




The cascade to complexity: modeling the evolution of first-of-a-kind systems in problem-solving design processes

Torben Beernaert ^{1,2,3}, Pascal Etman ², Maarten de Bock³ and Marco de Baar ^{1,2}

¹Dutch Institute For Fundamental Energy Research (DIFFER), Eindhoven, The Netherlands

²Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands

³ITER Organization, Route de Vinon-sur-Verdon, CS 90 046, 13067 St. Paul Lez Durance Cedex, France

Abstract

First-of-a-kind engineered systems often burst with complexity – a major cause of project budget and time overruns. Our particular concern is the structural complexity of nuclear fusion devices, which is determined by the amount and entanglement of components. We seek to understand how this complexity rises during the development phase and how to manage it. This paper formulates a theory around the interplay between a problem-solving design process and an evolving design model. The design process introduces new elements that solve problems but also increase the quantifiable complexity of the model. Those elements may lead to new problems, extending the design process. We capture these causal effects in a hierarchy of problems and introduce two metrics of the impact of design decisions on complexity. By combining and incorporating the Function-Behavior-Structure (FBS) paradigm, we create a new problem-solving method. This method frames formulation, synthesis and analysis activities as transitions from problems to solutions. We demonstrate our method for a nuclear fusion measurement system. Exploring different design trajectories leads to alternative design models with varying degrees of complexity. Furthermore, we visualize the time-evolution of complexity during the design process. Analysis of individual design decisions emphasizes the high impact of early design decisions on the final system complexity.

Keywords: Complexity, Model-based systems engineering, Function-behavior-structure modelling, Engineering design, Nuclear fusion

Received 10 May 2023
Revised 10 July 2024
Accepted 12 July 2024

Corresponding author
Pascal Etman;
L.F.P.Etman@tue.nl

© The Author(s), 2024. Published by Cambridge University Press. This is an Open Access article, distributed under the terms of the Creative Commons Attribution licence (<http://creativecommons.org/licenses/by/4.0>), which permits unrestricted re-use, distribution and reproduction, provided the original article is properly cited.

Des. Sci., vol. 10, e28
journals.cambridge.org/dsj
DOI: 10.1017/dsj.2024.27



1. Introduction

Nuclear fusion is among the largest research and development (R&D) projects. ITER, the first-of-a-kind experimental fusion reactor, currently under construction in the south of France, is one of the most complex machines in the world. We are concerned with the *structural* complexity of this machine, which relates to the amount, diversity and entanglement of technologies that need to be developed and integrated (Federici *et al.* 2016). Cost and duration of such R&D projects depend highly on these aspects. It is in the designers' best interest to manage structural

complexity. But this requires the answer to a fundamental question: How does this kind of complexity manifest itself during the design process? Structural complexity – hereafter simply referred to as complexity – is the main focus of this paper.

Uncertainty related to technical performance might be an important driver of complexity. In early development stages, designers have to make conceptual design decisions with only preliminary knowledge about their future implications (Simpson *et al.* 1998; Tan, Otto, & Wood 2017). Committing to decisions is necessary, however, to advance the engineering project. In the 18s-year-long conceptual design stage of ITER, the project had generated around 4,000 scientific publications. This number had reached 25,000 only 15 years after the conceptual design review. It seems that knowledge output sharply increases toward late design stages.

Unfortunately, a solution trajectory that is promising in early stages may give rise to new problems in later development. Such problems could have been avoided if the right knowledge was available at the time. However, instead of revising an earlier decision, designers have a tendency to solve such problems by adding more design elements (Adams *et al.* 2021). Those elements will contribute to complexity, further increasing the likelihood of problem propagation (Eckert, Clarkson, & Zanker 2004; Watson *et al.* 2019).

