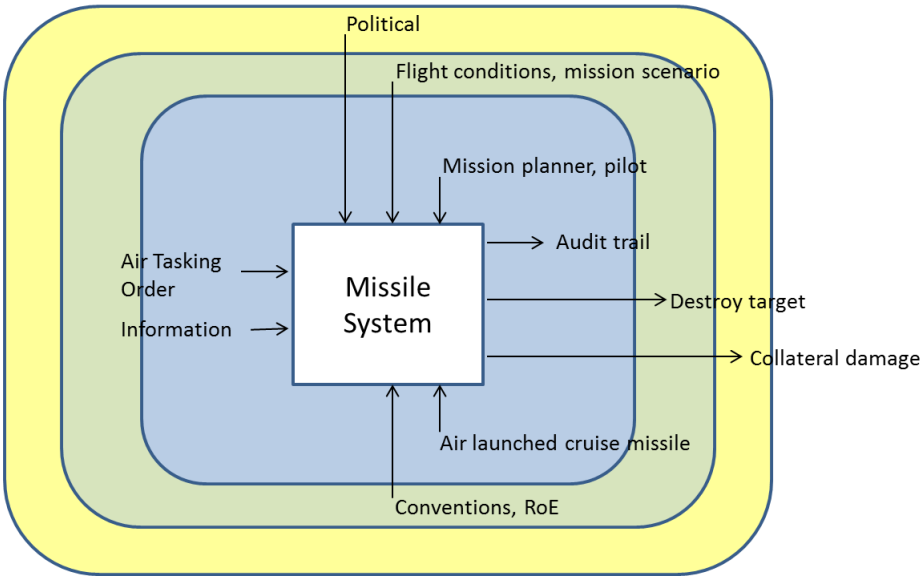


Figure 37: Missile example: Functional context diagram

8.2.2 Step 2: Devise functional chain framework

a) Determine functional requirements and flows from the needs of the contextual analysis of step 1.

b) Elaborate candidate mission functional chains according to Transformation viewpoints, starting with client functionality and observing the principle of Simplicity where possible

A comprehensive, if considerably simplified for the purposes of this example, functional description can be developed from the client needs. The functionality can be described under transformation, resource, management and viability headings as follows:

Transformation

- Missions will need to be carefully planned (Mission planning)
- The missile will need to be safely launched from and aircraft and may not be launched at an exact launch point (Launch)
- The missile will fly a long range (Propulsion)
- It will navigate autonomously to the target (Navigation)
- The missile will follow a predefined route and the associated terrain (midcourse guidance)
- Target recognition is required on final engagement (Terminal guidance)

- The missile will use explosives to destroy the target (Lethality)

Resource

- Fuel management
- Electrical power management
- Air flow management
- Information management

Management

- Automated sequence of operations

Viability

- Test
- Thermal management

c) *Identify Function interaction types*

d) *Develop the functional architecture of functional chains according to Table 10 of section 5.2.2, minimising the partitioning of fundamental blocks and trying to achieve a functionally independent design. Clustering methods such as N^2 or DSM may be used to aid in the identification of candidate functional chains.*

Starting with the Transformation functions, an analysis of coupling can be aided by using the N^2 or DSM tool. If these functions were represented in a Design Structure Matrix the functional interfaces would be as Figure 38.

	Transformation functions								
	Ae	Mp	La	Na	Mi	Te	Fl	Le	Pr
Aerodynamics									1
Mission planning									
Launch		1		1			1		
Navigation		1							
Midcourse guidance		1	1	1			1		
Terminal guidance		1		1	1		1		
Flight control	1	1	1		1	1			
Lethality		1				1	1		
Propulsion	1								

Figure 38: Missile example: N^2 of functional interaction (not clustered)

Analysis either by hand or by the use of proprietary software for DSM, helps to show clusters of interactions (Figure 39).

	Transformational functions										
	Le	Fl	Ae	Pr	La	Mi	Te	Na	Mp		
Lethality	1						1		1	3	To
Flight control		1	1		1	1	1		1	5	
Aerodynamics		1	1	1						3	
Propulsion			1	1				1		3	
Launch		1			1			1	1	4	
Midcourse guidance		1			1	1		1	1	5	
Terminal guidance		1					1	1	1	4	
Navigation								1	1	2	
Mission plan									1	1	
	1	5	3	2	3	2	3	5	7		
	From										

Figure 39: Missile example: N^2 of functional interaction (clustered)

Plotting of Visibility vs Dependency (see section 6.2) we get Figure 40:

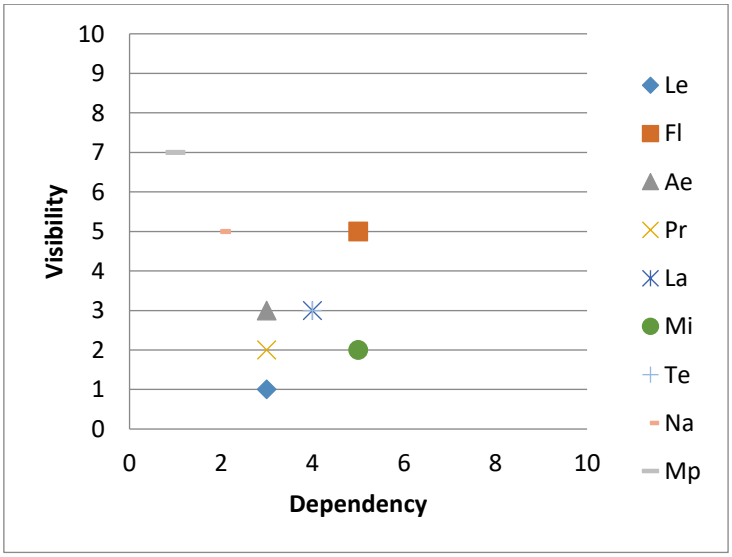


Figure 40: Missile example: visibility vs dependency diagram

The decomposition is far from clean, with Flight Control in particular being both highly visible and dependent. It is therefore important to further examine the nature of the interactions to establish the *functional interaction types*.

Navigation, guidance and control

There is no attempt here to create a detailed technical design of the missile guidance system, but a generic block diagram of missile guidance system in a three degree of freedom representation (attitude, height, distance) is given in Figure 41 (Lin, 1991).

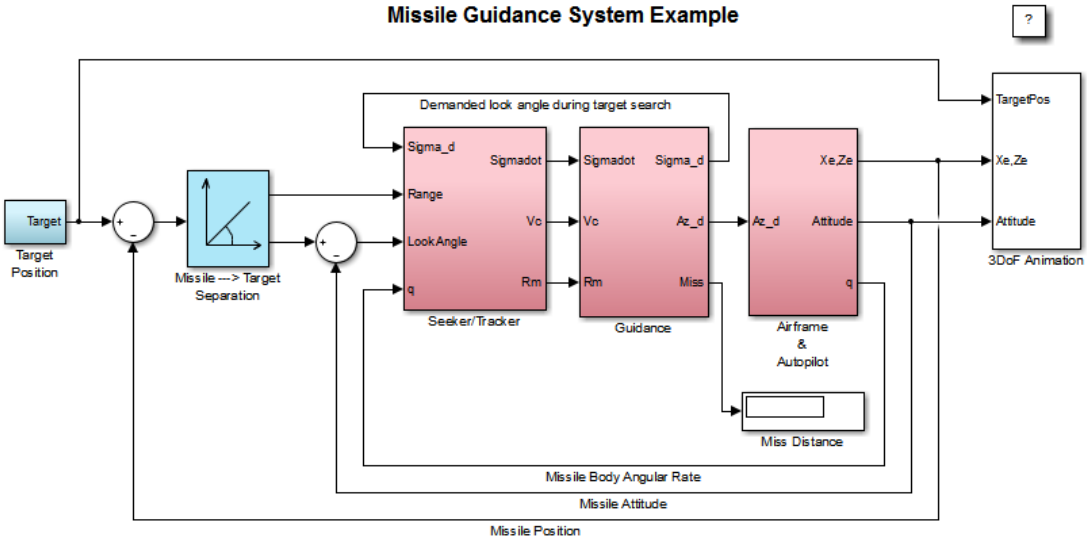


Figure 41: Missile example: generic missile system guidance schematic

Immediately it is obvious that the missile guidance system is a coupled system with a number of control loops. For a cruise missile, a route can be determined in advance and launch and midcourse guidance is achieved by demands from comparing navigation position measurements with those required. In the terminal phase, homing guidance is performed based on the look angle and sightline rate, q. From a system design point of view this is not a very helpful architecture, as it suggests that most of the major functionality of the missile has to be considered and designed together – the control loop and critical chain relationships preventing logical partitions in Figure 42.

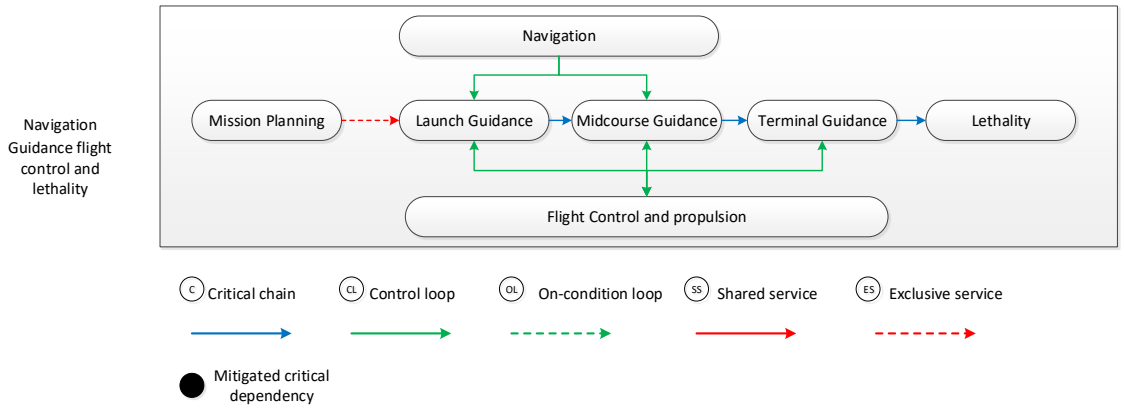


Figure 42: Missile example: initial functional chain framework

The following are potential strategies:

- The main purpose of the propulsion function is to maintain cruise speed to achieve time on target – some variation will be required to maintain speed through manoeuvre, due to drag, but as high ‘g’ manoeuvres are not required decoupling from the guidance functions can be considered.
- There are often non-linear effects that make the design of the autopilot particularly challenging, however, if these non-linear effects can be controlled then decoupling of attitude control from the generation of guidance commands is facilitated. Non-linear effects can be due to varying velocities, changing mass properties and distributions and relative dynamics of missile and its aim-point/waypoint. For a cruise missile with a fixed aim-point, velocity is constant, mass distributions can be to a large extent controlled and waypoints/aim-points are typically stationary. In these circumstances the missile is more like an aircraft or UAV, and control strategies in these cases can allow a decoupling of the autopilot from the guidance system (Sadraey & Colgren, 2005).
- The form of guidance changes throughout the mission due to different manoeuvres, flight conditions and information available. There is an opportunity to divide the functionality at the point that ‘handover’ from one form of guidance to the other occurs, on the assumption that is within a nominal handover ‘basket’:
    - handover to terminal guidance would be when the target is expected to be in the field of view with sufficient manoeuvre capability to engage it
    - for midcourse guidance it will be achieving a waypoint with sufficient accuracy to navigate the terrain below
    - for the launch phase it will be the accuracy of the launch aircraft achieving the launch point in order to engage with the planned route early enough.
- The final part of the sequence can also be considered separately by considering that lethality is dependent on the end conditions of terminal guidance, but for a stationary target these can be defined as a “basket” of parameters that the terminal guidance must achieve to assure acceptable engagement geometry with the target to achieve the required lethality performance.

These strategies enable the following functional chains, as shown diagrammatically in Figure 43:

- Navigation
- Launch guidance
- Midcourse guidance
- Terminal guidance
- Lethality
- Flight control
- Propulsion

Dividing the *functional chains* in this way, has the potential to ease the management of the design. Using the strategies discussed, many of the issues associated with division of *fundamental blocks* have been mitigated and chains have been shortened. “Option 1” reflects the consequential partitioning of the functions:

- Mission planning functions are decoupled as mission planning takes place in advance of the mission.
- Launch guidance, midcourse guidance, terminal guidance and lethality have a chain relationship, but these have been decoupled by defining a “basket” of conditions to satisfy in order to pass to the next phase of the chain.

- Flight control has effectively been decoupled from the guidance loops allowing them to be developed independently. Propulsion and its control will exhibit coupling with the manoeuvres required by the attitude control autopilot, but constraining manoeuvre capability will enable this to be decoupled and concentrate on achieving time on target.

Figure 43 indicates where the violations of *fundamental blocks* have been mitigated. Figure 44 shows the resulting simplified form.

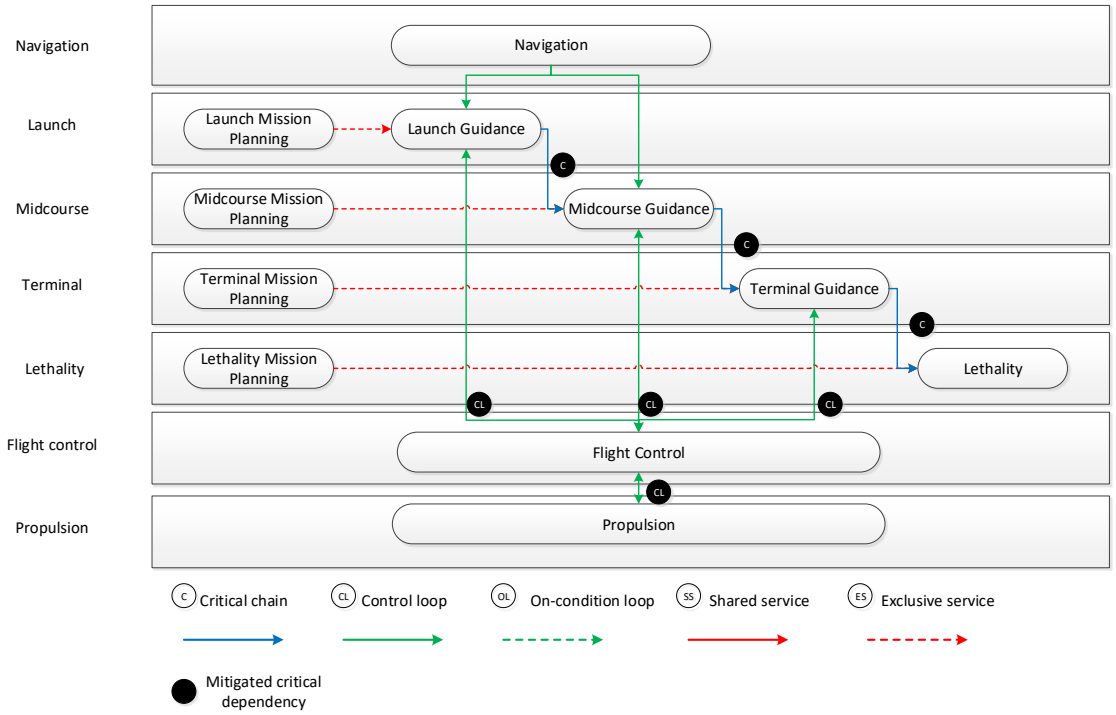


Figure 43: Missile example: functional chain framework (option 1)

Although *Functional blocks* have been compromised, tried and tested strategies have been employed to mitigate the risks. Broad *functional chain* structures are still maintained to ensure management of the decoupled blocks. At the same time the design definition has more structure to understand and develop.

The analysis for option 1 suggests:

- 7 functional chains (these relate to areas of similar function or discipline, where common approaches, design or methods might be employed by a team):
    - Navigation
    - Launch, Midcourse and Terminal guidance
    - Lethality
    - Flight control
    - Propulsion

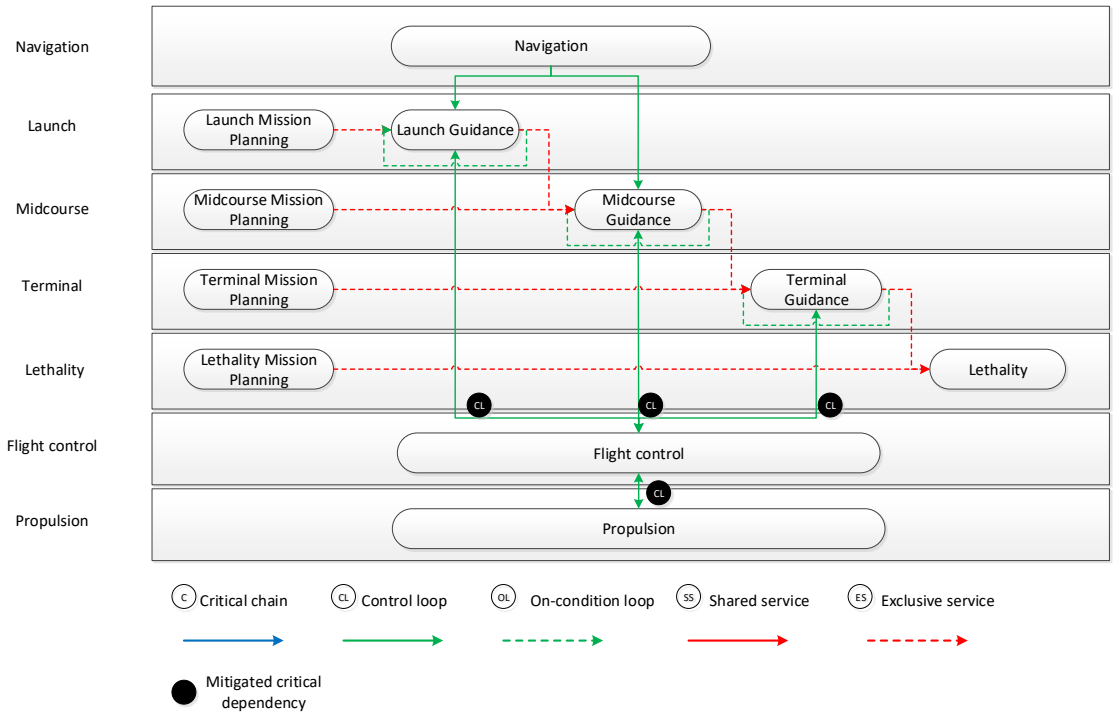


Figure 44: Missile example: functional chain framework (option 1 simplified)

- There were 9 violations of *fundamental blocks* between *functional chains*, but 7 of these have been identified and mitigated, with two violations remaining

Option 2 offers an alternative, which differs from option 1 in that guidance chains are responsible for both attitude control and propulsion control for their phase. This simplifies the system design by removing many critical dependencies between chains rather than mitigating them as discussed for option 1. This would be advantageous if the coupling of propulsion and flight control with the guidance functions is significant. With the reduction in size, power requirements and cost of modern navigation sensors, it may be possible to consider independent navigation functions as well (to simplify further).

The analysis for this option 2 suggests:

- 5 functional chains:
  - Navigation
  - Launch, Midcourse and Terminal guidance
  - Lethality
- There are now only 2 violations of *functional blocks* between *functional chains*

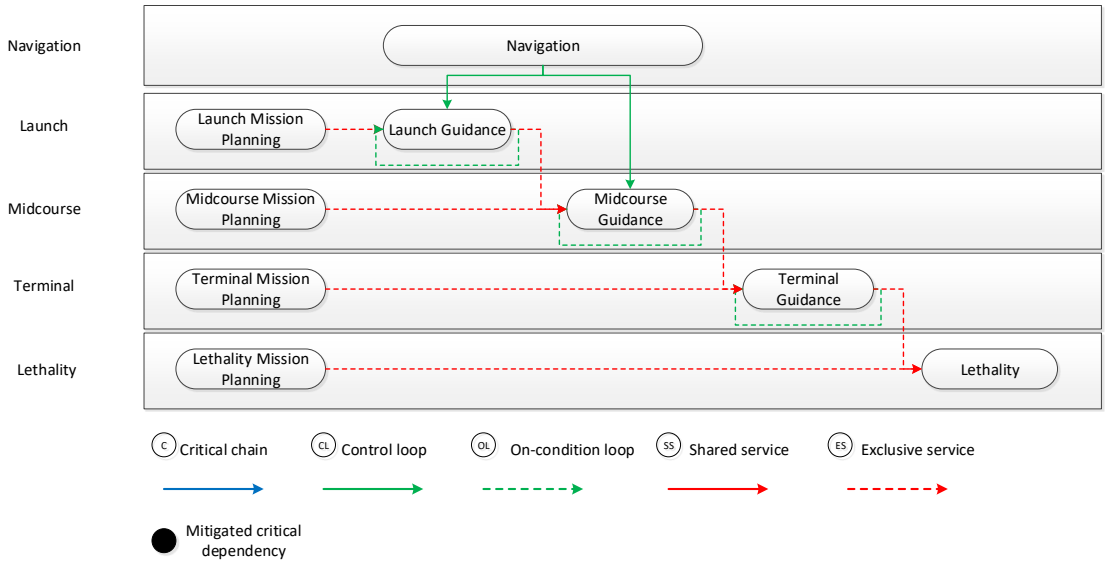


Figure 45: Missile example: functional chain framework (option 2)

There are assumed to be no *judgement* types or *human issues,* as planning is subject to a fixed "tasking order" and flight path is considered here as predetermined. These types, if anywhere, will be determined in the mission preparation phase and *human issues* would be included if the decision process and rules of engagement were considered as part of the "tasking order".

So far we have considered the functions directly responsible for the client's need - the primary "mission functions". These need to be supported by secondary functions for resource management, system management and viability management. These are elaborated in the Table 19.

Table 19: Missile example: viability and resource functions

Category	Subcategory	Function	Subsystem
Viability management	Synergy	Phase, state and mode control	Mission controller, Airframe, communications
	Survival	Not addressed	Not addressed
	Maintenance	Test function	BITE
	Homeostasis	Thermal management EMC management	Structure, insulation Shielding, earthing, filtering
	Evolution	Not addressed	Not addressed
Resource Management	Fuel chain		Thermal Battery, alternator and voltage conversion Fuel tank, pump, injector, engine, exhaust
	Air flow chain		
	Electrical power chain		
	Information management		

There are therefore further resource and viability management functional chains to consider. Resource functional chains are described in Figure 46.