Mirror technology is a characteristic example of the exploratory nature of R&D. Modern fusion devices are equipped with many optical measurement systems that feature metallic mirrors in close proximity to the fusion plasma (Costley *et al.* 2005). The hostile environment causes multiple problems that have led to the addition of novel solutions: Extreme temperatures are managed through cooling systems (Salewski *et al.* 2008), while newly developed cleaning systems remove any optical contamination (Leipold *et al.* 2016; Ushakov *et al.* 2020; Stephan *et al.* 2021). But analysis shows that together, these cooling and cleaning systems will lead to an electrical grounding problem. This problem can be resolved by adding yet another element: a notch filter (Dmitriev *et al.* 2019). The complexity of the system, in terms of number of components and interactions, has grown significantly in this design trajectory.

We need a deeper understanding of how we can effectively solve unexpected problems. Even under extreme uncertainty, it is good practice to solve the right problem with a minimal amount of complexity. Our aim is to support designers in making complexity-conscious design decisions, by studying the following questions:

- Which prior design decisions have led to the manifestation of a particular design problem?
- Can we observe and control the complexity of an engineered system over time?
- How has each design decision contributed to complexity?

We seek to answer these questions via a newly proposed theoretical basis for the manifestation of design problems. The theory revolves around a model of the designed system that is subject to a series of design decisions. Each design decision adds new elements, thus expanding the model. We mathematically formulate two metrics that capture each decision's contribution to complexity. The theory is then specialized with the Function-Behavior-Structure (FBS) ontology, to create a systematic problem-solving method. We demonstrate this method for a nuclear fusion optical measurement system, illustrating system-level decision-making and complexity management.

We start this paper with a literature background on design problems and solutions, how to represent products and systems, techniques to manage complexity, definitions of complexity, dependency structure matrices and relevant applications of systems engineering in nuclear fusion. We then define the core ideas of this work by presenting our general theory of manifesting problems and complexity. In the following section, we frame the theory in the FBS ontology. The final section of this work comprises the demonstrations.

2. Background

Summers & Shah (2010) distinguish three aspects of design: the design problem, the design process and the design product. We refer to the latter as the design solution. It is this solution that seems to exhibit an ever-increasing complexity, and somehow new design problems seem to manifest themselves in it.

The design problem is a statement of needs, requirements and objectives, which in practice is often ambiguous and ill-structured (Jonassen, Strobel, & Lee 2006). It is therefore commonplace for designers themselves to interpret and formulate design problems (Daly *et al.* 2018). Problem exploration is an important aspect of the design process (Martinec *et al.* 2021; Obieke, Milisavljevic-Syed, & Han 2021) and even requires the generation of potential solutions (Zhang & Ma 2021). Indeed, it seems natural that problems and solutions coevolve over time (Dorst 2019). However, we have not seen systematic methods that trace arising problems during the design process.

There are varying ideas about what happens in the design process, but most of them have converged into three simple, yet powerful, notions: Function, Behavior and Structure (FBS) (Gero 1990; Umeda *et al.* 1990). *Function* is what the product is used for, *structure* is what it is, and *behavior* is what it does (Gero & Kannengiesser 2004). The FBS ontology has been at the basis of modern systematic engineering design approaches (Pahl *et al.* 2007), helping engineers to define essential design steps. One example is synthesis, the transition from function to a structure that is recurring throughout literature (Brunetti & Golob 2000; Mathias *et al.* 2011; Drave *et al.* 2020; Ramsaier *et al.* 2020).

Most current-day solutions are too complex to be viewed as a product of a single function, a single behavior and a single structure. They are better represented as systems, i.e., arrangements of interdependent products with multiple functions, behaviors and structures. The widely accepted V-model specifically defines decomposition and integration activities to manage the holistic systems engineering process (Forsberg & Mooz 1991).