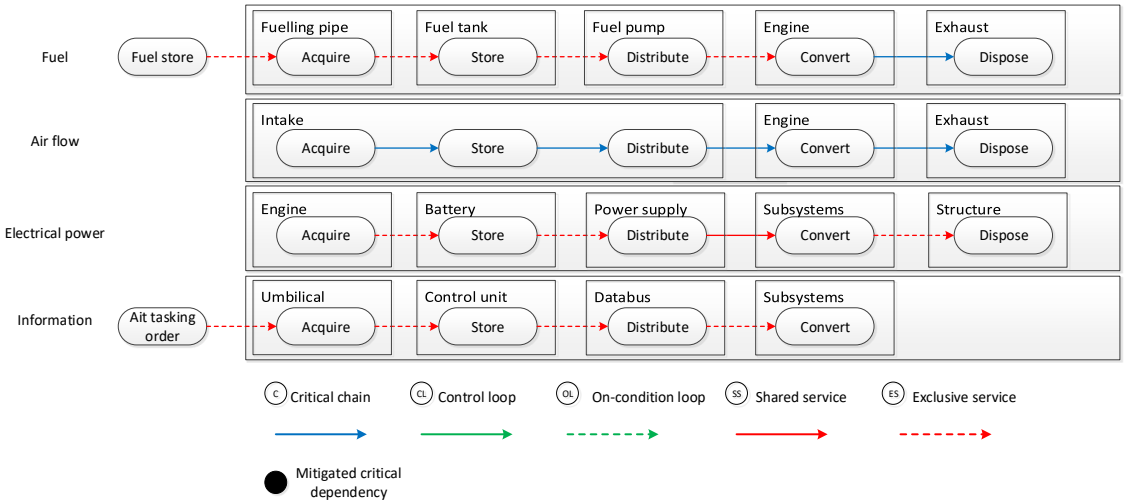


Figure 46: Missile example: viability and resource functional chains

Viability chains can be thought of as follows:

- Management – service
- Test function – service
- Thermal management – chain

In the interest of keeping the levels of information at a manageable level for this analysis, resource management chains will be considered, but viability chains will be excluded. Inclusion of resource management chains requires a modification of the function chain interaction diagram for both suggested framework options, as in and Figure A - 7 and Figure A - 8.

8.2.3 Step 3: Conceive the concept framework

a) Elaborate functions to include functions that allow subsystems to be proposed and make a mapping of function to physical design observing constraints of the Functional interaction types

b) Opportunities for similar functionality being performed by a common subsystem should be identified where possible

To progress, the method requires elaboration of the functions implicit in the functional chain analysis of the previous step. This allows the selection of subsystems expected for the physical solution. Figure A - 9 shows a potential solution for option 1. The equivalent for option 2, is in Figure A - 10. It can clearly be seen that the attempt to simplify the functional framework has created a seemingly more complicated arrangement for the physical solution. However, it is the nature of the interactions rather than the number of them that determines the complication. The fact that the launch, midcourse and terminal phases are sequential and therefore demands are separated in time, should create a more manageable solution.

c) Consider cohesive and dispersive influences on the physical design

d) Establish Form appropriate to both function interaction types and other dispersive/cohesive drivers to devise subsystem boundaries within the broader functional framework.

From a cohesive perspective:

- Survival is important for a cruise missile, but there are important constraints on what is possible. A long range missile that is air launched is usually range limited by the size that can fit under the aircraft. Improvement of survivability through size reduction is therefore not an option.
- The missile does not have a direct operator and so operation can only be addressed for the mission planning and in this case will be addressed by HCI design.
- There is a need to locate elements in the design to be externally compatible with the environment: the seeker needs to be at the front of the missile to see the target; the actuators would be expected to be at the rear of the missile; the exhaust needs to be at the rear and the engine would be expected to be both adjacent and allowing enough room for the intake; the aircraft connection will be at the top near the midpoint of its length; inertial instruments would normally be around the aerodynamic centre

- Cohesion for internal compatibility will largely be centred around thermal management as sources of significant heat need to be close to parts with thermal capacity and thermal management is a particular challenge for long flight times. Significant sources of heat are the engine and thermal batteries and these should be near to the fuel tanks, which provide a thermal sink. Areas where conduction will be important are between systems for earthing, bonding and grounding
- Structure requires cohesion. For a missile, the structure provides rigidity for mission operations, but also aids its ability to manage internal heat. Proximity to structure is important to all physical subsystems and is expected over the length and width of the frame.

The following are also specific and important considerations that would apply to the design from a dispersive perspective:

- For survival, there may be the expectation of separation between redundant mission critical systems. It is probable that a view will be taken that surviving a kinetic hit is not a priority as it is unlikely and no life is endangered if it does happen.
- From a maintenance point of view there needs to be access to subsystems for maintenance. For a missile, space is at a premium and therefore a decision on access and space for maintenance will be a carefully considered analysis.
- Electromagnetic compatibility will require adequate isolation of electromagnetic noise; noise is shielded and filtered both from an import and export perspective.
- From a safety perspective, there will be expected to be isolation between the warhead and sources of heat, such as the engine or thermal battery. For safety there are also implications on design of the firing circuits for the warhead; apart from when the firing chain is activated, the components need to be isolated from each other. As the circuit only needs to be made when it is used, this can be a lifecycle solution (step 4).
- In general, for reliability, it would be considered good design practice to build in reliability by isolation of sensitive components from adverse conditions. Therefore electrical components should be isolated from heat generating equipment such as the engine and thermal battery. In line with an independent design, the first assumption should be that heat is self-contained at a subsystem level.

8.2.4 Step 4: Lifecycle solution

a) Mitigate any architectural conflicts across timeline, managing the effect of unavoidably 'compromised' architectural constructs by separation over time

b) Establish a lifecycle solution, whilst not compromising principles already applied in the previous steps, "design for" additional lifecycle related benefits

c) Standardisation enables common solution to achieving functionality, and the reduction of variety achieved reduced complication by similarity (section 5.1)

d) Any conflict with previous steps will have to be addressed according to relative merits

The first part of this step is to address architectural conflicts across the timeline. Conflicts have been identified in the previous two steps; by conflict it is meant that it has not been possible to observe the ideal design strategies. This will result in fundamental blocks straddling functional chains creating difficulties in analysis and in integration. However a further opportunity is to separate functionality in time over the mission or the lifecycle. Potential conflicts to be addressed for functional option 1 are in Table 20 and those for option 2 are in Table 21.

Table 20: Missile example: addressing conflicts by lifecycle resolution (option 1)

Potential conflicts	Lifecycle resolution
Mission planning to guidance	Simplified model of missile available to mission planning
Lethality firing chain	Components keep separately as Line Replaceable Units until mission. Firing chain not completed until mission criteria have been completed

Cruise missiles are mainly kept in storage in a benign environment and so need little maintenance. Ideally components of the missile will be consistent with the full design life and therefore without need for replacement. However, it is usual for explosive items to have a limited life and need replacing during the life of the missile. Therefore, explosive components should ideally be grouped together for ease of access and replacement.

For a new concept it is often unlikely to have a detailed view on materials, but this would be possible for upgrades. For a missile, many components would be restricted in terms of classification and would require careful disposal – as there are not large numbers this is usually acceptable.

Table 21: Missile example: addressing conflicts by lifecycle resolution (option 2)

Potential conflicts	Lifecycle resolution
Actuators and engine are common to all guidance functional chains.	As the guidance functionality is treated in sequence there is no conflict of their service provided by actuator/engine to the rest of the missile. They are therefore effectively exclusive services and need to design to the most demanding case.
Mission planning to guidance	Simplified model of missile available to mission planning (as Option 1)
Lethality firing chain	Components keep separately as Line Replaceable Units until mission. Firing chain not completed until mission criteria have been completed (as Option 1)

8.2.5 Step 5: Evaluate architecture

a) Calculate the relative merit of the architecture is given by the Relative architectural score (RAS)

Evaluation at this stage does not include viability, management and test functionality as this will require a more detailed definition than is possible here. All frameworks will require this, but perhaps to a greater or lesser extent depending on the number of separate components in the design.

To evaluate the architecture, seven criteria are evaluated in

Table 22.

Table 22: Missile example: architecture evaluation

	Framework 1	Framework 2
Context suitability	Yes	Yes
Critical function modularity	0.944	0.875
Critical degree modularity	0.973	0.969
Dispersion index	Not evaluated[7]	Not evaluated[7]
Bridge modularity	0.998	0.998
System boundary modularity	1	1
Relative architectural score	3.915	3.842

As with the Lego Mindstorms example in the previous section, both of these designs have been created using the method and therefore would expect to have good Relative architectural scores. The most significant difference is in the Critical degree modularity. This is due to the allocation of flight control design to each of the launch, midcourse and terminal guidance functional chains for this architecture. This creates slightly more critical interfaces, but may facilitate the appropriate design of algorithms by the relevant functional chain design authorities.

[7] Not possible to evaluate without a conceptual physical design of each missile, which is outside the scope of this research.

9 COMPARISON OF METHODOLOGIES

9.1 Candidate methods for comparison

In section 3, a number of methods were identified as candidates for comparison with the Critical interaction modular design methodology, the approach being developed by this research. These were selected on the basis of prominence and popularity as:

- Systematic design (SAPD) by Pahl and Beitz (Pahl et al., 2007) as representative of design focused methods (Roozenburg & Cross, 1991)
- Axiomatic design by Suh (Suh, 1997) as representing of attribute focused design
- TRIZ by Altshuller due to its relative popularity in industry
- Product design by Ulrich and Eppinger (K. Ulrich & Eppinger, 2008) due to its popularity in the US
- Total design by Pugh due to its higher than average use in industry.

An initial analysis of each method is carried out against the requirements of Table 5, and this is summarised in Table 23. Comparison of the methodologies against the requirements shows that three candidates are potentially suitable for the comparison (shaded green), and two are not suitable (shaded red). The table shows an obvious divide between stage based models and activity based models. Stage models attempt to describe the entire lifecycle, emphasising the “stage gates” that are required to be achieved to assure good design. At the end of each stage there is a definition of what level of design maturity should have been achieved and what artefacts should have been produced. As a result, they are often at an outline level, leaving the engineer to work out how the design activity should be performed and its effectiveness should be achieved. This level of detail about the ‘how’ is usually contained within the activity based models, which tend to address a particular stage, defining activities to a greater depth and in a way that actively helps engineers achieve an effective design. Stage based models therefore concentrate on assurance while activity based models focus in more detail on the development and improvement of the design and what it can achieve. The distinctly different approaches make it desirable to have a candidate of both types.

Of the activity based methodologies, Suh’s Axiomatic design is the only method that starts with a problem statement (TRIZ aims to improve existing or already conceived solutions). Also TRIZ is a collection of concepts rather than having a prescribed approach; therefore Suh’s Axiomatic Design is chosen as the activity

based methodology. Of the stage based models, Pugh's Total Design does not employ an analytical approach in design preferring to develop many candidates and select a best - this lack of analysis means that Total design is not suitable for comparison. Product design and Systematic design are similar approaches and Systematic design is favoured as it is perhaps most established and represents a 'unified approach' that many stage based models adhere to.

Table 23: Comparison of existing system design methodologies

Required characteristics of methodology		**Systematic design**	**Axiomatic Design**	**TRIZ**	**Product design**	**Total design**
Scope	Concept phase	Yes (entire design process)	Yes (concept phase only)	Yes (concept phase only)	Yes (entire design process)	Yes (entire design process)
Starting point	Both problem and solution based starting points	Yes	Yes	No (only solution based)	Yes	No (only problem based)
Approach	Concrete, prescriptive procedural and analytical	Yes	Yes	No (not prescriptive)	Yes	No (not analytical)
Models	Either activity or stage based	Stage based	Activity based	Activity based	Stage based	Stage based
Aim	Design improvement	Yes (by analysis)	Yes (by analysis)	Yes (by analysis)	Yes (by analysis)	Yes (by selection)
Desirable Support (to concept design):						
Methods (relevant to concept stage)	Yes	Yes	None	Yes (TRIZ toolkit)	Yes	Yes
Means	Yes	No	No	Yes (Innovation machine)	No	No
Notation	Yes	Yes	Yes	No	No	No

9.2 Comparison of methods study for Central heating

In this chapter, the methods chosen in Chapter 3 on Methodology are to be applied to a common problem in order to compare their utility in systems design compared with the approach designed for this research, which is referred to as the Critical interaction modular design methodology. The problem chosen is a domestic central heating system; the advantage of applying the process to a domestic architectural example is that it enables comparison with established and well-tried techniques. The systems of a domestic house are rich enough to examine many different functions of systems; services, control, chains, decisions and judgements. It allows the opportunity to exercise all aspects of systems

design, but remain simple enough so that observations and conclusions can be readily achieved. The central heating system is a typical example and includes:

- Services of energy supply and the supply of a media for achieving heat transfer
- Control of temperature
- Chains of heat exchange and exhaust of waste products
- Human issues between user and owner stakeholders
- Decisions about setting the required temperature versus heating costs and economic running of the system.

The task of designing a central heating is applied to three candidate methods, one in each of the following sections:

- Systematic design (section 9.3)
- Axiomatic design (section 9.4)
- Critical interaction modular design methodology (section 9.5)

In each case, the methodology is analysed in terms of how the central heating problem is addressed in terms of:

- Requirements analysis
- Functional design
- Systems design
- Evaluation of solutions

9.3 Systematic Design solution

9.3.1 Requirement analysis

Clarification of the task

The method describes the following actions to clarify the task:

a) Compile the requirements:
    - What objectives is the intended solution expected to satisfy?
        - What properties must it have?
        - What properties must in not have?
        - Determine quantitative and qualitative data for the checklist.

b) Specify demands and wishes clearly, ranking wishes as being major, medium or minor in importance

c) Arrange requirements in clear order as follows:

- Define the main objective and main characteristics
- Split into identifiable subsystems, functions, assemblies etc or in accordance with the main headings of the checklist

d) Determine that listed requirements are technically and economically achievable.

e) Consider client, state of technology, standards and guidelines, future developments.

Detailed guidance is not provided in terms of the stakeholders or the systems and conditions of the environment. The client is mentioned at e) with an inference that this is the source of requirements along with standards and guidelines. The client is the only explicit stakeholder for the generation of objectives for a), their importance for question b) and their ranking in question c). The method distinguishes two types of client; anonymous customer and specific customers in order to address a given market segment as well as the primary and specific customer. In c), there is also an assumption that the system can split into identifiable subsystems, suggesting that any architectural decisions can be made in a simplistic way. Equally, at d) an evaluation is required to ensure the requirements are achievable. The process described is just an outline and implies that concept design work will be required in order to answer the questions posed, but no specific guidance is given to achieve this.

There is a detailed set of steps for generating the structure of a requirement specification, though there is an admission that at the early concept stage it is not possible to make precise requirements. The overall impression is an approach with a well-defined procedure, but not one that identifies the variety of sources of requirements and constraints.

9.3.2 Functional design

Conceptual Design

Abstraction and problem formulation

A procedural set of questions are asked to enable the designer to abstract the solution neutral problem. These are to ask if the crux of the problem is:

- To improve the technical functions
- To reduce weight or space

- To significantly lower costs
- To significantly shorten delivery times
- To improve production methods

Here we are talking about providing an improvement to heating functions of the house by installing a new central heating system which in this case could be "Ensuring centrally controlled gas heating of spaces of a house to achieve a specified level of temperature in each".

Systematic broadening of problem formulation

Consider extensions of, or changes to, the task in order to test whether it is well defined by abstraction. In this case we may consider whether individual rooms need heating or whether:

- a centralised heat source may be distributed around the house
- an alternative to gas heating represents the best chance of a solution.

In order to allow comparison with other methods, gas central heating system with heating sources in each room will be assumed for this example, unless the method (as will be the case for Axiomatic design) explicitly rejects it.

Establish functional structures

High level functionality is broken down into lower level sub-functions according to whether the design is original, adaptive (by analysis of the existing product) or variant (using established building blocks). As a newly installed system, this will be assumed as an *original* design.

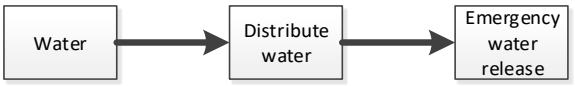
A functional structure is formed by considering "flows" of energy, material and signals, starting with what is considered to be the main flow then developing auxiliary flows. The main flow consists of sub-functions that directly contribute to the high level functions, whereas auxiliary flows contribute indirectly (these could be viability functions). Initial analysis may not provide sufficient detail to allow choice of architecture and so the method advises the following guidance in developing the functional structure relevant to this example:

1. First derive a rough functional structure from functional relationships you can identify in the requirements list.
2. Logical relationships may lead to functional structures
3. Functional structures require flows of energy, material and signals to be addressed. Start with the main flow and iterate to achieve the auxiliary flows.

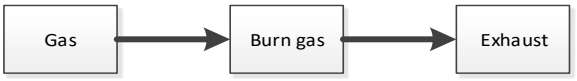
4. Several sub-functions recur in most structures (i.e. change, vary, connect, channel, store)
5. From a rough structure, variants can be derived that allow alternative solutions
6. Functional structures should be kept as simple as possible
7. A selection procedure should be used in the early stages to identify only promising solutions

In the case of central heating the main flows are arguably of heat *energy* and water *material*.

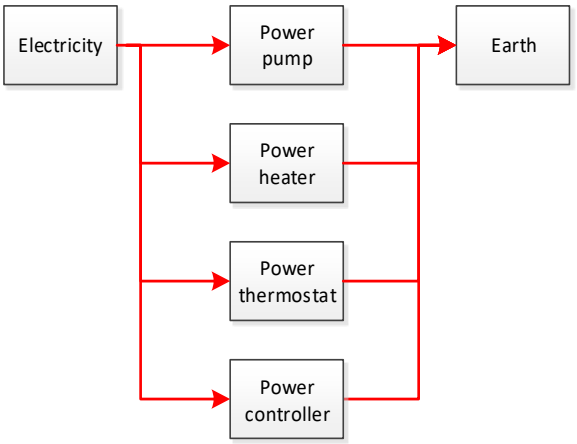
Elaborating the water material flow would give:



Elaborating the gas flow would give:

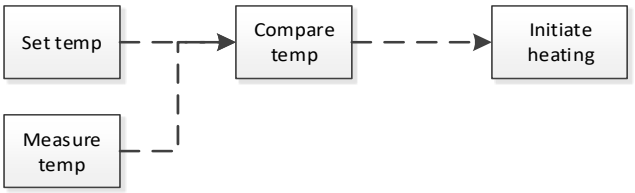


Elaborating the electrical energy flow would give:

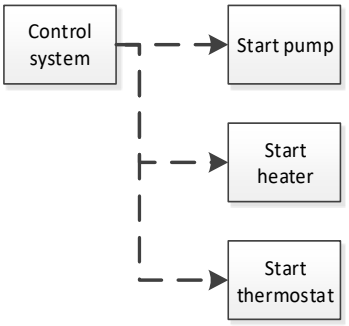


Elaborating the heat energy flow would give:

Elaborating the signal flow would give:



Elaborating control flow would give:



A combined view of this functionality is made in Figure 47.

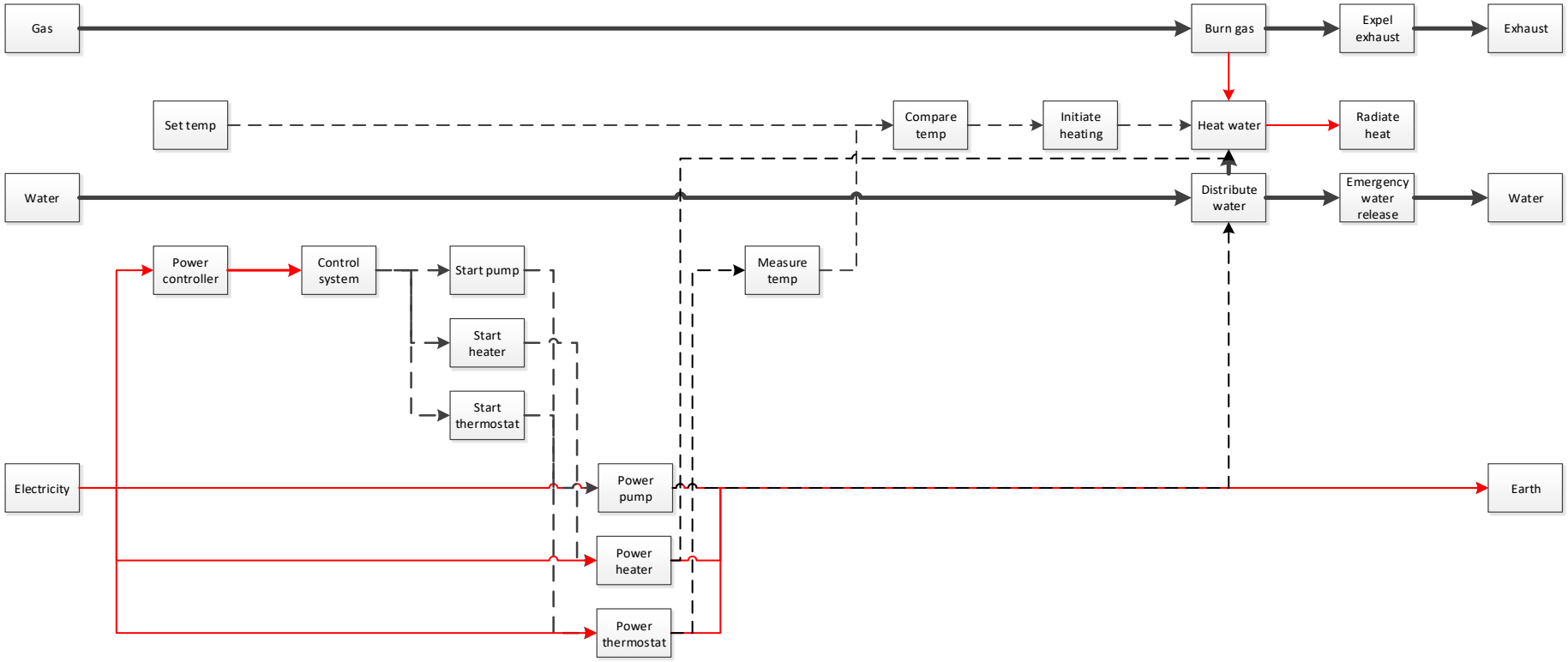


Figure 47: Central heating functional architecture (Systematic design)

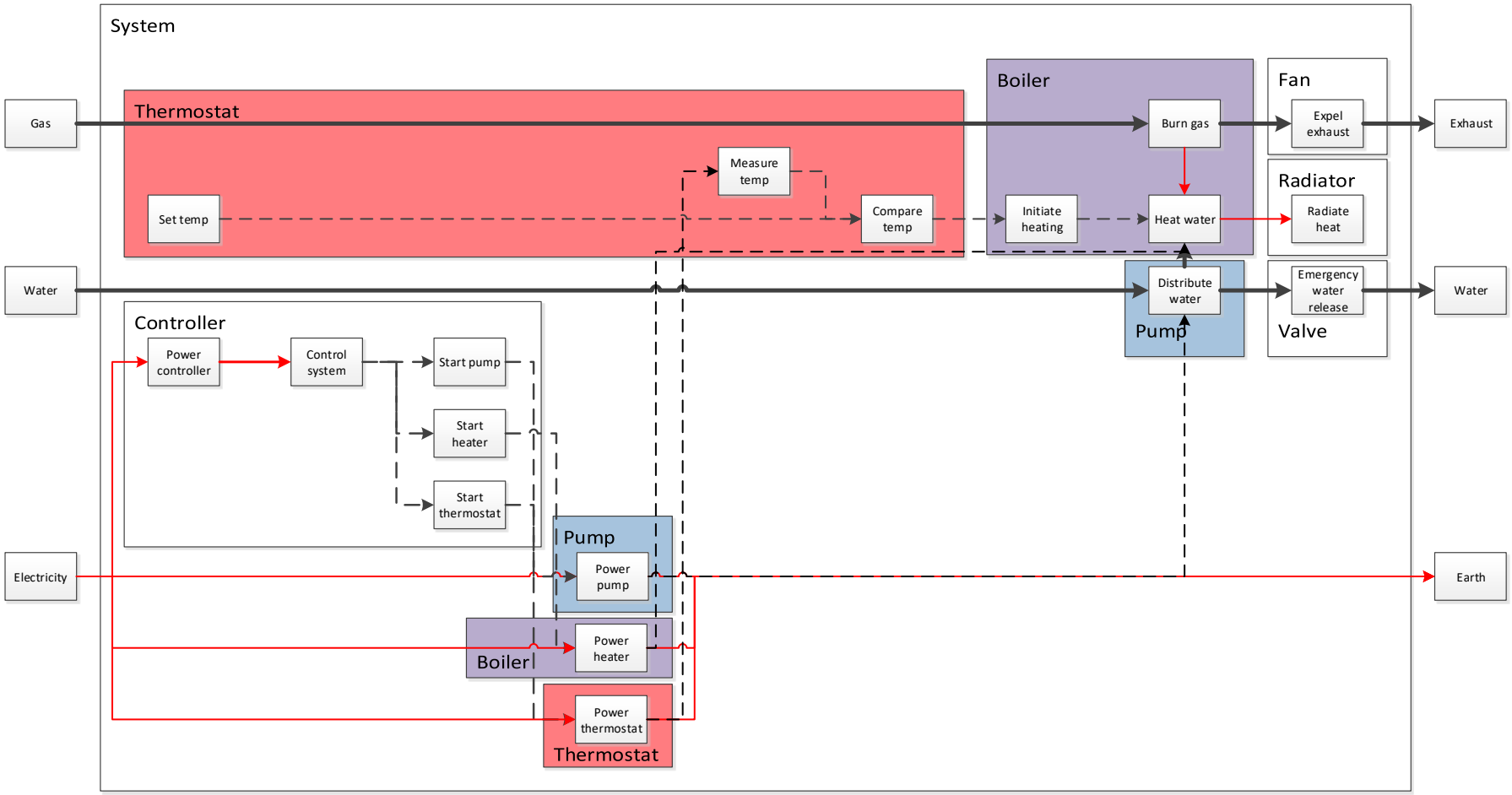


Figure 48: Central heating architecture design (Systematic design)

Examining the guidance above is not clear how the ‘functional structure’ should be created from here. There is a distinction between *original designs* and *variant designs* and as this is a new system in the house then it has been considered an *original design*. However, installation of the system with a well-defined house space makes this a heavily constrained problem, but constrained by the environment and not existing solutions. It is likely that consideration of these constraints is a necessary part of developing the functionality, but this is not prescribed at this stage. There is therefore no progression possible until the *working structures* are considered.

Logical considerations

Constraints are used to establish the logical analysis of functional relationships in the design to construct relationships between subfunctions as AND, OR or NOT relationships. In this example it might be necessary to state that the boiler should not operate without either gas or water supply.

9.3.3 System design

Develop *working structures*

The emphasis of this step is to “determine a physical effect needed for the fulfilment of a given function and also its geometric and material characteristics” (Pahl et al., 2007); this is termed as a *working principle*, a group of which will be used to form a *working structure*. Counsel is given that “it is often difficult to make clear distinction between the physical effect and the form design features”. The end result is intended “to lead to several solution variants, that is, a solution field”.

In the search for a *working principle* for each sub-function, there may be more than one option and the idea is that these can be systematically combined at a synthesis stage. Solutions may be proposed that fulfil more than one sub-function, and in this way a functional architecture will emerge.

Some structure in the process is introduced (a sort of structured brainstorming), with the following suggested as possible methods for proposing candidate *working principles* as:

- conventional aids
    - literature
    - natural systems
    - existing technical systems
    - analogies
    - measurements and model tests

- methods with an intuitive bias
    - brainstorming
    - method 635
    - Delphi method
    - Synetics
    - Combination
- Methods with a discursive bias…..

Systematic combination is suggested as a method of bring together the *working principles* determined through this brainstorming. Use of a *Morphological matrix* can be used for this, but such a matrix will need to be generated in advance from candidate *working principles*. For an organisation that regularly produces a particular type of system, such a matrix might usefully be compiled to capture the organisational knowledge for this step of the process.

Figure 48 is a suggested mapping of sub-functions generated earlier, onto candidate working principles. This seems a reasonable partitioning based upon the author's experience and though there are other options there is no guidance to say which partitioning may be better than another from an architectural view; a decision is therefore delayed until a full evaluation of each concept options is possible, which could be much later in the design process. At this stage there has been no direction to consider the house structure as a driving environmental influence; how many radiators are required, where should the thermostats be placed in order to best control the temperature of the space? It could be stated in requirements, but these are architectural considerations and it should be for the designer to evaluate the options. An experienced heating engineer may well be able to choose a number of different control solutions to the problem, but a methodology should require such a step in the process.

"Design for" principles are not explicitly applied to the concept phase and are left to the "embodiment phase". The implication is that these might have an impact later in the design process with any rework implications that this involves. Therefore, unless there is a specific requirement to be met for the attribute it may not be addressed in the concept decision process and could lead to a non-optimal concept.

9.3.4 Evaluation of solutions

Concept evaluation

A weighting technique is proposed, based on evaluation of goals rather than the design or architecture. The authors admit that at an early concept stage this may

only be an evaluation based upon the likelihood of meeting the requirements, especially essential ones. There is an emphasis on both technical, economic and safety characteristics and a recognition that parameters are likely to be qualitative rather than quantitative. Suggested evaluation criteria are given in Table 24.

Table 24: Checklist with main headings for design evaluation during the conceptual phase (Pahl and Beitz)

Main headings	Examples
Function	Characteristics of essential auxiliary function that follow out of necessity from the chosen solution principle or concept variant
Working principles	Characteristics of the selected principle or principles with respect to simple and clear-cut functioning, adequate effect, few disturbing factors
Embodiment	Small number of components, low complexity, low space requirement, no special problems with layout or form design
Safety	Preferential treatment of direct safety techniques (inherently safe), no additional safety measures needed, industrial and environmental safety guaranteed
Ergonomics	Satisfactory man-machine relationship, no strain or impairment of health, good aesthetics
Production	Few and established production methods, no expensive equipment, small number of simple components
Quality control	Few tests and check needed, simple and reliable procedures
Assembly	Easy, convenient and quick, no special aids needed
Transport	Normal modes of transport, no risks
Operation	Simple operation, long service life, low wear, easy and simple handling
Maintenance	Little and simple upkeep and cleaning, easy inspection, easy repair
Recycling	Easy recovery of parts, safe disposal
Costs	No special running or associated costs, no scheduling risks

At the concept stage there will be limited information available to properly evaluate these parameters and understanding of the system is often not advanced enough to have fully formulated requirements. As a result weighting factors are advised for extremely important requirements only, instead striving for an approximate balance of performance against all parameters. A score of 0-4 is applied, with an indication of associated maturity and terms are the summed to find an aggregate score.

For the central heating example developed here, the evaluation parameters in each case might be as in the following

Table 25. It makes little sense to attempt specific values as this is a subjective evaluation, but comments are made in each case in terms of the feasibility of evaluation at this concept stage, for the concept identified.

Table 25: Evaluation parameters for a central heating concept (Systemic Design)

Main headings	Examples	
Function	Characteristics of essential auxiliary function that follow out of necessity from the chosen solution principle or concept variant	**Experience shows that the solution should be achievable, but nothing in the architectural analysis can suggest what difficulties may be encountered for the *working principles* proposed for the required functions.**
Working principles	Characteristics of the selected principle or principles with respect to simple and clear-cut functioning, adequate effect, few disturbing factors	
Embodiment	Small number of components, low complexity, low space requirement, no special problems with layout or form design	Safety critical elements of gas, hot water, pressurised system, exhaust products will enforce constraints for embodiment in the house.
Safety	Preferential treatment of direct safety techniques (inherently safe), no additional safety measures needed, industrial and environmental safety guaranteed	As this is gas central heating, it should be possible to use components that are certified safe for purpose. System integration will be highly regulated and using established principles.
Ergonomics	Satisfactory man-machine relationship, no strain or impairment of health, good aesthetics	Man-machine interface is in the controller. This can be established at subsystem level.
Production	Few and established production methods, no expensive equipment, small number of simple components	Tools for installing equipment are not specialist, however production of the boiler will involve expensive process.
Quality control	Few tests and check needed, simple and reliable procedures	Checks associated with gas and hot water circulation will be involved. Bespoke system will require bespoke application of quality procedures by skilled installers.
Assembly	Easy, convenient and quick, no special aids needed	Bespoke system, with safety critical components will mean an involved and relatively expensive assembly process.
Transport	Normal modes of transport, no risks	Normal modes of transportation can be assumed for the domestic components for the system.
Operation	Simple operation, long service life, low wear, easy and simple handling	Simple operation can be expected
Maintenance	Little and simple upkeep and cleaning, easy inspection, easy repair	Use of pressurised hot water system is subject to corrosion. Requires periodic service and checks. Repair is often difficult due to concealed pipe work.
Recycling	Easy recovery of parts, safe disposal	No particular issues associated with disposal, metals should be easily recovered.
Costs	No special running or associated costs, no scheduling risks	The boiler is a relatively expensive component. Regular servicing costs are not necessarily typical of electrical alternatives, but running cots are typically lower

Without a more detailed assessment of the concept design, this evaluation will be of a generic nature and will give little indication of the technical difficulties in the functional design. Candidate options could be compared, but performance

and behavioural issues will only become clear when system models are created and used to evaluate performance.

9.4 Axiomatic Design Solution

9.4.1 Requirement Analysis

No process is prescribed for this step and therefore Axiomatic Design must rely on support from other methods. Suh recognises that there may be constraints on the design due to boundary conditions, internal environmental requirements or design decisions (Suh, 2005), but does not indicate how these might be analysed. Suh refers to capturing a societal need with the requirement to capture this and to formalize it; starting from a solution neutral environment. The method is at best descriptive here, relying on users to derive their own process from examples. The author describes the need for creativity by “good” designers. The approach assumes that the designer should work with stakeholders to generate a set of requirements and then generate a set of functions that can be shown, by a mapping, to address those requirements.

9.4.2 Functional design

Assuming that the previous step has outlined the Customer Attributes (CA), these need to be used to generate Functional Requirements (FR). Axiomatic design identifies the activity of mapping, but does not facilitate the generation of functional requirements. This is again left to the designer and it is explained that the process is both a creative and iterative one, with no single correct answer. However, at each level of decomposition of the functional requirement it has to be rationalised against a concept of physical design and a set of associated design parameters to ensure it can meet the Independence Axiom. If a concept that meets the axiom cannot be found then this indicates a badly chosen set of functional requirements: “when the Independence Axiom is violated by design decisions made, we should go back and redesign rather than proceeding with a flawed design” (Nam P. Suh, 2001). The lack of clarity of requirement definition is a potential weakness as if the primacy of requirements isn't established, it complicates the design process i.e. a compliant solution that doesn't meet the axioms may be rejected in favour of a non-compliant one that does.

Without guidance as to a suitable requirement the following is taken:

“Ensuring centrally controlled gas central heating of spaces of a house to achieve a specified level of temperature in each”

This might then be used to generate the following Functional Requirements:

- Heat the various spaces of a house

- Control the temperature of those spaces within a specified temperature range
- Allow user setting of the required temperature range

When proposing a set of Functional Requirements (FR), we are directed to observe Corollary 2, Minimisation of FRs. It might be argued that the need to specify a temperature range and the means of achieving it should be combined, which results in a reduction to two FRs:

- Heat the various spaces of a house
- Control the temperature of those spaces within a *selected* temperature range

The next step is to determine the functional hierarchy, which may require conceptualisation of the physical design (a process referred to as “zig-zagging”). In this instance, we need to conceptualise the house as an existing structure of rooms and spaces. This allows us to decompose the FRs further and assign ranges and tolerances (illustrative values are used in this case) for each:

- Heat the various spaces of a house
    - Heat rooms x,y,z; tolerance/range – up to 25 Celsius
- Control the temperature of those spaces within a *selected* temperature range
    - Set required temperatures in individual rooms – range 10 to 25 Celsius
    - Control temperature of individual rooms - range 10 to 25 Celsius

Tolerances are required as they form part of the compliance with Axiom 1; if the design to meet a given FR allows other FRs to remain within acceptable tolerance then Axiom 1 is satisfied. As confirmation of at least one valid design solution that meets Axiom 1 is required, this step cannot be completed without confirmation of the next step. In accordance with Suh’s corollaries it is also necessary to explicitly require heating and control of each room as they are independent requirements. For this analysis, therefore, two rooms will be assumed.

9.4.3 System Design

A mapping is required between FRs and DPs. The choice of DPs is described as creative and non-unique. It is for the designer to propose a design and then compare it with the axioms, modifying it as necessary to achieve compliance. Suh (Suh, 2001) suggests that the DP for a system can be its components and so a heat source and control source is assumed for each room. Note that although this

may result in redundancy this should become clear from the subsequent analysis and can then be corrected for.

The design matrix for this case would then be as in Figure 49.

	Heat room 1	Heat room 2	Control room 1	Control room 2
Heat room 1	X			
Heat room 2		X		
Control room 1	X		X	
Control room 2		X		X

Figure 49: Design matrix central heating

This is a triangular matrix in Suh's terms, which represents a decoupled system. Corollary 7 clearly favours an uncoupled solution, which can be obtained from combining the two operations associated with each room, giving 'diagonal' matrix of Figure 50.

	Heat and control room 1	Heat and control room 2
Heat and control room 1	X	
Heat and control room 2		X

Figure 50: Design matrix for Axiomatic Design's optimum heating solution

The FRs to reflect this will now be of the form:

"Control the room temperature to within a selected temperature range of +/- 3 Celsius"

This can be addressed by having a heating source in each room, rather than a centralised boiler and heating pump. Such a measure would enable the temperature setting and measurement to be incorporated into the same physical part - as a solution it complies with Suh's Corollary 3 and enables a common design to be used in each room, which supports Corollary 4. However, a solution that has the thermostat as part of the unit would not seem an optimum design as this will allow localised heating within a room. A better solution is likely to having a thermostat that is in a different part of the room. Suh's method does allow this, as for two solutions that satisfy the *Independence Axiom*, the *Information Axiom* can be a final arbitrator. With the thermostat separate from the radiator, there should be an increased probability of meeting the requirement to keep the whole room at a temperature.

The solution is likely to need an electric unit, as having a gas boiler in each room is undesirable from a safety, comfort and cost perspective; the room temperature will be controlled by appropriate setting of each radiator. So rather than a traditional central heating system this approach would suggest decentralising with an independent room based solution – such as Figure 51.

Figure 51: Candidate solution for Axiomatic Design's optimal heating solution

This may seem an extreme interpretation of the method, but each room requires temperature to be controlled to independently set levels and according to different thermal parameters (due to size, windows, outside walls); this will involve independent measurement of each room and independent heating according to the measurement. The relationship between these functions is one of feedback and Suh's method will not allow functional partitioning to proceed in such an instance.

The next step of the method is to assess the 'information' required of each candidate solution. This is defined by the Information Axiom and requires the designer to assess the probability of meeting the FR requirements with the architectural solution. As defined earlier the solution needs to heat the room to a temperature and control it within 3 degrees centigrade. A consideration will be whether the radiator will have the heating capacity for the room; a simple analysis considering the size of the room and its insulation should determine this. If we assumed that achieving the intended solution was going to be difficult due to complex considerations of heat flow then we would conclude that the information level was high – what level would it need to be before a decoupled solution became more favourable? Suh recognises that this could be a possibility, but maintains that in uncoupled solution will always be more favourable.

The above solution is indeed a valid option, but other commonly used centralised heating systems are excluded from consideration by the method, even though they might represent a more economical solution. In reality, centralised systems are possible, but the method doesn't allow these to be explored due to its insistence on an uncoupled design where available. If an uncoupled solution doesn’t exist then the method requires a "near" uncoupled design, where either a clear performance analysis can be made or a clear decoupling strategy can be applied. This is often not the case in modern complex systems as admitted by the author (Suh, 1990).

A final point is that this step is designed to address and provide particular DPs, and these are assumed to be independent. Therefore the method cannot deal with non-functional attributes and the trade-offs that these create.

9.4.4 Evaluation of solutions

The method addresses Manufacturability. The method is extended to ensure that there is a one to one relationship between DVs and Process Variables (PV). This might be possible for components, but at a large scale system level such an interface will be too complex to be dealt with in this way. A particular DV might be associated with a subsystem or assembly, each of which could comprise multiple production techniques.

9.5 Critical interaction modular design methodology

9.5.1 Requirement analysis

Step 1: Analyse the context type and requirement

a) Establish context type (in order to choose problem solving approach, architectural strategy and risk)

Context types are identified in Table 26.

Table 26: Central heating; Context types

Context Type	Quadrant	Approach	Architectural Strategy	Risk
Process				
Problem	PBN	A systematic approach is possible drawing on a wealth of past experience in designing central heating systems	Likely to follow precedent as many alternative architectures will have been devised that can help characterise the problem and solution	L
Evolution	Obsolescence management	Solution design is expected to be largely static over lifetime, with changes limited to replacement of parts and possibly future extensions.	Modular, standardized parts should be considered for ease of replacement	L
Response	Routine	Standard project management	No special measures are required of the process	L
Requirement				
Situation	Clean sheet	Explore requirements and options for new system	A new architecture is required though constrained by the existing house structure	H
Divergence of values	Pluralist	A soft systems analysis can be employed to understand the various stakeholder views	Stakeholder conflicts may need separation of elements in the solution due to differing value for money criteria	M
Management	Manageable	Can progress with clear ownership and definition of external boundaries	Can rely on clear definition and responsibilities at the system boundary	L
Solution				
Risk	Play it safe	Design will be according to established safety and service related regulations	Assume regulated requirements and measures at the system boundary, with the need to consider safety critical items in architecture	M
Complexity	Simple	Simple models can be developed to understand the behaviour of the system	A clear, well defined boundary can be assumed along with every possibility of a modular design	L
Organization				

Context Type	Quadrant	Approach	Architectural Strategy	Risk
Coordination	Centralised or off-the-shelf	There can be a single design authority providing clarity of responsibility and design. COTS options can be considered.	Central authority means that there can have clarity of external interfaces with clear flow-down of requirements	L/M
Target	Stock-in-trade	There will be clear expectations of what needs to be provided	There will be clear architectural drivers	L
Business area	Trade	Specific and well qualified skills will be required in assembly and commissioning	None	M

This step has helped to characterise the problem, the required solution and the organisation required to produce it. The risk associated with developing the design is medium to low; four context types being of a medium risk level and seven at a low level and just one at high. A system designed should be able to proceed with confidence that this is a reasonably well precedented problem with manageable and achievable solutions.

b) Understand stakeholders and environment of the system in order identify all influences and capture requirements

This step is also to provide the important contextual information that will influence the architectural design:

- The boundary of the house could be at its external walls or the boundary to the land that the house is on
- It is already built and composed of individual rooms, that are separated by internal doors, are reasonably well insulated from each other (by regulation), but that have different sizes, thermal properties and opening windows all of which creates different requirements for each space
- The house is assumed as single ownership and therefore there will be a simple client relationship to general external utilities/services (not necessarily true when split into apartments)

In this context, a means of independently controlling the temperature of each room space to a desired temperature is required.

A more detailed analysis of the context would yield:

- By stakeholder (using CATWOE and PESTLE)
    - householder (owner)
    - occupants (clients)
    - service regulation (environment)
    - environmentalists (weltenshauung)

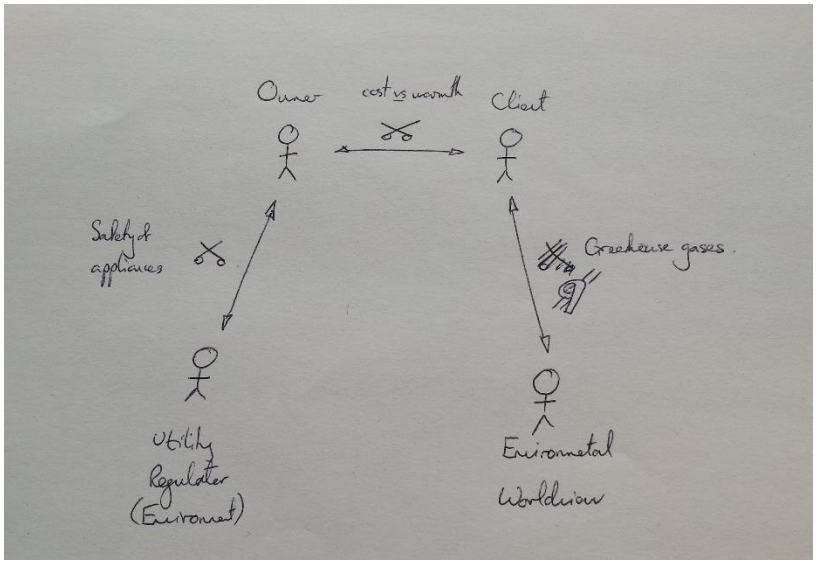


Figure 52: Human issues in central heating

- By object:
    - House; on two floors, its internal spaces need heating to a controlled temperature
    - Assumed use of a hydronic solution (heat transfer using water)
    - External supplies of gas, water and electricity
- By location/environment:
    - Weather conditions
    - External heating sources, heat loss
    - Access to service supplies

These can be shown diagrammatically on the Functional Context Diagram of Figure 53.

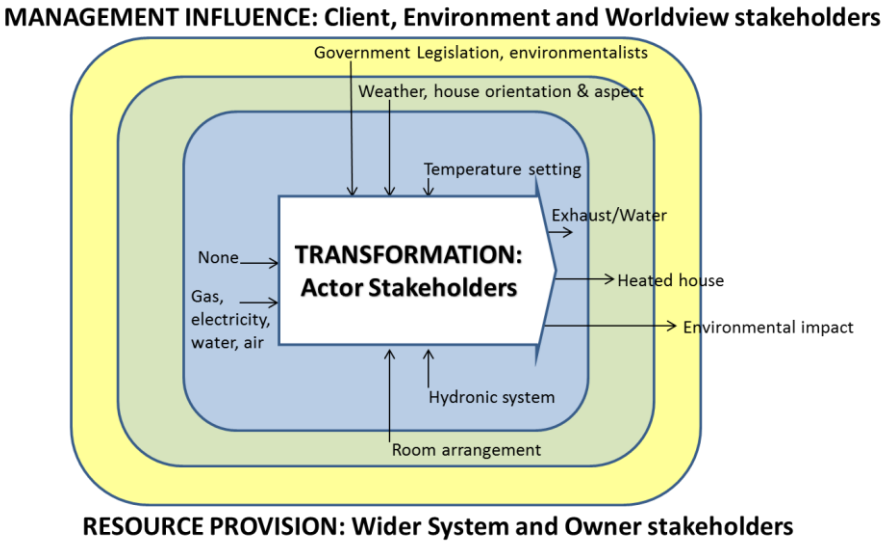


Figure 53: Functional Context Diagram for household central heating

9.5.2 Functional design

Step 2: Devise the functional framework

a) Determine functional requirements and flows from the needs of the contextual analysis of step 1.

b) Elaborate candidate mission functional chains according to Transformation viewpoints, starting with client functionality and observing the principle of Simplicity where possible

For the need of "independently controlling the temperature of the space in a house to a desired temperature". The functionality of the transformation here is simple and so it is possible to examine further resource, management and viability functions at this stage.