A system's complexity is dependent in part on its architecture: a mapping between functions and physical components (Ulrich 1995). Designing an architecture is often framed as a single-level decision problem, for example, by function-means analysis (Johannesson & Claesson 2005), and can therefore only explore a limited solution space. Recent developments have moved to architecture design as a multilevel decision problem (Bussemaker, Ciampa, & Nagel 2020; Panarotto *et al.* 2022). These methods systematically generate diverse system architectures that can be further analyzed and optimized. However, these works neither quantify complexity nor analyze the contribution of each decision to the properties of an architecture.

In practice, large-scale engineering projects have to deal with more terms than just function, behavior and structure. Complexity in such projects is also a product of stakeholders, requirements and many nontechnical factors (Watson *et al.* 2019; Yang *et al.* 2019). Particularly in R&D of complex physical systems, budget and schedule are dominated by modeling, prototyping and verification procedures.

A modern way to deal with such heterogeneous complexity is Model-Based Systems Engineering (MBSE). MBSE envelopes techniques that apply models as central, ‘single source of truth’ artifacts that formalize heterogeneous and multi-disciplinary design information. The model can be inspected through viewpoints that provide relevant information to stakeholders, whether they are in physics, engineering, construction or procurement. When we speak of a model M , we mean the most abstract representation of a design that includes, but is not limited to, a system architecture. We refer to Estefan (2007) and Dickerson & Mavris (2013) for a complete overview of MBSE development, and to Madni & Sievers (2018) for a recent status quo of the field.

Systems theory is at the basis of MBSE. It models a system as $M = (E, R)$, with E a set of entities and R a collection of relations on E (Lin 1999). We refer to E and R as *design elements*. Depending on the modeling paradigm, elements may represent components, requirements, use cases, functions, variables, etc. The model $M = (E, R)$ connects systems engineering to network theory, by interpreting a system as a network with nodes E and edges R . This representation has enabled Sinha & de Weck (2013) to implement network metrics in a mathematical definition of structural complexity:

$$\zeta(M) = \sum_{i=1}^N \alpha_i + \left(\sum_{i=1}^N \sum_{j=1}^N \beta_{ij} A_{ij} \right) \frac{\varepsilon(A)}{N}, \quad (1)$$

where $N = |E|$ is the number of nodes in the network, α_i is the scalar internal complexity of node $e_i \in E$, β_{ij} is the scalar complexity of edge $(e_i, e_j) \in R$, A is the binary adjacency matrix of the network and $\varepsilon(A)$ is the matrix energy. If node e_i represents a subsystem, its internal complexity α_i may be acquired by applying Equation (1) recursively. Otherwise, the complexity α_i could be assessed via Technology Readiness Levels (TRLs) (Sinha & de Weck 2013). The matrix energy represents the ‘intricateness’ of the network structure and can be obtained through singular value decomposition (Klema & Laub 1980). So in Equation (1), complexity is determined by the number and internal complexity of the individual nodes and edges, and the structure of the network. Determining values for α and β is out of the scope of this work, so we assume 1 where possible.

Potts, Johnson, & Bullock (2020) argue that, while Equation (1) ‘is certainly a useful representation of an engineered system for systems engineers, the complexity of this representation is not necessarily the complexity of the system itself. Indeed, complexity can have many attributes, many of which are qualitative and intangible (Watson *et al.* 2019). Even experienced systems engineers find it hard to agree on the definition and importance of complexity (Potts *et al.* 2020).

Notwithstanding, Equation (1) has found some useful applications. Albeit in very limited samples, Sinha & de Weck (2013) have observed that development costs increase super-linearly with ζ . Raja, Kokkolaras, & Isaksson (2019) have used this metric for analysis of integrated load-carrying structures in an aerospace application. For a recent overview of other commonly used patterns and metrics

in network-based analysis of engineered systems, we refer to Paparistodimou *et al.* (2020).