- Resource functions can be examined against the Hitchin GRM model, suggesting that resource function should address acquisition, storage, distribution, conversion and disposal of the resource:
    - For gas: provision of gas, metered and distributed by pipe, burnt and then expelled as exhaust
    - For water: provision of water, which is stored in a top up tank, distributed by pipe and disposed of when necessary by emergency water release
    - For electricity: provision of electricity, distributed by cable, converted to energy (of various forms) and disposed of to earth in an emergency case

- Management functions:
    - Control temperature by comparison of a measured temperature to a set level
- Viability
    - Regulate heating by monitoring energy use in order to keep within a monthly budget

These functional chains can be initially drawn out as Figure 54.

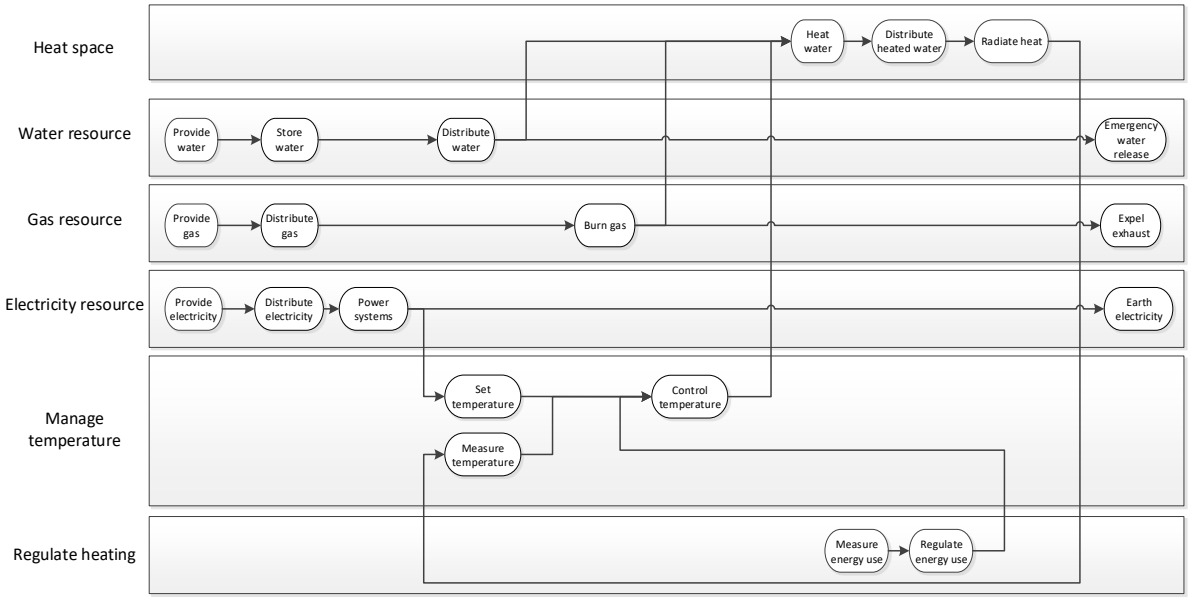


Figure 54: Initial functional chains of central heating example

e) Identify Function interaction types

f) Develop the functional architecture of functional chains according to Table 10 of section 5.2.2, minimising the partitioning of fundamental blocks and trying to achieve a functionally independent design.

However, to improve the functional structure the *functional interaction types* can be analysed as follows:

- Distribution of heated water to radiators is a *critical chain*, but this also involves control of temperature in a *control loop*
- Provision of gas as a shared *service* requires eventual disposal of exhausted gas as a *critical chain*
- Provision of water as a *shared service*, including storage in a top up tank may require emergency water release as a *critical chain* as it is a pressurised system

- Provision of electricity as a *shared service*, is considered as energy that is either converted to useful action or requiring grounding, the latter of which is considered as a *critical chain*
- Set temperature is a manual action requiring j*udgement* based upon measured/experienced temperature
- Regulation of heating is applied in order to control running costs (dominated by use of gas) and is a j*udgement*
- These is a potential conflict between the costs for the owner stakeholder vs warmth experienced by the client stakeholder, which could be seen as *human conflict* (Hc)

With these considerations the above functional representation can be analysed with the additional understanding of the *Function interaction types* as in Figure A - 11. It should be noted that the *functional chains* have been modified; this is in order to reduce the incidence of *fundamental block*s straddling the *functional chain* boundary. 'Heat space' has to recognise that heating radiators requires a hot water distribution *critical chain*, arranged in a *control loop* that is controlling the temperature to a set level. The setting of temperature requires a *judgement* that relies on experiencing the temperature in the space itself and therefore should not be separated from the room. The 'Water resource' functional chain is buffered from the 'Heat space' functional chain by the 'store water' function (which we might anticipate being a header tank). Water, gas and electricity provision are expected to be derived from public utility networks and therefore should be considered as *shared services*, but ones that can be relied upon in terms of capacity (the critical dependencies are therefore mitigated). The 'Regulate heating' *functional chain* function (related to the Owner stakeholder) has a possible human conflict with the Client stakeholder and therefore the functionality is expected to benefit from being separated; hence the Regulate heating chain is separated from the Heat space functional chain. The resulting *functional chain* arrangements only display a conflict in *fundamental blocks* where they exist at boundaries to the property. Those for incoming services are mitigated, as is the earth, for which there is a standardised procedure for domestic household applications. The remaining critical dependencies are for the boiler exhaust and the pressure relief valve – these cannot be mitigated as their features will depend upon the eventual design of the system.

9.5.3 System design

Step 3: Conceive the concept framework

a) Elaborate functions to include functions that allow subsystems to be proposed and make a mapping of function to physical design observing constraints of the Functional interaction types

b) Opportunities for similar functionality being performed by a common subsystem should be identified where possible

As this is an installation of a system for heating in an existing building, an attempt must be made to determine the level of the house architecture at which the function of controlling the temperature needs to be considered; three levels of control can been considered:

- Control of temperature at the whole house level
- Control of temperature at the level of each floor
- Control of temperature at the level of individual rooms

If temperature is controlled at the level of the house, a temperature sensor would need to be consistently at the coldest place of house and therefore other rooms may be too hot. The system would then need to be balanced to try to account for the differences between rooms, but this would only suit typical conditions. A temperature sensor on each individual floor would provide better discrimination than at the house level and settings can be based on the different usage patterns of the two floors and the fact that heat rises. Such a configuration would be more appropriate for open plan floors, rather than with partitioned rooms, where temperature is allowed to equalise over the entire floor. The most appropriate solution to achieving control of temperature across all space in a house is to independently control each room to a level appropriate for each living space as required. Ideally, this assumes doors can be shut for no “leakage” when temperature regulation between rooms is required. This analysis implies that the ‘heat space’ *functional chain* should ideally be applied at individual room level.

The functional chains of water, gas and electricity provision are related to external services and as they provide shared services within the house and they are best considered at house level. The ‘regulate heating’ *functional chain* is servicing a household level issue of cost of ownership and therefore this could be considered to be regulated at the household level too.

Further considerations from a functional perspective are:

- *Independence and failure* – chains cannot be shortened for this example (although exhaust, water pressure release and electrical grounding would benefit from employing short physical distances, as would the loops distributing water to each room that will be inferred in the next step of the process)
- *Balance activities* - temperature control loops will need to account for ambient temperature variations and sizes of spaces in order to provide uniform heating. Where not subject to control action, loops need to be balanced so that they interoperate appropriately; in the case of hydronic central heating, water distribution loops are in parallel and they need to be balanced in order to manage pressure differentials in the system (Caleffi, 2009). Various control philosophies may need to be explored for an optimal configuration (Tahersima, 2012).
- *Parallel activities* – spaces will be heated in parallel

c) Consider cohesive and dispersive influences on the physical design

d) Establish Form appropriate to both function interaction types and other dispersive/cohesive drivers to devise subsystem boundaries within the broader functional framework.

Components for the system to perform this functionality would be: external utilities (gas, electricity and water supply inlets), gas boiler, water pump, radiators, exhaust fan, pressure valve, earthing point, thermostat, and controller.

An initial mapping of functions to these components is most likely to be that of Figure A - 12.

It can be seen from Figure A - 12 that the critical interfaces at the external boundary of the house still exist in the physical domain. The impact of this on the system can however be considered to be mitigated due to the fact that the shared services for water, gas and electricity are subject to regulated interfaces which is designed, tried and tested to provide sufficient capability to deal with normal domestic usage. However the identified solution assumes a boiler component for each heated space – in a house with multiple rooms (assumed to be 10 in this example) this will not only require multiple boilers, but each boiler would require shared services of gas, electricity and water and significantly complicate the arrangement.

A compromise solution is likely to be required and a number of options are available. Assuming a hydronic heating system (i.e. using a water based heat

transfer medium), the following are available as summarised in Table 27 (Taco Learning Center, 1998).

Table 27: Options for hydronic system designs

Potential solution types	Central control	Zone control	Comments
Series (single or multi circuit)	Yes	No	Minimal piping, fittings and installation costs. Furthest radiator takes longer to warm up.
Two pipe (reverse or direct return)	Yes	No	Parallel circuit means more even heating of radiators
Manifold (direct return)	Yes	Yes	Valves used to provide independent temperature control for each chosen location
Primary-secondary	Yes	Yes	Valves and pumps used to provide independent temperature control and flow, providing better control overall.

The first two centrally controlled options simply don't have the variability in their control to be able to cope with changing conditions; as mentioned earlier, radiators could be balanced to suit a nominal set of conditions, but this can only be effective in those nominal conditions. Manifold and Primary-secondary options provide the necessary capability to control the temperature in individual rooms, which is achieved by individual temperature control loops and independently controlled hot water flows/chains. This leaves the decision as to whether the Manifold design (with its control of water flow by simple on/off positions of a valve from a plenum supply at pressure) or the Primary-secondary design with its independent flow-rate control should be preferred. The Primary-secondary design with its greater degree of control of the water loop flow for individual rooms can be used to speed the heating of a particular room, but at the disadvantage of more complicated control (complicating the *shared service*) and potentially at great cost of electricity. The Critical interaction modular design methodology would therefore select the simpler Manifold design on the basis that it provides the same functionality, but in a simpler way. If there were doubts about the performance of the Manifold system (for instance in cases where there was a wide difference in heating requirements between rooms), then both options could be retained for further analysis and comparison.

In the UK the commonly employed method of providing zonal control is by the use of thermostatic valves on the radiators of a Two-pipe design. This is arguably a simpler method again than the Manifold design, but again complicates the nature of the *shared service*; from the perspective of the thermostatic valves, they

will be at different distances from the boiler and therefore will have to operate with water of different temperatures and from the perspective of the boiler it would have an ideal requirement to supply each valve according to different requirements.

[Note: whilst independent flow-rate control is often used in the US, they are not commonly employed in the UK. When applying the method the author, with experience only of UK heating systems, was inclined to think that such an option was ideal, but not a practical proposition. However, the frequent use of similar concepts in the US suggests that the method is capable of challenging existing norms and allowing providing better options.]

A schematic of an example Manifold design is given in Figure A - 13. This alternative architecture (without towel warmer to avoid complicating the diagram) is shown in Figure A - 14 a*nd has* achieved the use of a single boiler, which manage costs and will also provide a safer solution compared with the architecture of Figure A - 12 (as will be apparent in the next step). Instead of independently controlled heat sources, independent water loops for each room are provided by pressurised supply and return manifolds, which then assumes two shared service functional interactions (shown). *In terms of fundamental blocks that have been partitioned across elements of the system, the architecture of Figure A - 14 shows the three mitigated shared services for the external utilities of gas, water and electricity. There are also the critical chains for removing exhaust fumes, relieving water pressure in a fault situation and grounding of electricity; whilst these critical chains will be needed in every design, the independent boiler in each room would require an exhaust fan for each room; further difficulties of this will be returned to when considering safety and external compatibility later in the process. A single boiler solution has however achieved a minimal set of critical chains for exhaust and grounding.*

b) & c) Consider cohesive and dispersive influences to establish Form appropriate to Function Types and other dispersive/cohesive drivers

A cohesive and dispersive influence assessment would suggest the following drivers:

- External compatibility: analysis would suggest radiators should be located by windows and safety outlets should be on an external wall. This helps with heat circulation, but also the area beneath window is often ‘dead’ space from a furniture perspective
- Safety: the boiler should be both close to the gas supply inlet and by an external wall for the exhaust outlet and safety reasons. Earth cabling should be in accordance with regulation. No such spatial limitations need

apply for water and electricity due to their ease of distribution and safe containment. Boilers should be kept out of living areas.

- Internal compatibility: there should be a separation between the radiator and thermostat within a room. This prevents localised heating within a room and is a reason for not having thermostatic valves on radiators.

An example schematic for a house is given in Figure 55. This shows a view as to the optimum placing of the boiler, radiators and thermostats for a building. This can then be used to aid the *dispersion index* calculation in the evaluation step.

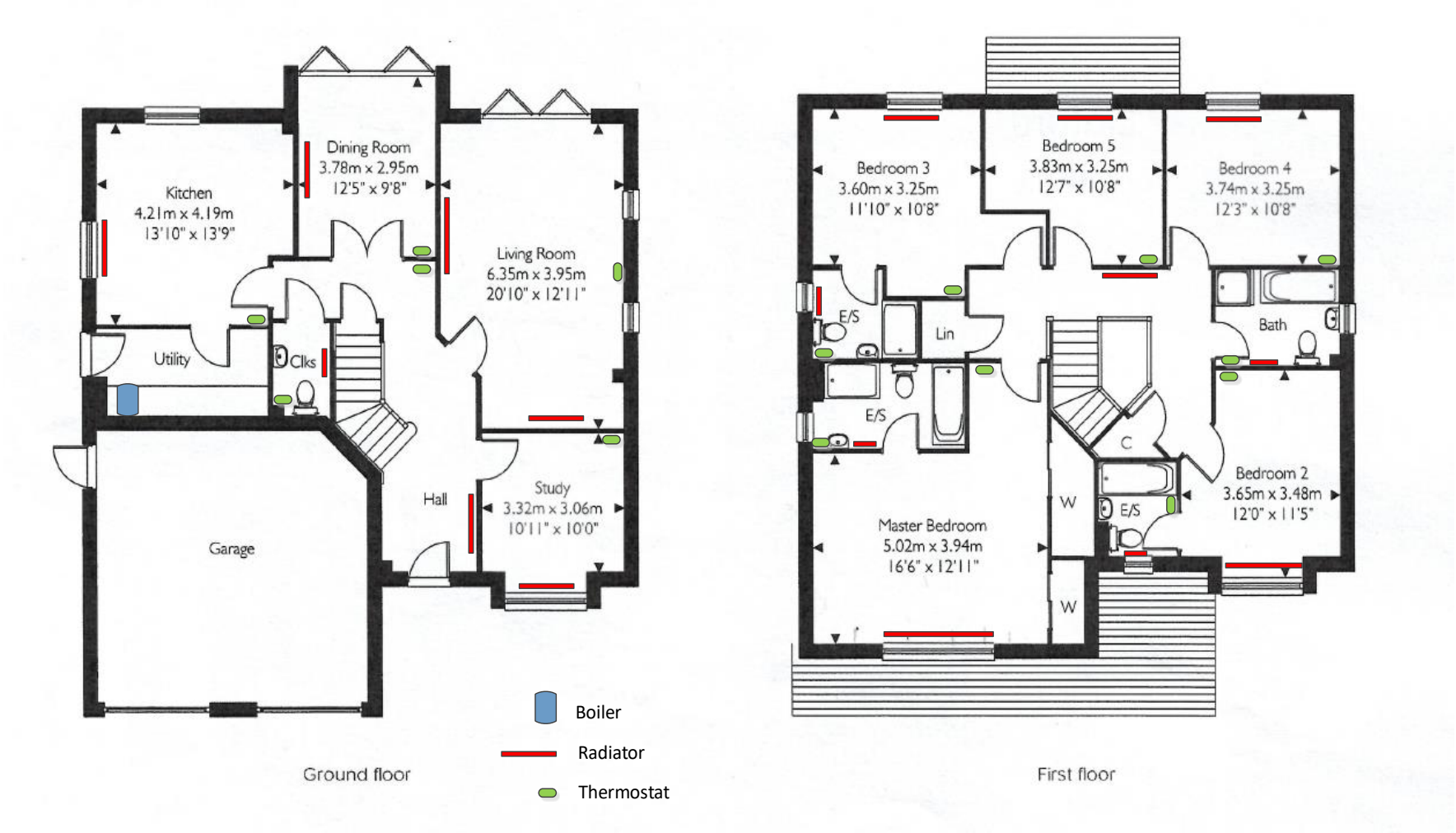


Figure 55: Schematic of example placing of components in central heating system

Step 4: Lifecycle Solution

a) *Mitigate any architectural conflicts across timeline, managing the effect of unavoidably 'compromised' architectural constructs by separation over time*

b) *Establish a lifecycle solution, whilst not compromising principles already applied in the previous steps, "design for" additional lifecycle related benefits*

c) *Standardisation enables common solution to achieving functionality, and the reduction of variety achieved reduced complication by similarity (section 5.1)*

d) *Any conflict with previous steps will have to be addressed according to relative merits*

From the previous steps of the approach, there are not anticipated to be poor architectural constructs that need to be separated over time. However, design for lifecycle could include the following measures:

- Production Independence – items are Commercial off-the-shelf items (COTS), with the exception of a bespoke arrangement of interconnections (pipework and wiring) which will be bespoke to the build, and so the solution is inherently modular with has a high degree of production independence
- Line Replaceable Units (LRU) – in a modular design consisting COTS items, these items can be LRUs. The individual elements of the system that may need replacing during the system life are likely to be:
    - the boiler and this could be line replaceable with the exhaust expulsion fan
    - pump
- Organisation independence – utility ownership is established at the boundary of the property - they will require assurances of safe and suitable design. All other parts of the system are assumed to be under the single ownership of the householder. An exception would be if the property is leasehold with multiple ownership boundaries (such as apartments), which would lead to a requirement for an architecture that ensures independent supply/ metering
- Standardisation – standard COTS items can be employed as the system design is a common one, with elements that are used in other system designs. Multiple spaces in a house enable similar modular components to be used.
- Reconfiguration – this heating system is a centralised system and so reconfiguration is not anticipated as a driver. If it was, then there would be more of an argument for functional independence of the subsystems. With the current design, the decoupling of heating supply and control for each room does increase possibilities for reconfiguration

- Recycling (including reuse and disposal) – not applicable as choice of materials has not been made at this level of design

In summary, the use of standard COTS items ensures a modular design allowing suitable strategies for LRUs to be chosen. Leasehold apartments could lead to a the need for further architectural boundaries to be imposed within the boundaries of the property for the purposes of metering.

9.5.4 Evaluate solutions

a) Calculate the relative merit of the architecture is given by the Relative architectural score (RAS)

The Modular Approach Methodology favours “primary-secondary” to provide effective, independent control of rooms of house. Simpler “two pipe” systems cannot provide the variability to cope with varied environmental conditions. An option of having a heat source (gas boiler) in each room is discounted for costs and safety reasons.

Table 28: Architectural evaluation of central heating options (Critical interaction modular design methodology)

	Option 1	Option 2
Context suitability	Yes	Yes
Critical function modularity	0.81	0.89
Critical degree modularity	0.980	0.976
Dispersion index	Not a suitable design	0.757
Bridge modularity	0.999	0.999
System boundary modularity	0.972	0.972
Relative architectural score	-	3.761

Option 1 is not suitable as it suggested boilers in living spaces. Even if this hadn’t been an issue the dispersion index would have been low due to the desire for the boilers to be close to external walls for exhausted gases. Option 2 has a reasonable *Relative architectural score*, compared with the previous examples of chapter 8, but receives a lower score largely due to the need to centralise the boiler source for reasons of costs and safety.

With only limited experience of the Critical interaction modular methodology to date, it has not been able to characterise what represents a ‘good’ score for each of the parameters or indeed therefore the *Relative architectural score*. The development of this understanding is a suggestion for further research identified in section 10.6.

9.6 Discussion

9.6.1 A high level comparison

Systematic Design and Axiomatic design were chosen as examples as they met criteria set for a methodology that would address the aims of the methodology being developed in this research. Table 29 shows that the Critical interaction modular design methodology developed also meets these criteria. The next sections analyse the performance of each of the methodologies to establish their strengths and weaknesses.

Table 29: Comparison of methodologies

Required characteristics of methodology		**Systematic design**	**Axiomatic Design**	**Critical interaction modular design methodology**
Scope	Concept phase	Yes (entire design process)	Yes (concept phase only)	Yes (concept phase only)
Starting point	Both problem and solution based starting points	Yes	Yes	Yes
Approach	Concrete, prescriptive procedural and analytical	Yes	Yes	Yes
Models	Either activity or stage based	Stage based	Activity based	Activity based
Aim	Design improvement	Yes (by analysis)	Yes (by analysis)	Yes (by analysis)
Desirable Support (to concept design):				
Methods (relevant to concept stage)	Yes	Yes	None	Yes
Means	Desirable	No	No	No
Notation	Yes	Yes	Yes	Yes

A summary of the steps of each method against a generic description of the systems design process is given in Table 30.