Finally, it is worth highlighting advances in Dependency Structure Matrix (DSM) modeling techniques to manage complexity. A DSM is a matrix representation of a network model that can be analyzed, organized and annotated to reveal critical aspects of complex problems (Eppinger & Browning 2012) and has been an important tool in the study of complex architectures (Browning 2016; Wilschut *et al.* 2018). Hamraz *et al.* (2013) have combined DSMs and the FBS ontology into a tool for multi-domain change management.

How are the above systems engineering methods represented in nuclear fusion development? Most attention seems to be on requirements engineering (Cinque *et al.* 2020), and axiomatic design, a systematic design method (Suh 1990; Di Gironimo *et al.* 2015; Marzullo *et al.* 2017; Lanzotti *et al.* 2023).

We find only little work on system architectures: Grossetti *et al.* (2018) use MBSE to define the architecture of heating and current drives; Moscato *et al.* (2022) evaluate the functional performance of various conceptual tokamak cooling systems; Dongiovanni *et al.* (2018) have documented the systematic architectural design of a neutron diagnostic subsystem, explicitly accounting for complexity; and Beernaert *et al.* (2022) have used a system architecture as a framework to organize multiparty engineering collaborations. All are affected by complexity, although none of these works actively reduce it.

We identify a gap at the intersection of various fields. Problems and solutions seem to coevolve during the design process, but there is no method to formalize the dynamics of problem causality. Techniques proposed by Bussemaker *et al.* (2020) and Panarotto *et al.* (2022) can generate diverse system architectures from a multistage decision framework. However, they are unable to define the contribution of each decision to an architecture and do not include the evolution of problems, solutions and complexity in time.

The question of complexity remains at the heart of nuclear fusion development. Applications of system architecture techniques in nuclear fusion are sparse, but there seems to be a growing awareness of their benefits (Wolff *et al.* 2018). Being able to trace complexity as the design process unfolds will benefit the nuclear fusion project on all levels.

We have begun to address these gaps in previous work, by proposing a method to formalize problem manifestation and their impacts on engineering models (Beernaert *et al.* 2021). We continue this research by including the FBS ontology in our method. This leads to a more practical, systematic design method that we demonstrate for a nuclear fusion application.

3. Theory of problem-solving

This section constitutes the main contribution of this paper, introducing a novel theory of the relation between complexity and a problem-solving design process.

First, we must define the dynamics of a model-based design trajectory. Let M_t be a system design model, and observe its expansion over discrete time steps t . Refer to Figure 1. In this sequence of model instances, every transition from M_t to M_{t+1} represents a problem-solving design process that addresses a problem p and adds elements as a solution s . The collection of problems and solutions encountered in

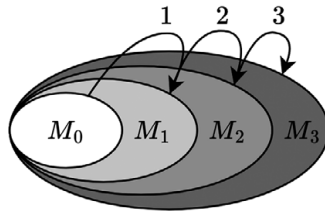


Figure 1. A model expands over three subsequent design processes. Model M_t is a subset of any subsequent models $M_{>t}$.

this sequence are denoted as P and S , respectively. Model M_0 represents the context of the system to be designed and defines the root design problem.

We define each of these design processes $d \in D$ by a tuple $d = (p, s, t)$. We refer to p as a local design problem and to s as a local design solution. Both p and s are sets of design elements, i.e., subsets of M . The time at which d is implemented is t . We assume an incremental design sequence, i.e., only a single d can be implemented at t . Design process d implies a mapping from problem to solution:

$$d : p \rightarrow s. \tag{2}$$

We assume that both problem and solution can be defined in terms of design elements, such as requirements, components and parameters. The problem elements p are already present in the model before the design process, so $p \subseteq M_t$. The solution, however, is described in newly generated elements. The model is expanded to cover these solution elements: $M_{t+1} = M_t \cup s$. We can also write M_t as the union of an initial model M_0 and the outcome of all design processes that have been implemented until time t :

$$M_t = M_0 \cup \{s \mid (p, s, t_d) \in D \text{ and } t_d \leq t\}. \tag{3}$$

Note that M_0 and s are sets of design elements, such that M_t becomes the union of design elements: $M_t = M_0 \cup s_1 \cup s_2 \cup \dots \cup s_t$.