Table 30: Summary of processes to be compared

	Critical interaction modular design methodology	Axiomatic Design	Systematic design
Requirement analysis	Step 1 "Analyse the context type and requirement" 11 Context types are used to help inform an architectural strategy. The method prescribes process for a variety of start points (both new and existing) Structured approach a) Record stakeholder needs and nature of interaction b) Consider objects being acted on or systems interacted with c) Consider impact of location/ environment d) Record constraints imposed by system level decisions (mechanism) e) Capture using Functional Context Diagrams	Advocates an optimum design so assumes a new design. Suggests that House of Quality can be used for existing design, but no attempt to prescribe its integration into the method. No attempt to prescribe problem definition	**"Clarifying the task and elaborating the solution"** The method recognises three different starting points and these are discriminated within a prescriptive process. No structured/formal process for identification of requirements, but uses an outline set of questions. Defines means to abstract to a solution neutral problem
Functional design	**Step 2: "Devise mission functional framework"** a) Elaborate mission functional chains according to Transformation, Resource provision and Management Influence viewpoints b) Determine Function types c) Address completeness by elaborating Viability and Resourcing	**"Determination of FRs in Original Design"** No prescription – left to "experience" of designer	**"Establishing function structures"** Start with main flow, break down into sub-functions and examine logical interactions. Use material, energy and signal flow classifications
Systems design	**Step 3: "Conceive concept framework"** a) Consider dispersive (spatial/insulation/isolation including filtering) drivers between system elements b) Consider cohesive (conduction) drivers for elements as above c) Consider association drivers for elements d) Establish Form appropriate to Function Types and other dispersive/ cohesive drivers Step 4: "Reconcile against lifecycle solution" Address architectural conflicts across timeline Design for, according to: Organisation independence, Production independence, Standardisation, Line/ Lifecycle Replaceable Units, Reconfiguration, Recycling	**"Decomposition of the design" and "design helix"** Map FRs to DPs, applying design axioms, corollaries and theories Not applicable	**Embodiment Design** a) Combining solution principles to fulfil the overall function b) Selecting suitable combinations c) Firming up into concept variants d) Three basic rules (clarity, simplicity, safety) e) Embodiment design checklist
Evaluate solution	**Step 5: Evaluate architecture** Evaluation based on modularity indices Step 6: Evaluate system	Satisfaction of Axioms	Evaluating the concept variants

9.6.2 Systematic Design

Systematic Design clearly prescribes steps to perform during concept development. The step of *clarification of the task* is high level and does not develop key elements that may form a key part of the requirement. The only stakeholder referred to is the client and there is not explicit recognition of the constraints that will be applied by the environment. In this step there is an implicit assumption that the split of the system into subsystems, functions and assemblies is a straightforward task. The step of *conceptual design* encourages abstraction to a solution neutral problem, though this does not account for the constraints that would apply if this was not an *original*, but an *adaptive* or *variant* design. The seven steps to *elaborate functional structures* are clear and the concept of *flows* is a very effective way of elaborating functions and their interactions, although interactions are only distinguished in terms of *material*, *energy* and *signal* flows rather than behaviour. The flows made it easy to identify key areas of functionality required of the heating system. However, the exercise exposed that there is no guidance as to what might be an effective functional architecture. The search for candidate physical subsystems is conducted by various forms of structured brainstorming, with an assumption that an organisation will have already developed guidance, in the form of a *morphological matrix,* on how these components can be combined. The system designer then has a requirement to propose system architectures from potentially viable combinations, where guidance based on previous experience is called for. Filtering of potentially useful concepts is left to the evaluation stage and is at this stage that the failure to identify environmental constraints as part of the task clarification step will potentially result in a poor design. *Concept evaluation* is against a set of attributes, many of which cannot be determined without a detailed assessment of the concept design which may not be possible in early design stages. Criteria suggested tend to be based on standard “design for” practice and therefore heuristic in nature. Without functional models, or past experience, it will not be possible to evaluate the functional design.

Strengths

- Prescribed set of steps with supporting diagrammatic notations
- Stresses a solution neutral approach
- Flows are an effective concept for developing functional description

Weaknesses

- High level view of requirements with limited contextual identification
- Not clear how to deal with existing solutions

- No guidance on partitioning of functionality
- Assumes existing organisational knowledge for guidance on partitioning the system
- Methods tend to be creative rather than analytical
- Evaluation is of effectiveness of design rather than architecture, which is only likely to be possible when design has advanced and models are available

The overall conclusion about this method is that there are useful techniques in helping to build a system definition, but lack of emphasis on requirements and analysis of functional design means that a system can only be evaluated when reasonably detailed concept models have been created and concept options are shared with stakeholders to validate or elaborate the initial requirement.

9.6.3 Axiomatic Design

Axiomatic Design provides no guidance on how to identify requirements. There is a suggestion that the method of Quality Functional Deployment can be used to facilitate the discussion of the requirements and priorities with a potential customer, but there is no attempt to identify stakeholders or important influences of the system context. There is also no guidance on the development of the functional architecture as Suh's axioms primarily relate to the way that function is partitioned to the system design and functional design is left to the 'experience' of the designer. Therefore it is concluded that for systems of moderate complication, the tools provided are not flexible enough to apply without the considerable support from traditional system design techniques. The Independence axiom puts a complete emphasis on functional independence in a way that favours uncoupling of the system and there is little flexibility in decision making - an independent design is paramount regardless of other factors or measures. The methods for establishing architectures are limited to consideration of functional partitioning and manufacturing only, with no obvious recognition of the need to consider non-functional parameters associated with the quality attributes of the system. Instead, the Information axiom is used as a method to assess the relative complexity of interfaces, but the method of calculation is not clear for complex systems and even simple mechanical interfaces involve a considerable level of calculation. The following are strengths and weakness identified and starred where observations coincide with Suh's own analysis.

Strengths:

- Holds firmly to the established technique of ensuring functional independence

- There is a recognition of the need identify the complexity in an interface

Weaknesses:

- No guidance on generation of requirements or the functional design
- Inflexible approach to architecture means that it may not suit large and flexible systems*
- It does not accept inevitable compromises of reuse in legacy systems, preferring a rigid adherence to axioms*
- It cannot deal with situations where the independence axiom cannot be met (Suh terms these “unstable systems”)*
- There is no recognition of human interaction, which can introduce unpredictable effects outside the design analysis*

9.6.4 Critical interaction modular design methodology

Unlike the previous methods, the Critical interaction modular design methodology takes a detailed view of the context in order to identify the requirements, and in doing so identifies potential architectural issues at the boundary of its system of interest. This is important as the system boundary should be seen as an integral part of the architecture as it forms a ‘subsystem’ boundary for its higher level system or system of systems. A key part of requirement identification is to be able to analyse characteristics of the context, *context types*, to determine relevant architectural strategies to be employed.

There are a number of concepts for developing the functional design: identifying stakeholders, objects and mechanisms of the system; incorporating the Generic Reference Model (Hitchins, 2008) to identify a complete set of system functions whose use can be extended to analyse instances of functional interaction. An important tool is the *functional interaction type* concept, as this offers a way of assessing the relative difficulties associated with the functional interface to be able to identify suitable points for functional partitioning – the emphasis being on where not to draw a boundary (functional blocks) rather than where the boundary should be, leaving more options open to the system designer. The concept of being able to mitigate critical interactions allows a more flexible and pragmatic approach in comparison to the rigid decision process of Axiomatic design, able to deal with the inevitable trade-offs of system design. The functional interaction types also reflect the different natures of function in a system, which apart from the identification of feedback in Axiomatic design, is not recognised by the other methods.

In terms of physical design, apart from the logical partitioning according to the analysis of the functional interaction types, the Critical interaction modular design methodology allows for the designer to identify strategies of cohesion and dispersion in the physical architecture, which are directly related to particular system benefits in terms of quality attributes. Further consideration of lifecycle allows a further opportunity for critical interfaces to be mitigated across time, but also identifies through life benefits from specific architectural strategies. The consideration of how the architecture can positively influence non-functional requirements of design and lifecycle is unique across the three methodologies considered.

Finally, the Critical interaction modular design methodology offers a means of evaluating the architecture design, which is not addressed by Systematic design and easier to implement than the Information Axiom of Axiomatic design. It focuses on interfaces as this is the main parameter that a system designer at any given level of the system can influence. A strength of the architectural evaluation method for Critical interaction modular design methodology is that its equations take established modularity measures (degree and bridge modularity) and use critical interactions as a means of identifying the behavioural complication associated with the interfaces.

Strengths:

- Strong emphasis on context and requirements and its impact on architecture for both new systems and those based on legacy
- Flexible mechanism for structuring the functional design and development of the system that accounts for the complication of systems behaviour
- Identification of how the system design can be used to have a positive impact on quality attributes and through life considerations, such as maintenance
- Provides a means of assessing the quality of architecture, before models are developed to assess system effectiveness

Weaknesses:

- Method of assessment does not yet incorporate accurate views of what should be considered good and bad measures of modularity

10 Conclusions

10.1 Overview

The benefits of employing the concept of modularity, and the associated principles of simplicity and independence, in systems design are well researched and documented. However, a search of the literature reveals a lack of methodologies to allow its exploitation, which is reflected in a low level of acceptance by industry.

The research for this PhD has examined the nature of interactions within a systems architecture in order to provide guidance on how modularity could be implemented with the systems design process in order to achieve the certain desired benefits. This is incorporated in a methodology named *Critical interaction modular design methodology*.

Validation of this methodology by applying it to a representative problem would be very difficult to achieve as there is currently no accepted way of evaluating an architecture and the evaluation of a systems design is a multi-criteria problem where a sense of value will be subjective, preventing an objective assessment of the output of the methodology. Instead, this research has chosen to demonstrate the application of the methodology, justifying the logic of its individual steps against accepted principles. Several examples of different levels of complication are demonstrated and for the final example, the Critical interaction modular design methodology is compared with two of the main existing methodologies.

10.2 Current state of knowledge

A search of the literature has been performed to establish the current state of knowledge for this research; to reveal areas of particular need and any current gaps that can be exploited.

It is found that there are a variety of different views on what is meant by the *systems design process* and *systems architecture*; a system design process seeks to create an architecture by applying known architecture principles for a more favourable system design. There is a significant body of knowledge about the architecture principles of modularity, independence, simplicity and similarity. However, there is an apparent lack of science behind the practical application of these principles and a lack of understanding of the mechanism by which a system designer is able to contribute to an effective architecture. This may be why the theoretical benefits associated with a good architecture are not always achieved in practice and a practical approach is required.

The literature on existing systems design methodologies has been reviewed to determine how they implement system architecting principles; their scope,

efficacy and the degree to which they are currently in used by the systems engineering community. The literature on system architecture shows the current role of patterns, architecting strategies and specific architecting methods and how they are being employed in systems design. However a survey in the literature has shown that there are no generally accepted systems design methodologies, with methods having a very low level of awareness and acceptance by industry.

In the field of software much effort has gone into the generation of structured methods, but whilst these introduce a formal systematic approach they usually manage information rather than generate it, and do not generally address non-functional requirements and the effect of the physical environment.

Lack of acceptance of methods is likely to be, at least in part, due to the lack of identification of how a systems architect can design an architecture in order to achieve desired system attributes or outcomes. There is little in the literature that would allow an effective assessment of 'quality' of the architecture once it has been developed.

10.3 A proposed methodology for system architecture design

The specific *research question* developed for this research was:

> *"How can modular architectural principles be applied to the early system concept design to manage system effectiveness?"*

The methodology used in this research relies on three aspects:

- Firstly, that a methodology that employs established beneficial architectural principles can itself be assumed, by induction, to benefit from these principles
- Secondly, that showing a methodology can be applied across a variety of complicated problems is an indication of its suitability of application to problems in general
- Thirdly, that favourable comparison of the methodology with the current leading methodologies in this field provides evidence of an advance in the field of knowledge.

The methodology developed here has been developed on the foundations of a modular architectural approach, employing principles of simplicity, similarity, independence and modularity. Key in the application of architectural principles to systems problems, is how they can by employed to address functional, physical and behavioural challenges in such a way that a system designer understands the implications of an architectural decision on the solution outcomes. The proposed methodology characterises different types of problem or challenges

(*context types)* enabling guidance of how architectural principles can be used in different circumstances. The concept of *context types* also draws upon a variety of diverse systems engineering techniques, helping to provide a platform for a unified approach of systems engineering for addressing a variety of problems.

A key concept developed by this research is that of *functional interaction types*. It recognises that different functionalities present varied degrees of challenge to the system designer. In recognising this, the system designer is able to identify *fundamental blocks* of functionality that should be grouped in the physical design and the development of which should not be shared across organisational boundaries. Appropriate management of these *functional interaction types* will simplify the analytical and developmental challenges for the system designer and the system respectively, reducing overall risk and leading to beneficial behavioural characteristics such as reliability and resilience. Principles of *cohesion* and *dispersion* can then be overlaid on the functional design to provide a favourable system design in terms of the quality attributes of survivability, reliability, safety, security, maintainability, environmental compatibility and operability. Further consideration is then given to elements of the system design that should be either *associated* or *disassociated* to address qualities that will promote better lifecycle properties. At each stage, observance of fundamental blocks, cohesive/ dispersive influences and association/dissociation may result in conflict, where consideration of separation in a temporal dimension can be employed.

The purpose of this research has not been to produce a methodology that can derive an “optimal” architecture, but rather one that can suggest architectures that have been designed to favour certain quality attributes and reduce development risk. This research has argued that a system’s architecture is intrinsically linked to its quality attributes and therefore Arrow’s Impossibility Theorem (Arrow, 1951) suggests that there can be no such thing as a best system architecture in terms of the outcomes it achieves. A subset of possible architecture designs can be devised and evaluated in terms of both how well they address architectural principles and whether they will result in favourable outcomes. An evaluation of the quality of the architecture has been proposed, which builds on existing measures in the literature, to measure how well steps of the methodology have been performed.

10.4 Case examples

The research has looked at a number of case examples.

The first example, a simple system derived from a Lego Mindstorms project used on a continuous professional development course, demonstrated a simple process line concept. This example was used to explore initial application of developing ideas for the method, and also to help develop elements of the method.

The second example, based on a missile design, introduces a more complicated system that exercises many of the f*unctional interaction types* developed as part of this method. The complication of a missile design would normally occupy a specialist systems team of many engineers and so it has only been possible to address a small part of the design here, but enough to exercise the end-to-end concept design process.

The third example involves the design of a central heating system and it is used to compare the performance of this method with the other major methods available to the systems engineer/ architect at this time. A simple example, it never-the-less demonstrates many *functional interaction types* and provides a manageable problem to compare analyses across the different methods

The method has therefore been applied to simple and complicated examples and this has demonstrated that the method is straightforward in its application, capable of dealing with a range of system complication.

10.5 Method comparison

The central heating example of Chapter 9, took a simple system concept that is both easy to comprehend and is also well developed in terms of possible architectural configurations that have already been widely deployed in building designs; this makes it possible to assess the different architectures that might be proposed by each method.

The first conclusion from the analysis is that the existing methods only address part of the ‘front-end’ process of examining the context and deriving the requirements. At best Axiomatic design and Systematic design provide an outline of what needs to be addressed in determining requirements and addressing the functions needed. They also do not adequately address how to evaluate different designs in a systemic way.

Axiomatic design has a much polarised view of what is acceptable, which allows little room for trade-off. As it cannot address the often conflicting drivers of different quality attributes, the favoured solution, based largely on functional

criteria, is quite likely to have undesirable properties when other quality measures are eventually addressed later in the process. The architecture of central heating system favoured by Axiomatic design was not ‘central’; the rules of the method strongly favour a decentralised, uncoupled solution where possible and only if this is not possible is a decoupled solution considered. The criteria used to discriminate between solution options (the Information axiom) is a single parameter and there is no guidance to how this single order of merit can represent the multi-attribute space (again, Arrow’s Impossibility theorem implies that combining multiple attributes with different stakeholder preferences in a single parameter of value is not possible).

Systematic design provided a good method for developing the functional design from a requirement by the concept of functional flows. However, the development of an architecture was then seen as a largely creative step based upon past experience, the existence of which cannot be relied upon and the reliance on which will inevitably stifle innovation. The judgement of whether an architecture is a good one can only be made when the design has progressed to a stage where its quality attributes can be evaluated directly; this can therefore only be achieved late in the concept definition. Even at this stage, meeting desired quality attribute requirements will not guarantee a design that can be easily developed and operated through its life. In contrast, the Critical interaction modular design methodology, by employing a modular philosophy will simplify processes through the lifecycle and employs specific steps in it process to ensure this.

To conclude about this comparison activity, both existing methodologies have little guidance on eliciting requirements and are not clear on how to deal with existing legacy systems and human interactions. Systematic design prescribes an effective concept to develop a functional description, but provides no guidance on partitioning of functionality and, using creative rather than analytical methods evaluation can only be of effectiveness late in the process. Axiomatic design has firm adherence to functional independence and the need to identify complexity/complication in interfaces, but provides no guidance on generation the functional design and has an inflexible approach to architecture that does not suit large and flexible systems. In contrast, the Critical interaction modular design methodology has a strong emphasis on context and requirements for both new and legacy systems. It employs a flexible mechanism for structuring the functional, physical and lifecycle designs for reducing complication and improving quality attributes and through life considerations, such as maintenance. Finally, it provides a means of assessing the quality of architecture, which can be performed before models are developed to assess system effectiveness. It is on this last point that the methodology has a weakness as has not been possible to establish accurate views of what should be considered good and bad measures of modularity; this should be the subject of further research.

10.6 Reflection on Research Approach and areas for further work

Research is a process of exploration; a foggy problem (Obeng, 1995) which is characterised by uncertainty in what the objective should be and what potential solutions are in meeting the objective. It means that a linear approach of performing a literature review, data collection, analysis and write-up is not always appropriate. Such an approach may be applicable when there is a mature body of knowledge to base research upon and a clear methodology to apply, such as experiments to achieve data to prove or disprove hypotheses. However study into the field of system architecture is not a mature science and the scale of problems required to exercise methods proposed make it difficult to achieve validation of approaches. Instead an approach of exploration using iteration and review has been employed where:

- Ongoing research leads to new unanticipated areas of interest
- Further literature search then demonstrates potential shortcomings of the ongoing research
- There is a change focus of research in order to maintain a manageable scope within the context of a doctoral study.

Whilst these circumstances cannot necessarily be foreseen, the effort and potential disruption to schedule needs to be provisioned for. The research started with a broad remit and this remit has been narrowed and focussed as areas that are of most interest and provide greatest contribution to knowledge became apparent. Correspondingly the title has changed from:

"Conceiving an Innovative System Design Approach for Complex Systems in Modern Context"

To:

"A process for the application of modular architectural principles to system concept design for improved effectiveness".

The current title is more focussed in that it reflects a clearer view of focus that has developed over the term of the research in terms of intended strategy (modular concepts, rather than other possible architectural strategies), the stage of the process that it applies to (concept design, rather than the full systems process) and what is expected to be achieved (improvement of system effectiveness).

The original intention was to investigate whether architectural principles existed (or could be developed) that would enable the designer to address the various measures of effectiveness of a system simultaneously; an initial concept was for

a set of aligned architectures, each one of which addressed a particular measure. However, it became clear that in all but a few fortuitous cases, a focus on a completely aligned architecture is likely to involve unacceptable compromises in overall performance and effectiveness. Instead a concept has been developed that identifies a number of key critical architectural constructs that need to be preserved in the systems design process.

Perhaps the most difficult aspect to address has been the validation of the findings of the research. System design:

- deals with many facets of system performance, effectiveness and design properties and as a result, examples tend to be large projects involving many man-years of experienced effort, which is outside of the scope of a doctoral study
- is used across many domains of science and technology and to show application in all would be infeasible
- deals with multiple criteria for success, for which methods of evaluation often lead to an oversimplified or subjective nature of evaluation and claims of validation are very difficult to substantiate.

Development of a method that could be claimed to provide a valid evaluation would therefore be a research topic in its own right and likely to be far larger than the scope of a PhD study. Instead, this research has relied both on use of accepted principles from both research and practice of others and the demonstration of their application in example cases. In this regard, the demonstration of the method is not intended to provide validation, but to show how the methodology that has been researched and developed, copes with and manages typical system design problems. The case studies were chosen to be varied, but achievable. However, the very nature of the method developed is to create modular designs, and therefore the demonstrations do not provide a full view of how the evaluation methods perform when addressing highly integrated, non-modular designs. This would be suggested as part of further research into the method and in particular to calibrate what is perceived as ‘good’ and ‘bad’ in terms of the individual components of the *Relative architectural score* (Equation 9).

Firstly, the literature on systems complexity was used to identify that a prime focus should be on the interfaces of the architecture and that established concepts around modularity are a key tool in managing complication (section 5.1). No effective way of dealing with the relative complication of interfaces was identified in the literature and, by examination of typical system interfaces, the concept of *functional interaction types* has been proposed as a means of helping to establish which interfaces are likely to provide the most issues, from a

functional or behavioural perspective. The concept of *functional interaction types* provides the system designer with a means to decide on how to achieve a modular design. Known influences of spatial characteristics on architecture have been researched and incorporated in the method in order to facilitate to achievement of a good physical and lifecycle design. Finally existing methods of characterising architectural quality have been developed to incorporate the insights provided by the functional interaction types, and in particular *critical interaction types*, to develop an evaluation method that characterises the quality of the architecture in modular terms. As part of the development of the functional design and identification of the *functional interaction types* a notation has been devised. Whilst this has been developed along with the concept and improved to reflect issues encountered during this work, it would benefit from consideration of more examples and from the input from a variety of system designers to ensure that it is unambiguous in its notation and is easy to use. Such further work should lead to a more formally defined nomenclature and syntax definition.

Different problem contexts were examined to understand how they influence the architectural decisions that a system designer should make, and this has led to the definition a new concept of *context types*. The system designer can use *context types* to determine architectural strategies, but it also creates a framework for unifying various strands of systems thinking into one methodology as it directs the designer to consider established principles of problems types, systems dynamics, soft systems thinking, critical systems thinking and systems of systems, as well as considerations for legacy systems, safety critical systems and urgent operational requirements (section 4.3). An area for further study is how the risk score proposed from the context types might be used to provide a qualitative indication of project risk. Whilst a method has been proposed for the relative evaluation of an architecture, a system will eventually need to be assessed by its performance and effectiveness also. Further study could develop a method for the evaluation of effectiveness, although it should be recognised that the means to do this may not be available in the earlier concept stages. The structure of the Functional context diagram proposed in this research lends itself to an evaluation of the system against performance (system of interest), interoperability (wider system of interest), compatibility (environment) and acceptability (wider environment) – with robustness over time. The author has used this to develop a method to identify completeness of evaluation and this concept could be developed to add the additional *system evaluation* step for the Critical interaction modular methodology (as identified in Figure 32).

Further learning points from this research study have been the benefits that can be achieved by concerted and continuous effort. Effective research needs to maintain a coherent thought process in order to produce a coherent output. With part-time study it is sometimes difficult to maintain a line of thought from one

research session to another, and continuous effort through the thesis write-up stage is key to presenting a coherent story for similar reasons.

REFERENCES

Alexander, C. (1964). *Notes on the synthesis of form*. harvard paperbacks.

Alexander, C. (1979). *The Timeless Way of Building*. *New York Oxford University Press*.

Altshuller, G. (2002). *40 Principles: TRIZ keys to innovation* (Vol. 1). Technical Innovation Center, Inc.

Arrow, K. J. (1951). *Social Choice and Individual Values*. *The Economic Journal* (Vol. 2).

Ashby, W. R. (1991). Requisite variety and its implications for the control of complex systems. In *Facets of Systems Science* (pp. 405–417). Springer.

Bachmann, F., Bass, L., Klein, M., & Shelton, C. (2005). Designing software architectures to achieve quality attribute requirements. *IEE Proceedings-Software*, *152*(4), 153–165.

Baldwin, C., & Clark, K. (2004). Modularity in the design of complex engineering systems. *Complex Engineered Systems*, 175–205.

Barkan, P., & Hinckley, M. (1994). Benefits and limitations of structured methodologies in product design. In S. Dasu & C. Eastman (Eds.), *Management of Design Engineering and management perspectives* (pp. 163–178). Kluwer Academic Publishers.

Bass, L., Klein, M., & Bachmann, F. (2002). *Quality attribute design primitives and the attribute driven design method*.

Bayliss, C. Y., & Clark, K. B. (1997). Managing in an age of modularity. *Harvard Business Review*, 46–58.

Bayrak, A. E., Collopy, A. X., Papalambros, P. Y., & Epureanu, B. I. (2018). Multiobjective optimization of modular design concepts for a collection of interacting systems. *Structural and Multidisciplinary Optimization*, *57*(1), 83–94.

Bell, T. (1991). Incredible shrinking computers. *Spectrum, IEEE*.

Belton, V., & Stewart, T. (2002). *Multiple criteria decision analysis: an integrated approach*. Springer Science & Business Media.

Boehm, B. W., Brown, J. R., & Lipow, M. (1976). Quantitative Evaluation of Software Quality. In *Proceedings of the 2nd International Conference on Software Engineering* (pp. 592–605). Los Alamitos, CA, USA: IEEE Computer Society Press.

Boulding, K. E. (1956). General Systems Theory--The Skeleton of Science. *Management Science*, *2*(3), 197–208.

Bucciarelli, L. (1994). *Designing engineers*.

Buede, D. (2000). *The Engineering Design of Systems: Models and Methods. Wiley*.

Campagnolo, D., & Camuffo, A. (2009). What really drives the adoption of modular organizational forms? An institutional perspective from Italian industry-level data. *Industry and Innovation*, *16*(3), 291–314.

Checkland, P. (1981a). *Systems Thinking, System practice. Chichester Wiley*.

Checkland, P. (1981b). Systems Thinking, Systems Practice.

Checkland, P., & Poulter, J. (2006). *Learning for action : a short definitive account of soft systems methodology and its use, for practitioners, teachers and students.* Wiley.

Chestnut, H. (1965). *Systems Engineering Tools. New York Wiley*.

Chung, L., Gross, D., & Yu, E. (1999). Architectural Design to Meet Stakeholder Requirements. In P. Donohoe (Ed.), *Software architecture* (Vol. 12, pp. 545–564). Springer US.

Chung, L., & Leite, J. do P. (2009). On non-functional requirements in software engineering. In A. T. Borgida, V. K. Chaudhri, P. Giorgini, & E. S. Yu (Eds.) (pp. 363–379). Berlin, Heidelberg: Springer Berlin Heidelberg.

Churchman, C. (1968). *The Systems Approach.* New York: Delacourt Press.

Clark, K. B. (1987). *Managing technology in international competition: The case of product development in response to foreign entry*. Division of Research, Harvard Business School.

Clements, P. C., & Northrop, L. M. (1996). *Software Architecture: An Executive Overview.* Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst.

Coffield, B. (2016). The System of Systems Approach (SOSA). *INSIGHT*, *19*(3), 39–42.

Conker, R. S., Moch-Mooney, D. A., Niedringhaus, W. P., & Simmons, B. (2007). New Process for “Clean Sheet” Airspace Design and Evaluation. In *7th US/Europe ATM Seminar* (pp. 1–10).

Coplien, J. (1997). Idioms and patterns as architectural literature. *IEEE Software*.

Crawley, E., Weck, O. de, & Eppinger, S. (2004). *The influence of architecture in engineering systems (monograph)*.

Cross, N., & Roy, R. (1989). *Engineering design methods* (Vol. 4). Wiley New York.

Dijkstra, E. W. (1968). The structure of the “THE” multiprogramming system. In

The origin of concurrent programming (pp. 139–152). Springer.

Ehrlenspiel, K., Kiewert, A., & Lindemann, U. (2007). *Cost-efficient design*.

Eppinger, S. D. (1991). Model-based Approaches to Managing Concurrent Engineering. *Journal of Engineering Design*, *2*(4), 283–290.

Eppinger, S. D., & Pimmler, T. U. (1994). Integration Analysis of Product Decomposition. In *ASME Design Theory and Methodology Conference* (pp. 1–10).

Eppinger, S. D., & Salminen, V. (2001). Patterns of Product Development Interactions. In S. Culley (Ed.), *13th International Conference on Engineering Design, ICED 01* (pp. 283–290). Professional Engineering Publ.

Ericsson, A., & Erixon, G. (1999). *Controlling design variants: modular product platforms*. Society of Manufacturing Engineers.

Evbuomwan, N., & Sivaloganathan, S. (1996). A survey of design philospohies, models, methods and systems. *Journal of Engineering Manufacture*, 301–320.

Finger, S., & Dixon, J. (1989a). A review of research in mechanical engineering design. Part I: Descriptive, prescriptive, and computer-based models of design processes. *Research in Engineering Design*, *1*, 51–67.

Finger, S., & Dixon, J. R. (1989b). A review of research in mechanical engineering design. Part II: Representations, analysis, and design for the life cycle. *Research in Engineering Design*.

Fitts, P. M., & Washington, D. C. (1951). Human Engineering for an Effective Air Navigation and Traffic Control System. *National Research Council*.

Flood, R., & Carson, E. (1983). *Dealing with Complexity: An Introduction to the Theory and Application of Systems Science*. Springer.

Forrester, J., & Cambridge, M. A. (1961). Industrial Dynamics. *Productivity Press*.

Francalanza, E., Mercieca, M., & Fenech, A. (2018). Modular System Design Approach for Cyber Physical Production Systems. *Procedia CIRP*, *72*, 486–491.

Frey, D. D., & Dym, C. L. (2006). Validation of design methods: lessons from medicine. *Research in Engineering Design*, *17*(1), 45–57.

Fricke, E., & Schulz, A. P. (2005). Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle. *Systems Engineering*, *8*(4), 342–359.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design Patterns:

Abstraction and Reuse of Object-Oriented Design. *Lecture Notes in Computer Science*, *707*(Mvc), 406–431.

Garlan, D., & Shaw, M. (1993). An Introduction to Software Architecture. In *Advances in Software Engineering and Knowledge Engineering* (Vol. 1, pp. 1–40).

Gershenson, J., & Prasad, G. (1997). Modularity in product design for manufacturability. *International Journal of Agile Manufacturing*, *1*(1), 99–110.

Gershenson, J., Prasad, G., & Zhang, Y. (2003). Product modularity: Definitions and benefits. *Journal of Engineering Design*, *14*(3), 295–313.

Grabowski, H., Lossack, R., & El-Mejbri, E. (1999). Towards a universal design theory. In *Integration of process knowledge into design support systems* (pp. 47–56). Springer.

Gu, P., & Sosale, S. (1999). Product modularization for life cycle engineering. *Robotics and Computer-Integrated Manufacturing*, *15*(5), 387–401.

Hammond, F. (1969). System Architecture for an information process utility. *IEEE Computer Group News*.

Harrison, N. B., & Avgeriou, P. (2007). Leveraging Architecture Patterns to Satisfy Quality Attributes. *Lecture Notes in Computer Science*, *4758*, 263–270.

Hillier, B., Musgrove, J., & O'Sullivan, P. (1972). knowledge and design. *Environmental Design: Research and Practice*.

Hitchins, D. (1992). *Putting systems to work* (Vol. 325). Wiley Chichester.

Hitchins, D. (2008). *Systems Engineering: A 21st Century Systems Methodology*. Wiley.

Huang, C.-C. (2000). Overview of modular product development. *Proceedings-National Science Council Republic of China Part a Physical Science and Engineering*, *24*(3), 149–165.

Hyvärinen, A., & Oja, E. (2000). Independent component analysis: algorithms and applications. *Neural Networks*, *13*(4), 411–430.

ISO/IEC/IEEE. (2011). ISO/IEC/IEEE 42010: Systems and software engineering — Architecture description. International Organization for Standardization, International Electrotechnical Commission.

ISO 25010:2011. (2011). Systems and software engineering - Systems and software Quality Requirements and Evaluation (SQuaRE) - System and software quality models. ISO - International Organization for Standardization.

Jackson, M. (1994). Critical systems thinking: beyond the fragments. *System Dynamics Review*, *10*(2–3), 213–229.

Jackson, M. (1998). Formal methods and traditional engineering. *Journal of Systems and Software*, *40*(3), 191–194.

Jackson, M., & Keys, P. (1984). Towards a system of systems methodologies. *Journal of the Operational Research Society*, 473–486.

Jarratt, T. A. W., Eckert, C. M., Caldwell, N. H. M., & Clarkson, P. J. (2011). Engineering change: An overview and perspective on the literature. *Research in Engineering Design*, *22*(2), 103–124.

Johnson, G., Scholes, K., & Whittington, R. (2008). *Exploring corporate strategy: text & cases*. Pearson Education.

Juran, J. M. (1954). Universals in management planning and controlling. *Management Review*, *43*(11), 748–761.

Kahneman, D. (2011). *Thinking, fast and slow*. Penguin.

Kelly, M. J. (1950). The Bell Telephone Laboratories--An Example of an Institute of Creative Technology. *Proceedings of the Royal Society B: Biological Sciences*.

Kim, Y.-S., & Cochran, D. S. (2000). Reviewing TRIZ from the perspective of axiomatic design. *Journal of Engineering Design*, *11*(1), 79–94.

Klein, M. H., Kazman, R., Bass, L., Carriere, J., Barbacci, M., & Lipson, H. (1999). Attribute-based architecture styles. In *Software Architecture* (pp. 225–243). Springer.

Krishnan, V., Eppinger, S. D., & Whitney, D. E. (1991). Towards a cooperative design methodology: analysis of sequential decision strategies.

Langlois, R. N. (2002). Modularity in technology and organization. *Journal of Economic Behavior & Organization*, *49*(1), 19–37.

Lawson, B. (1980). *How designers think: the design process demystified*. Routledge.

Lin, C.-F. (1991). *Modern Navigation, Guidance, and Control Processing*. *Prentice Hall series in advanced navigation, guidance, and control, and their applications*. Prentice Hall.

Loch, C. H., & Terwiesch, C. (1998). Communication and Uncertainty in Concurrent Engineering. *Management Science*, *44*(8), 1032–1048.

Mackley, T. (2005). 4.3.1 Generic Measures of Effectiveness for Systems. *INCOSE International Symposium*, *15*(1), 610–622.

Mackley, T. (2008). The role of lifecycle systems in the through-life engineering of system solutions. *INCOSE International Symposium*, 691–705.

Mackley, T. (2015). A problem solving method using Context Types. In *IEEE international conference in Systems Engineering* (pp. 438–445).

Mackley, T., Deane, J., & John, P. (2010). Addressing the time dimension and agility in the provision of capability. In *2010 5th International Conference on System of Systems Engineering, SoSE 2010*.

Maeda, J. (2006). *The Laws of Simplicity*. MIT Press.

Maier, M. W. (1998). Architecting Principles for Systems-of-Systems. *Systems Engineering*, *1*(4), 267–284.

Malmqvist, & Axelsson. (1996). a comparative analysis of the theory of inventive problem solving and the systematic approach of Pahl and Beitz. In *Proceedings of ASME DTM'*.

Mendoza, G. A., & Martins, H. (2006). Multi-criteria decision analysis in natural resource management: a critical review of methods and new modelling paradigms. *Forest Ecology and Management*, *230*(1), 1–22.

Midgely, G., & In, B. (2006). Systems Thinking in Evaluation. *In Evaluation Pp American Evaluation Association*, 0–11.

Midgley, G. (2006). Systems thinking for evaluation. *Systems Concepts in Evaluation: An Expert Anthology*, 11–34.

MOD. (2005). *The Acquisition Handbook: A Guide to Achieving Defence Capability, Faster, Cheaper, Better and More Effectively Integrated* (6th ed.). London.

Morris, R., & Parnas, D. L. (1971). On the Ciriteria to Be Used in Decomposing Systems into Modules. Carnegie-Mellon University.

NASA. (2007). NASA Systems Engineering Handbook. *Systems Engineering*, *6105*(June), 360.

Obeng, E. (1995). *All Change! The Project Leader's Secret Handbook.*

Orton, J. D., & Weick, K. E. (1990). Loosely Coupled Systems: A Reconceptualization. *The Academy of Management Review*.

Pahl, G., Beitz, W., Feldhusen, J., & Grote, K. H. (2007). *Engineering Design: A Systematic Approach*. (K. Wallace & L. T. M. Blessing, Eds.), *Springer* (Vol. 3). Springer.

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, *15*(12), 1053–1058.

Perrow, C. (1999). Normal Accidents: Living with High Risk Technologies. New Brunswick. NJ: Rutgers University Press.

Price, J. (1999). Christopher Alexander's pattern language. *IEEE Transactions on Professional Communication*, *42*(2), 117–122.

Pugh, S. (1991). *Total design: integrated methods for successful product engineering*. Addison Welsey.

Rechtin, E. (1992). The art of systems architecting. *IEEE Spectrum*, *29*(10), 66–69.

Roman, G. C. (1985). A Taxonomy of Current Issue in Requirements Engineering. *Computer*, *18*(4), 14–23.

Roozenburg, N. F. M., & Cross, N. G. (1991). Models of the design process: integrating across the disciplines. *Design Studies*, *12*(4), 215–220.

Russell, D., & Xu, J. (2007). Service oriented architectures in the provision of military capability. In *UK e-Science All Hands Meeting*. Citeseer.

Sadraey, M., & Colgren, R. (2005). UAV flight simulation: credibility of linear decoupled vs. nonlinear coupled equations of motion. In *AIAA Modeling and Simulation Technologies Conference and Exhibit* (pp. 15–18).

Sako, M. (2003). Modularity and outsourcing: the nature of co-evolution of product architecture and organisation architecture in the global automotive industry. *The Business of Systems Integration*, 229–253.

Senge, P. (1990). The Fifth Discipline: The Art of Practice of the Learning Organisation. *New York Doubleday Currency*.

Senge, P. M., Kleiner, A., Roberts, C., Ross, R. B., & Smith, B. J. (1994). The Fifth Discipline Fieldbook. In *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization* (p. 593). Doubleday.

Sharman, D. (2004). Characterising Complex Product Architectures. *Systems Engineering Vol 7 No*, *1 SRC-G*, p35-59.

Shaw, M. (1995). Patterns for software architectures. In *Procedings of conference for Pattern languages of program design* (Vol. 1, pp. 453–462). Addison-Wesley.

Sillitto, H. (2014). *Architecting systems : concepts, principles and practice*. College Publications.

Simon, H. (1962). The Architecture of Complexity. *Proceedings of the American Philosophical Society*, *6*.

Smith, R. P., & Eppinger, S. D. (1997). Identifying controlling features of engineering design iteration. *Management Science*, *43*(3), 276–293.

Snowden, D. J., & Boone, M. E. (2007). A framework for decision making. *Harvard Business Review*, (November), 15–17.

Sobieszczanski-Sobieski, J. (1988). Optimization by decomposition: a step from hierarchic to non-hierarchic systems. In *NASA/Air Force Symposium on Recent Advances in Multidisciplinary Analysis and Optimization*.

Sosa. (2003). Indentifying Modular and Integrative Systems and their Impact on Design Team Interactions. *Journal of Mechanical Design*, 240–252.

Sosa, M. E., Eppinger, S. D., & Rowles, C. M. (2007). A network approach to define modularity of components in complex products. *Journal of Mechanical Design*, *129*(11), 1118–1129.

Spooner, C. (1971). A Software Architecture for the 1970s: Part I - The General Approach. In *Software - Practice and Experience* (p. Vol 1 p5-37). Oxford University Press.

Steward, D. V. (1981). Design Structure System: A Method for Managing the Design of Complex Systems. *IEEE Transactions on Engineering Management*, *EM-28*(3), 71–74.

Suh, N. (1990). *The Principles of Design.*

Suh, N. (1995). Design and operation of large systems. *Journal of Manufacturing Systems*.

Suh, N. (1997). Design of Systems. *CIRP Annals - Manufacturing Technology*.

Suh, N. (1998). Axiomatic Design Theory for Systems. *Research in Engineering Design - Theory, Applications, and Concurrent Engineering*, *10*(4), 189–209.

Suh, N. (2001). *Axiomatic Design: Advances and Applications*. Oxford University Press.

Taguchi, G. (1986). *Introduction to Quality Engineering*. Asian productivity association.

Tomiyama, T., Gu, P., Jin, Y., Lutters, D., Kind, C., & Kimura, F. (2009). Design methodologies: Industrial and educational applications. *CIRP Annals - Manufacturing Technology*, *58*(2), 543–565.

Tyree, J., & Akerman, A. (2005). Architecture decisions: Demystifying architecture. *IEEE Software*, *22*(2), 19–27.

Ullman, D. (2003). *The mechanical design process* (Third). McGraw-Hill Higher Education.

Ulrich, K. (1995). The Role of Product Architecture in the Manufacturing Firm. *Research Policy*.

Ulrich, K., & Eppinger, S. D. (2008). Product Design and Development. *McGraw-Hill Higher Education*, (4), 1–368.

Ulrich, W. (1987). Critical heuristics of social systems design. *European Journal of Operational Research*.

Urbaczewski, L., & Mrdalj, S. (2006). A comparison of enterprise architecture frameworks. *Issues in Information Systems*, *7*(2), 18–23.

Von Bertalanffy, L. (1950). The theory of open systems in physics and biology. *Science*, *111*(2872), 23–29.

Walden, D., Roedler, G., Forsberg, K., Hamelin, R., & Shortell, T. (2015). *Systems engineering handbook: A guide for system life cycle processes and activities*. Wiley.

Warfield, J. N. (1973). On Arranging Elements of a Hierarchy in Graphic Form. *IEEE Transactions on Systems, Man, and Cybernetics*, *SMC-3*(2).

Waring, A. (1996). *Practical systems thinking*. Cengage Learning EMEA.

Wasson, C. (2006). *System Analysis, Design, and Development: Concepts, principles and practices* (First). Wiley-Interscience.

Weick, K. E. (1982). Management of organizational change among loosely coupled elements. *Change in Organizations*, *Vol 375*, 408.

Wijnstra, J. G. (2001). Quality attributes and aspects of a medical product family. In *System Sciences, 2001. Proceedings of the 34th Annual Hawaii International Conference on* (p. 10–pp). IEEE.

Wolstenholme, E. (2003). Towards the defintion and use of a core set of archetypal structures in systems dynamics. *System Dynamics Review Vol*, *19*(1), 7–26.

Wolstenholme, E. (2004a). Using generic system archetypes to support system thinking. *System Dynamics Review*.

Wolstenholme, E. (2004b). Using generic system archetypes to support thinking and modelling. *System Dynamics Review*, *20*(4), 341–356.

Wong, H., Qaisar, S. U., & Ryan, M. J. (2016). Assessing design dependencies in modular systems. In *2016 Annual IEEE Systems Conference (SysCon)* (pp. 1–8). IEEE.

Woods, E., & Rozanski, N. (2005). Using architectural perspectives. In *Software Architecture, 2005. WICSA 2005. 5th Working IEEE/IFIP Conference on* (pp. 25–35). IEEE.

Wynn, D., & Clarkson, J. (2005). Models of designing. In J. Clarkson & C. Eckhart (Eds.), *Design process and improvement: A review of design* (pp. 34–59). Springer.

Yang, M. C. (2007). Design methods, tools and outcome measures: a survey of practitioners. In *Proceedings of IDETCCIE 2007* (pp. 217–225).

Yassine, A., Falkenburg, D., & Chelst, K. (1999). Engineering design management: an information structure approach. *International Journal of Production Research*, *37*(13), 2957–2975.

Yoshikawa, H. (1981). General design theory and a CAD system. *Man-Machine*

Communication in CAD/CAM.

Zehra, N. (2015). *Research Methodology*. medworldonline.com.

APPENDICES

Appendix A Oversized figures

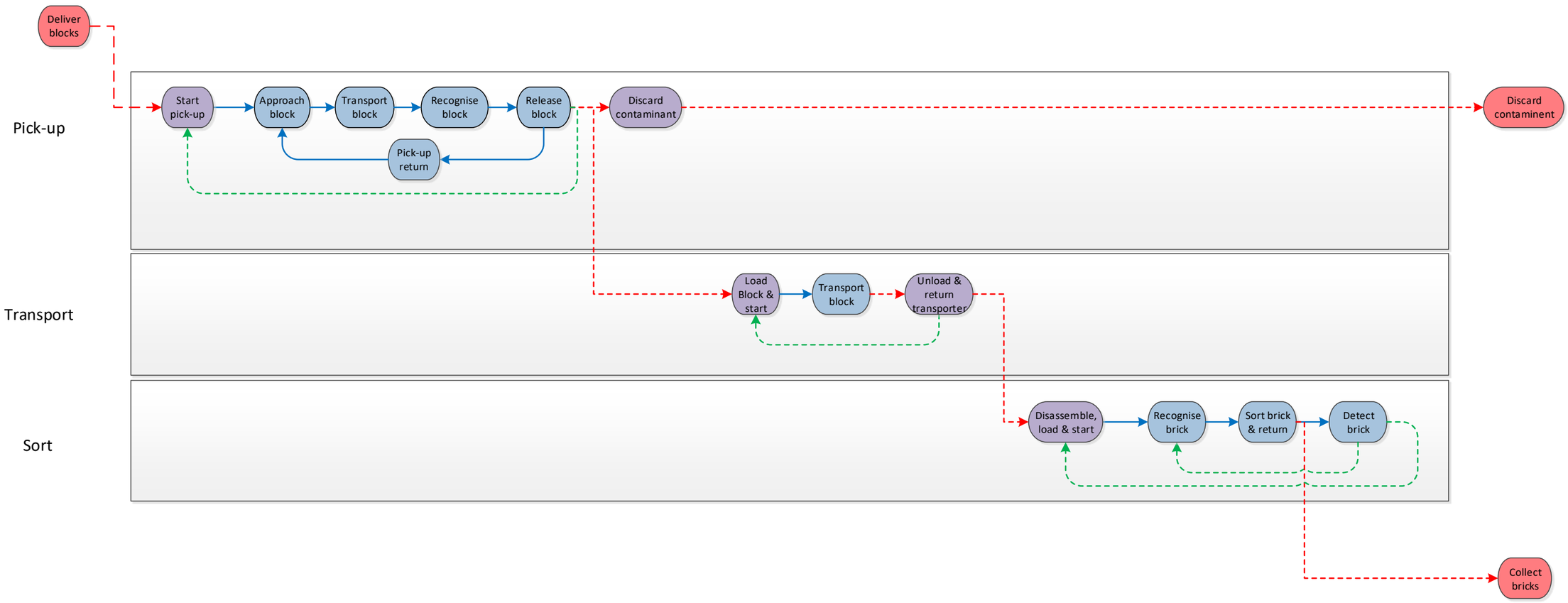


Figure A - 1: Lego Mindstorms example: Functional chain framework (option 1)