A key assumption in our theory is the neutrality of design elements: An element does not in itself imply a design problem or a design solution. In fact, we build this theory on the presumption that a single element can be both a problem and a solution at the same time.

What is a solution for one process may well become a problem for another. Consider the chain of processes depicted in Figure 2. An initial model M_0 exhibits some design problem that is the input of a problem-solving process $d_1 = (p_1, s_1, t_1)$. The outcome of that process adds new elements to the model, expanding it to $M_1 = M_0 \cup s_1$. However, in a subsequent stage, those added elements pose a new problem p_2 that should be solved by d_2 . But even M_2 is not free of problems, and the design process has to continue.

We adopt the following vocabulary, so that these dynamics can be properly framed. All problems, except initial problems, *manifest* themselves as a result of a design decision. We distinguish *defined* and *discovered* problems, depending on the conditions for their manifestation. If the designer anticipated the problem manifestation when making the decision, and made the decision in awareness of that manifestation, we say that the problem has been defined by the designer. Conversely, if the designer did not anticipate the problem, we say that the problem

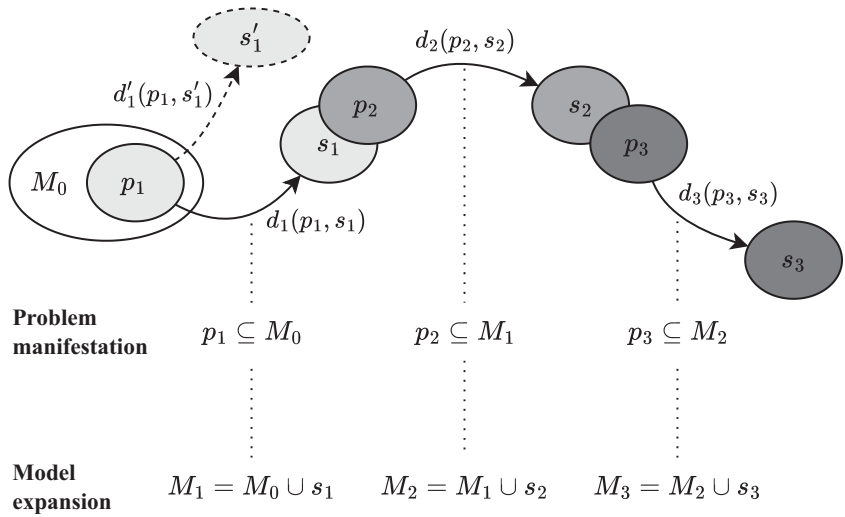


Figure 2. Design as a sequence of problem-solving processes. Each arrow represents one process as the mapping from a problem to a solution, both in terms of design elements of a model M . Each process contains the manifestation of a design problem, and the expansion of the design model with the design elements from a newly generated solution. It can happen that a new problem manifests itself due to those elements, which extends the sequence.

has been discovered. Defined problems manifest themselves relatively soon after a design decision – or simultaneously – while discovered problems manifest themselves at a later stage. Defined problems are not necessarily bad, as they can help designers in decomposing and tracing the design process. We presume, however, that discovered problems usually impact the design negatively, since by definition they were not accounted for in decision-making. This does not mean, however, that a discovered problem can always be avoided.

When discovering a problem, we can do either of two things. We can *solve* the problem by introducing new elements, continuing the chain and increasing the design complexity. The other option – which may be more elegant – is to *avoid* the problem by reconsidering an earlier design step. If we repeat process d_1 and consider a different outcome (i.e., s'_1 instead of s_1), we are on a different design trajectory where problems p_2 and p_3 are potentially avoided. This is visualized by design process $d'_1 = (p_1, s'_1)$ at some time increment when a different trajectory is adopted. Note that in Figure 2, the time t of design process d is omitted.

Our modeling method is developed to identify and analyze cause–effect relations between problem-solving processes. The input data that are required from designers include the problems that were encountered, in terms of design elements; the solutions that were designed, also in terms of design elements; and the intended mappings between them, which solution solves which problem.

3.1. Problem causality

We define that problem p is caused by solution s if solution s has introduced any element that is describing problem p . This is easily identified as a non-empty

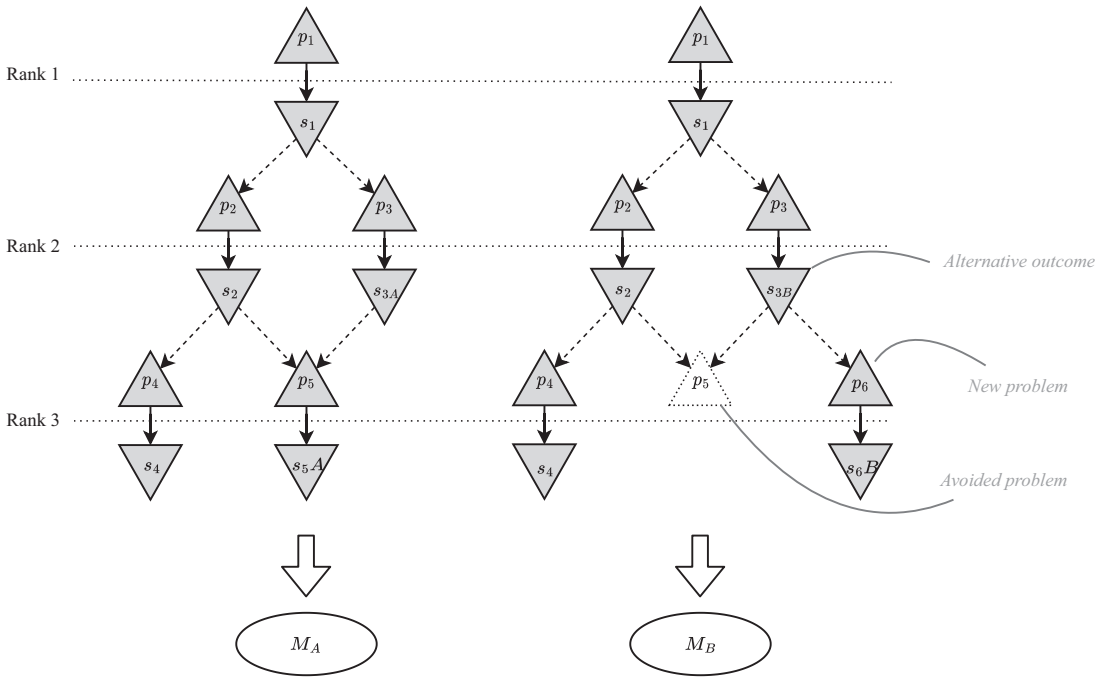


Figure 3. Two similar problem hierarchies represent how alternative decision-making leads to different design trajectories. Hierarchy B shows that p_5 is avoided by selecting an alternative solution to p_3 , at the cost of a new manifesting problem p_6 . The resulting design models are derived from Equation (3): $M_A = \cup\{M_0, s_1, s_2, s_{3A}, s_4, s_{5A}\}$ and $M_B = \cup\{M_0, s_1, s_2, s_{3B}, s_4, s_{6B}\}$.

intersection of those sets. We use the arrow notation to signify that p manifests itself due to s . The manifestation relations from solutions and the problem are collected in the set Q , where each $q \in Q$ is a tuple (s, p) and:

$$q : s \rightarrow p \Leftrightarrow s \cap p \neq \emptyset. \tag{4}$$

Figure 3 introduces the *problem hierarchy*, the causal structure of the design process that represents both design decisions and problem manifestations. This kind of graph, a directed acyclic graph, is necessary to study cause–effect relations (Rötzer *et al.* 2022).