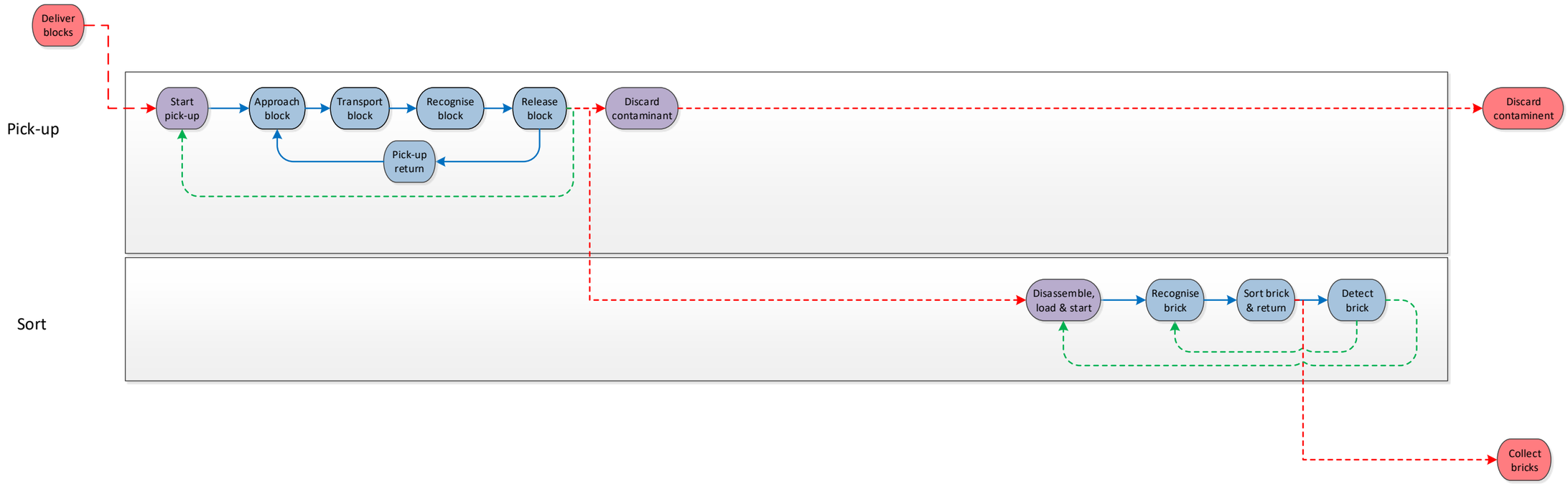


Figure A - 2: Lego Mindstorms example: Functional chain framework (option 2)

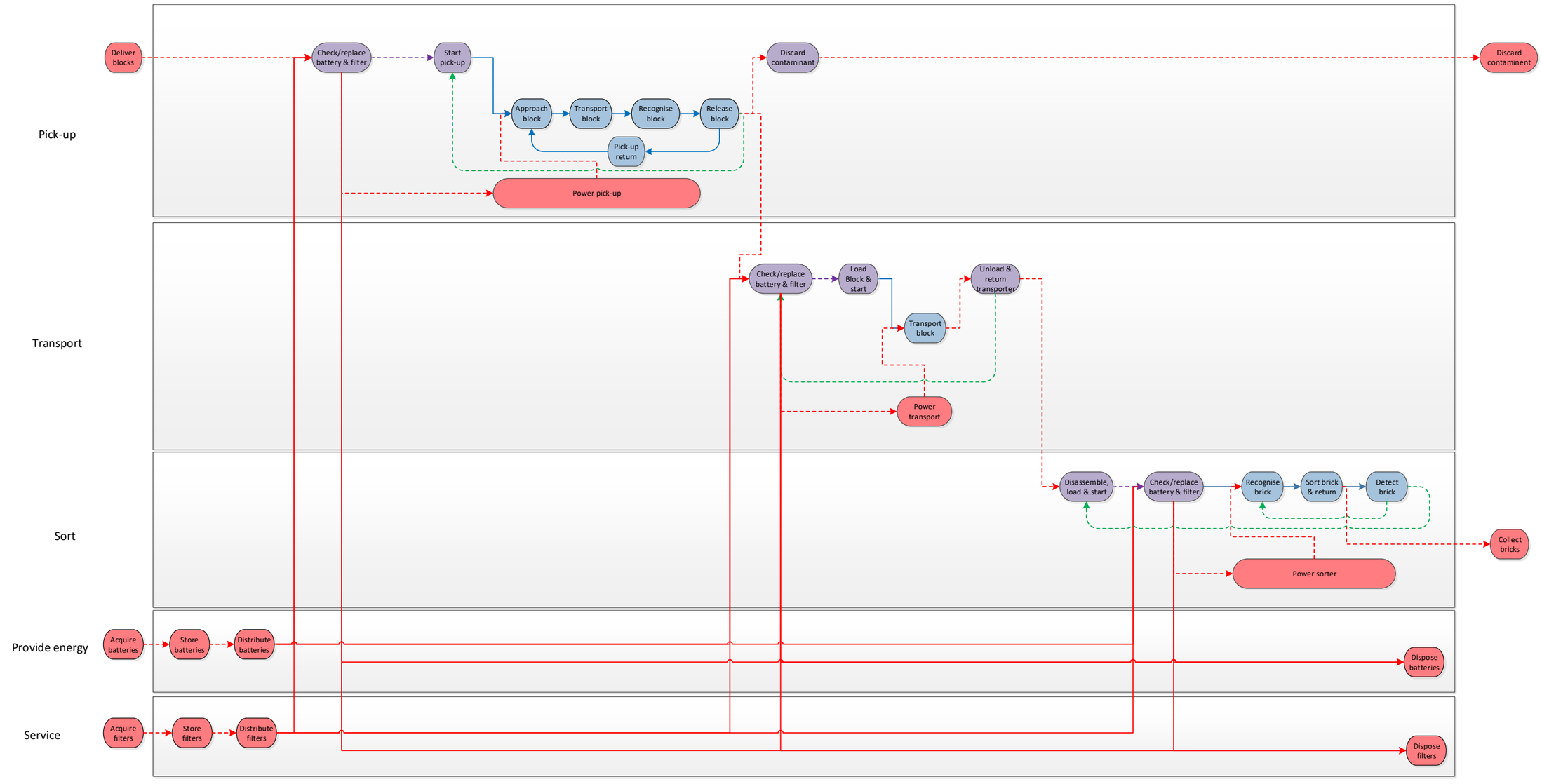


Figure A - 3: Lego Mindstorms example: Functional solution (option 1)

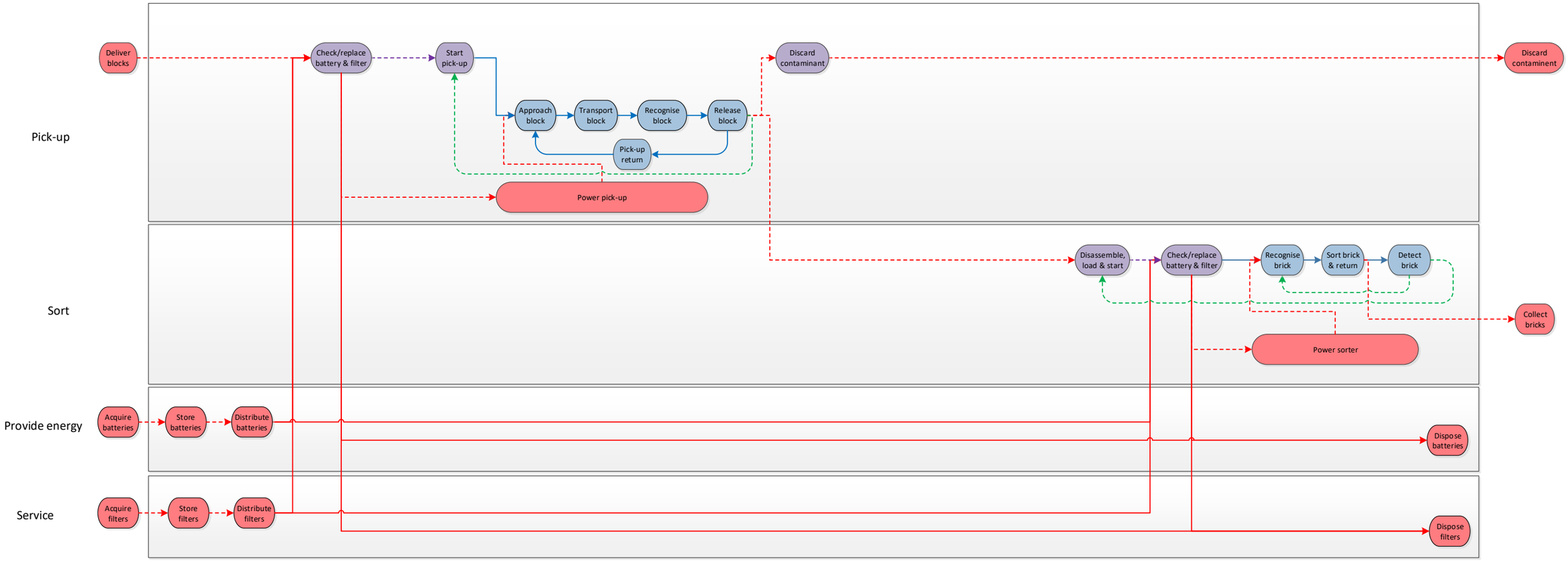


Figure A - 4: Lego Mindstorms example: Functional solution (option 2)

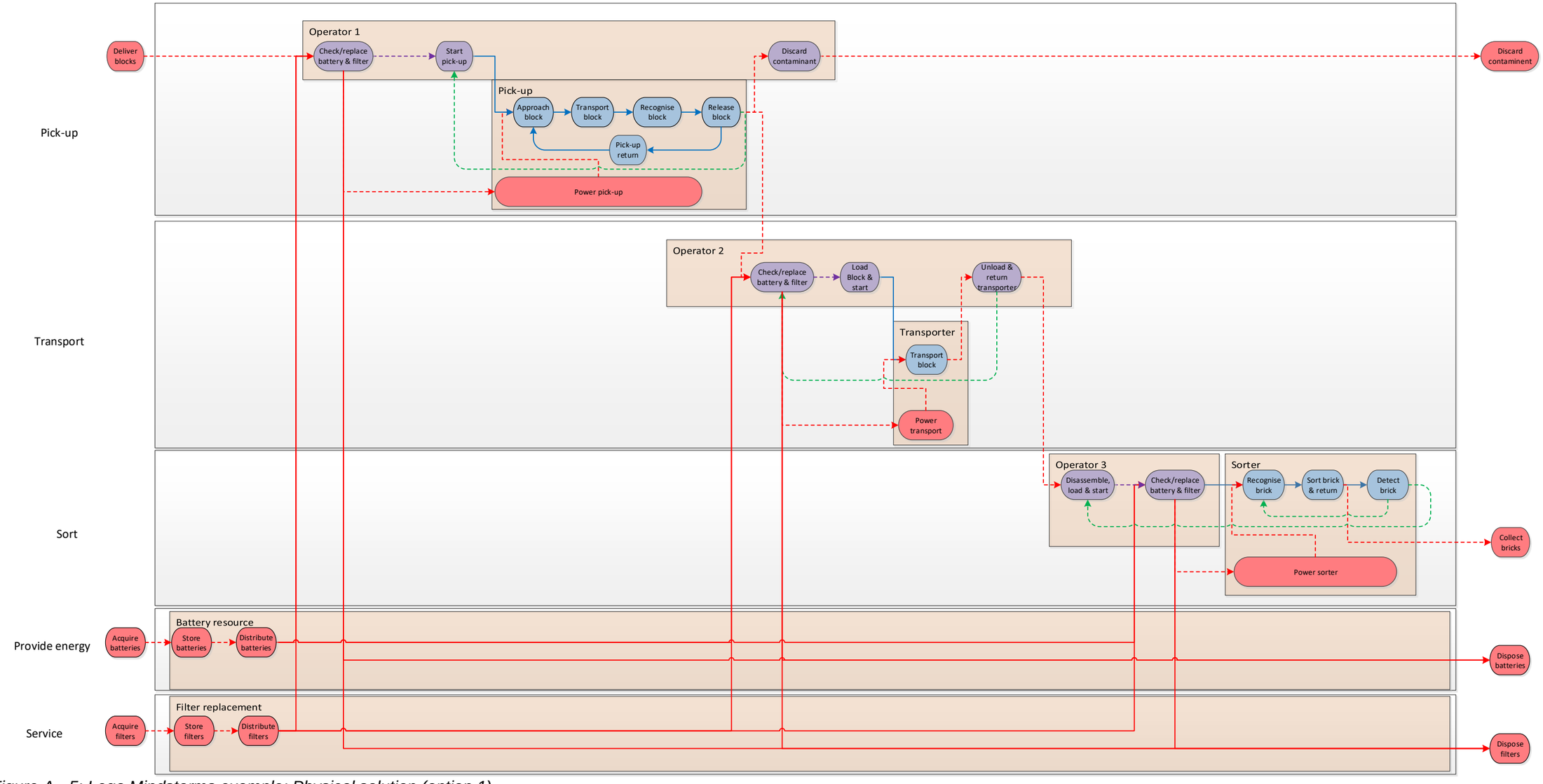


Figure A - 5: Lego Mindstorms example: Physical solution (option 1)

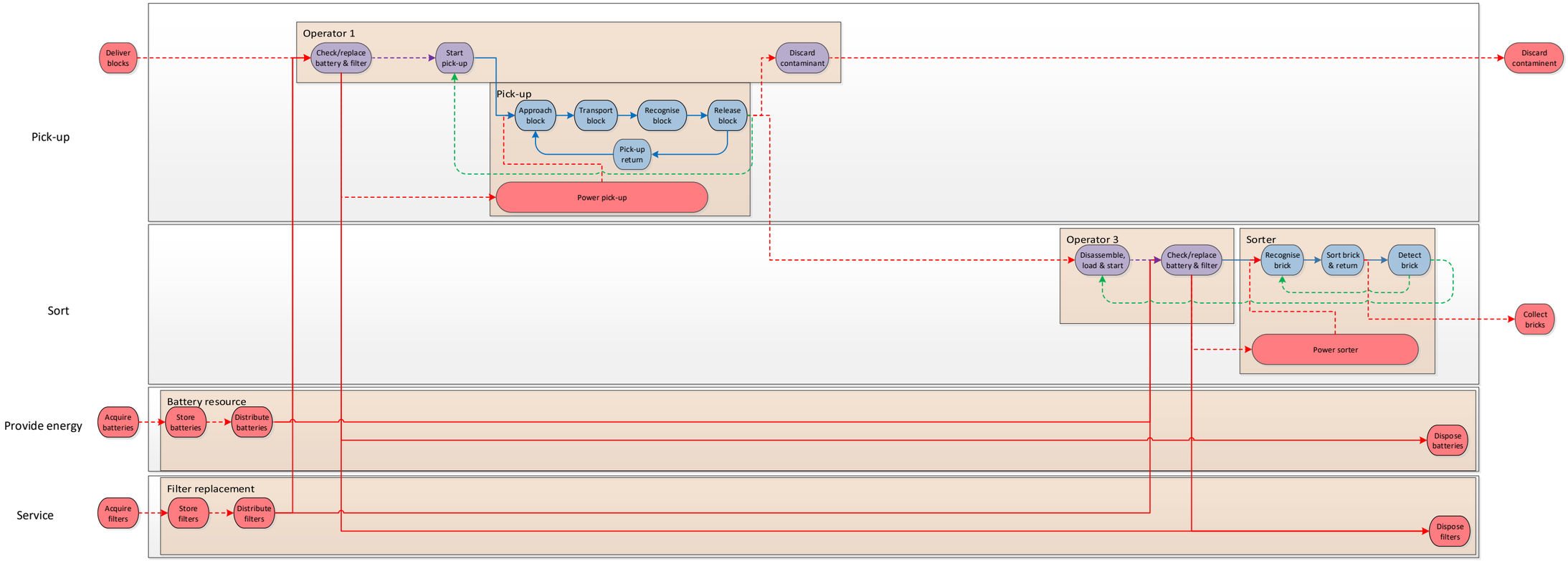


Figure A - 6: Lego Mindstorms example: Physical solution (option 2)

Functional Framework 1

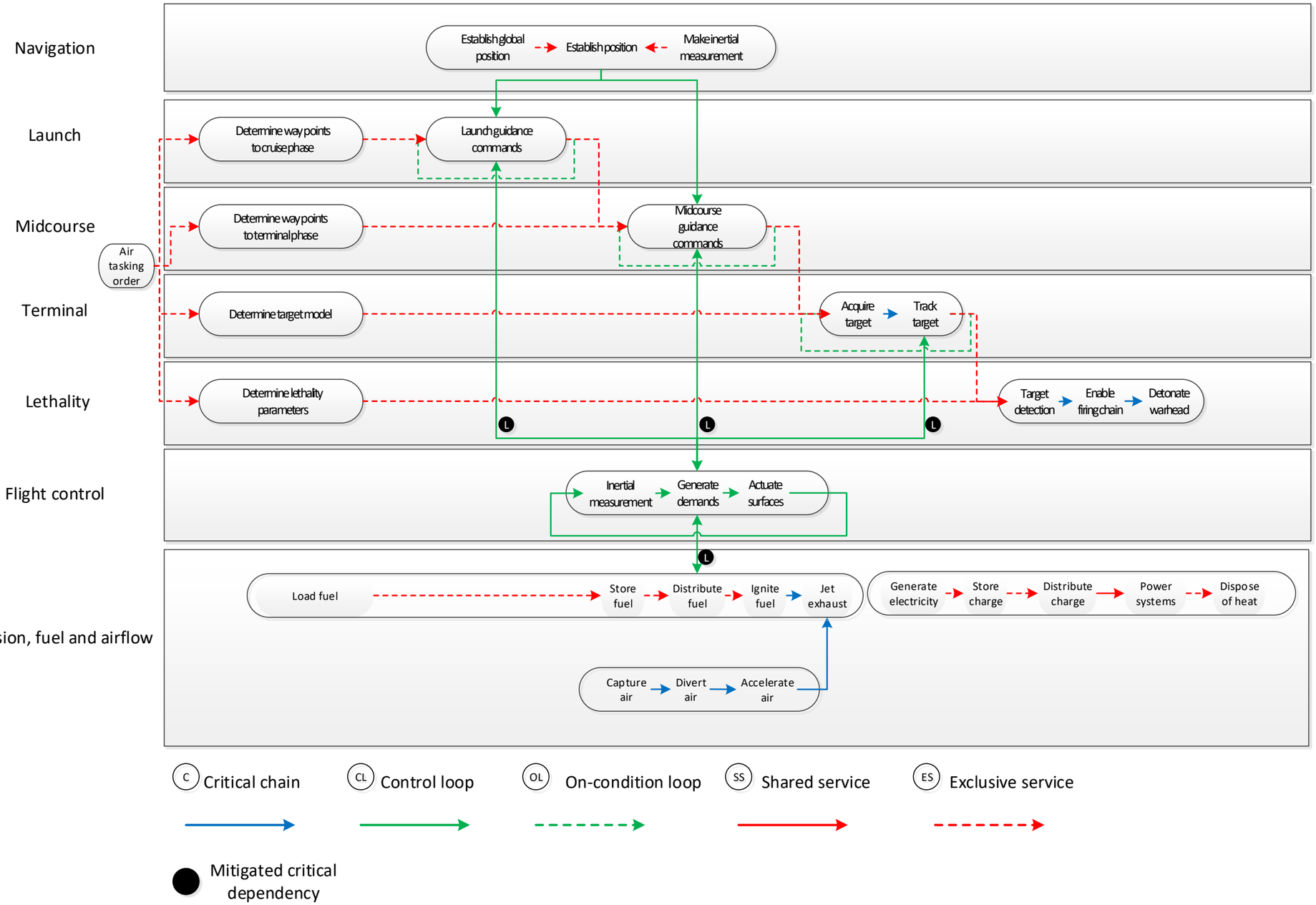


Figure A - 7: Missile example: functional solution (option 1)

Functional Framework 2

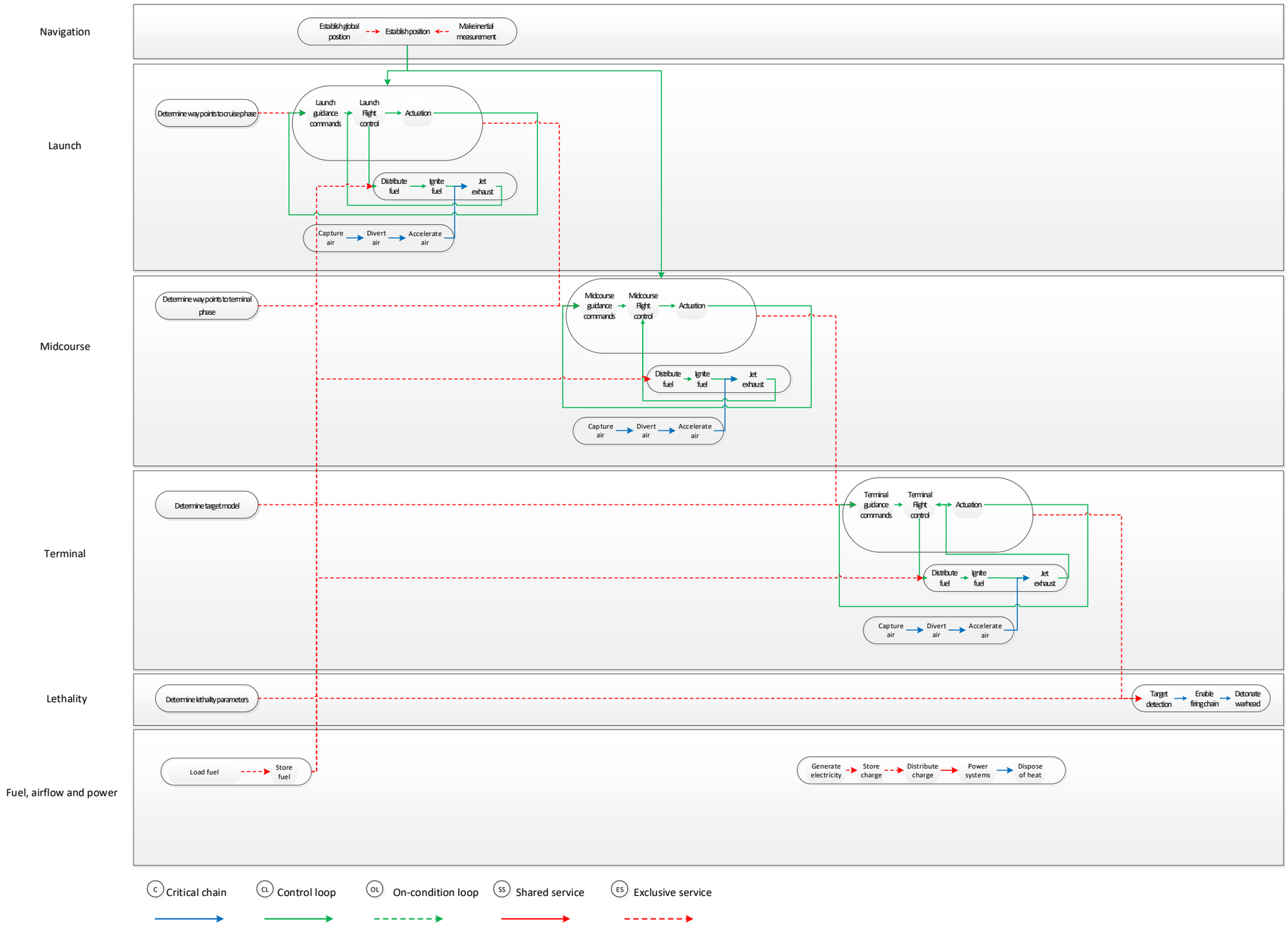


Figure A - 8: Missile example: functional solution (option 2)

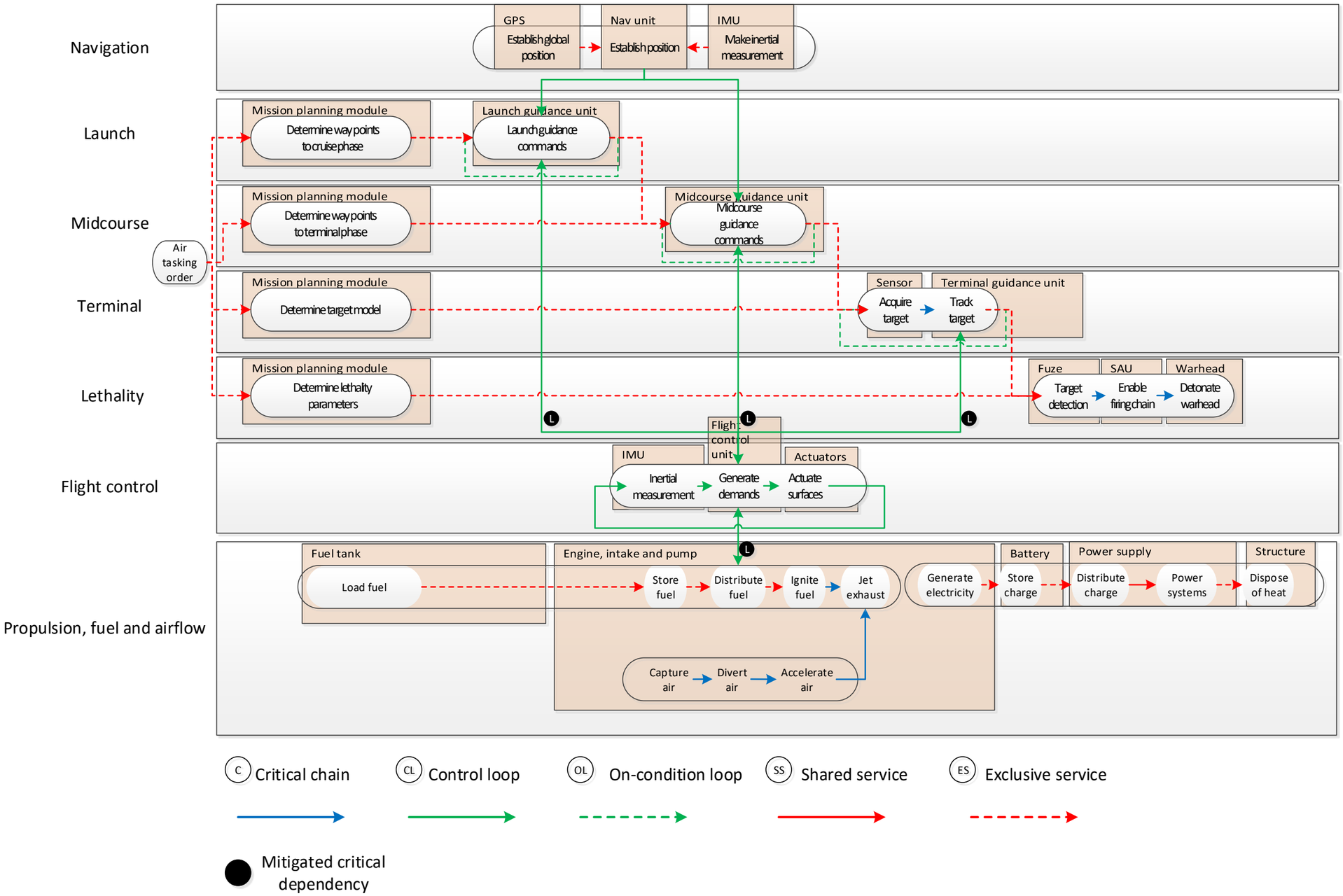


Figure A - 9: Missile example: physical solution (option 1)

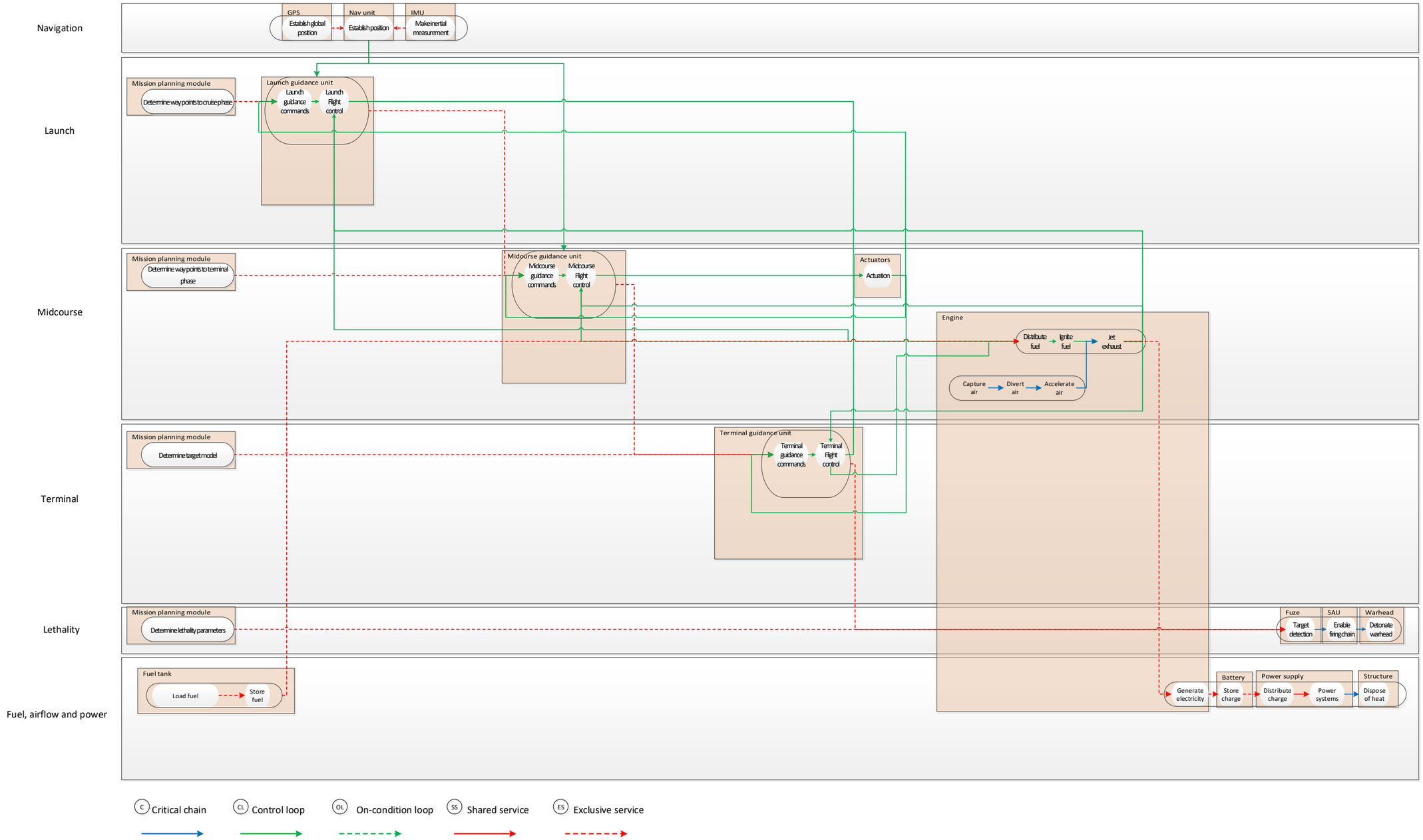


Figure A - 10: Missile example: physical solution (option 2)

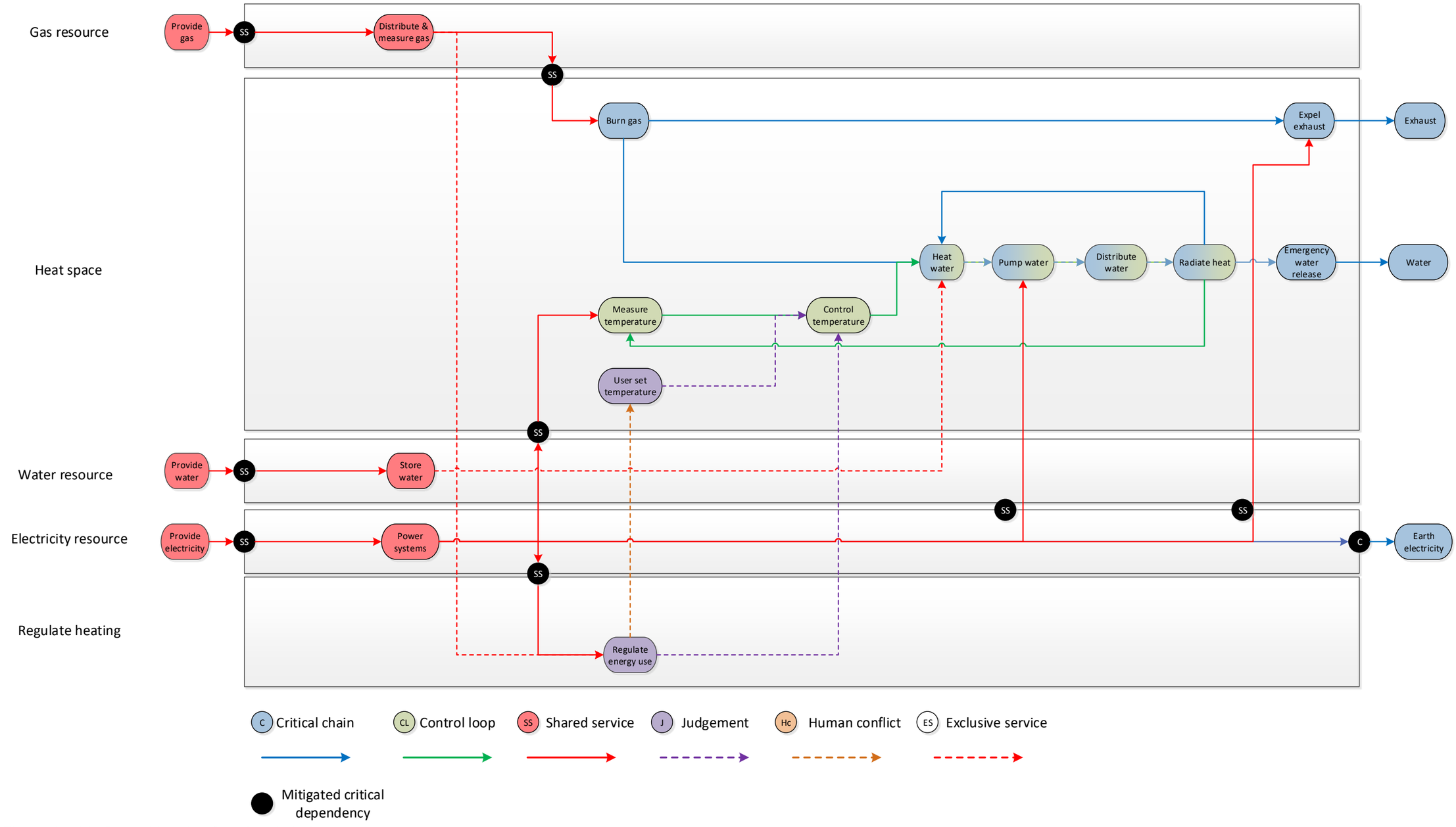


Figure A - 11: Central heating functional chains

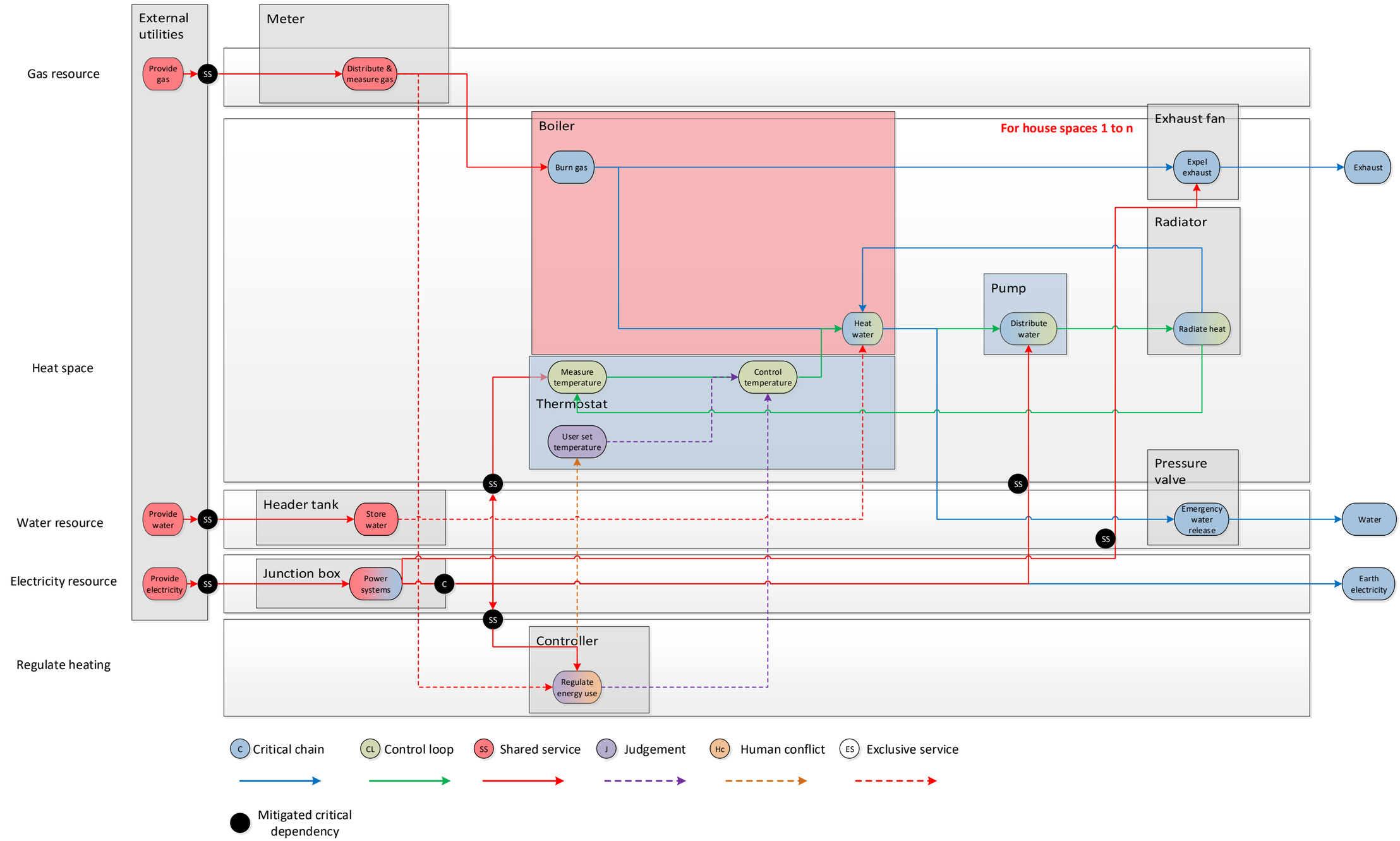


Figure A - 12: Initial mapping of heating functions to components

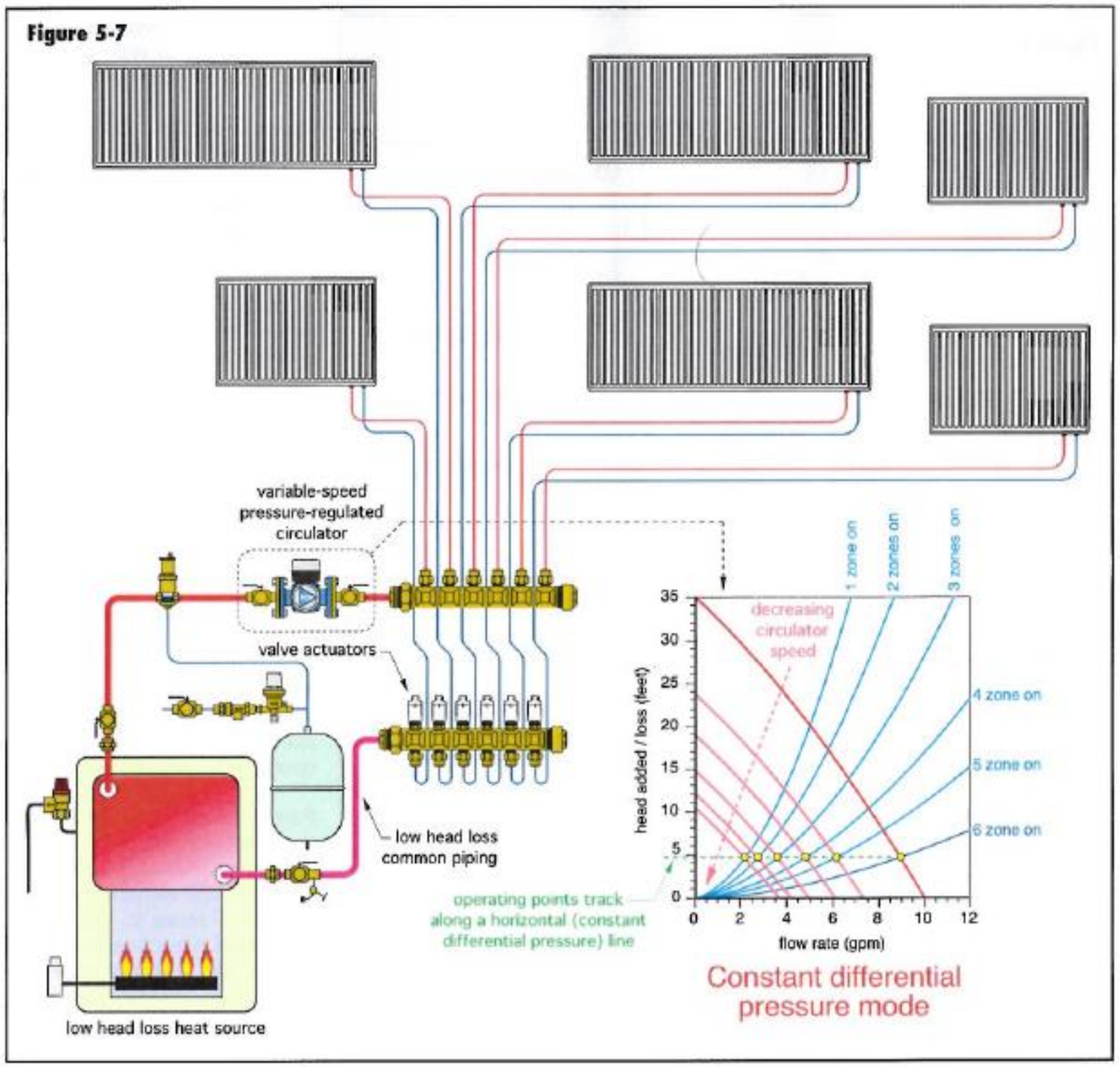


Figure A - 13: Example of a manifold design hydronic central heating system

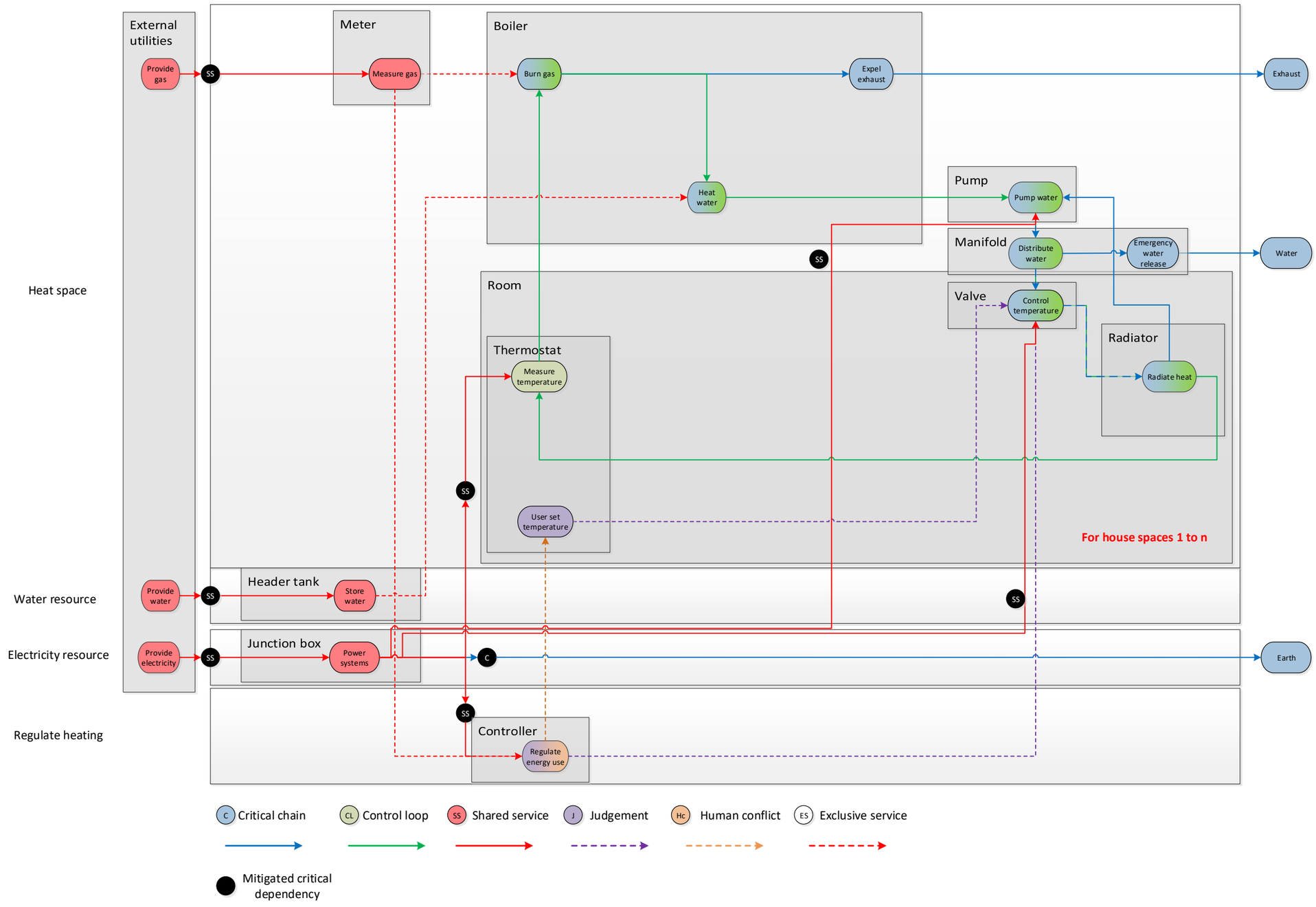


Figure A - 14: Alternative mapping of heating functions to components