The problem hierarchy $H = (E_H, R_H)$ is a tuple of nodes E_H and relations R_H . Problems and solutions form the nodes of the graph: $E_H = P \cup S$. They are visualized as triangles pointing upward and downward, respectively. The relations are design processes and manifestations: $R_H = D \cup Q$. A solid arrow from a problem to a solution signifies a design process, see Equation (2). As Figure 3 indicates, a single solution can cause multiple problems. A dashed arrow from a solution to a problem signifies manifestation, see Equation (4).

A node’s vertical position in the hierarchy represents its rank that can be obtained through partial ordering (Wallis 2012). Problems that do not manifest themselves due to any design decision have rank 1 and are called root problems. These are visualized at the top of the hierarchy. Problems that occur downstream, i.e., manifest themselves due to a series of decisions, have a high rank and are placed at the bottom of the hierarchy.

The problem hierarchy will be a helpful tool in design space exploration. It compactly visualizes the design reasoning steps that have led designers to a particular solution set. Critical decisions, those that lead to many problems, stand out at the top of the hierarchy.

Problem hierarchy A in Figure 3 could be a formalization of the mirror technology design process, given as an example in the introduction to this paper, if we interpret that:

- p_1 is the need to route light in proximity to the fusion plasma;
- s_1 is the use of a metallic mirror, a response to p_1 ;
- p_2 is the risk of an overheating mirror, a direct consequence of s_1 ;
- p_3 is the problem of mirror surface contamination, also a consequence of s_1 ;
- s_2 represents a liquid cooling system, addressing p_2 ;
- s_3 represents a cleaning system, addressing p_3 ;
- p_4 is a problem only related to the cooling system, e.g., water leaks;
- s_4 addresses the water leak problem through the use of dedicated seals;
- p_5 is the electrical grounding issue that arises from the combination of the cooling system s_2 and the cleaning system s_3 ; and
- s_5 is the notch filter that is proposed to resolve p_5 .

This visual representation emphasizes the far-reaching, and potentially underestimated, consequences of the decision to use metallic mirrors.

The formal description of the problem hierarchy $H = (E_H, R_H)$ allows us to define various sets that will support subsequent analyses. For any node $h \in E_H$, we can collect all adjacent input and output nodes in sets $\mathcal{X}(h)$ and $\mathcal{Y}(h)$, respectively:

$$\mathcal{X}(h) = \{h' \in E_H \mid (h' \rightarrow h) \in R_H\} \tag{5}$$

and

$$\mathcal{Y}(h) = \{h' \in E_H \mid (h \rightarrow h') \in R_H\}. \tag{6}$$

For example, node s_2 in hierarchy A of Figure 3 has a single input node and two output nodes: $\mathcal{X}(s_2) = \{p_2\}$ and $\mathcal{Y}(s_2) = \{p_4, p_5\}$. We furthermore define the recursive sets $\mathcal{X}^\infty(h)$ as the nodes that can reach h , and $\mathcal{Y}^\infty(h)$ as the nodes that can be reached from h :

$$\mathcal{X}^\infty(h) = \bigcup_{h' \in \mathcal{X}(h)} (\{h'\} \cup \mathcal{X}^\infty(h')) \tag{7}$$

and

$$\mathcal{Y}^\infty(h) = \bigcup_{h' \in \mathcal{Y}(h)} (\{h'\} \cup \mathcal{Y}^\infty(h')). \tag{8}$$

For example, that same node s_2 can be reached from the three nodes $\mathcal{X}^\infty(s_2) = \{p_1, s_1, p_2\}$ and can reach the four nodes $\mathcal{Y}^\infty(s_2) = \{p_4, s_4, p_5, s_{5A}\}$

The sets \mathcal{X}^∞ and \mathcal{Y}^∞ are essential to understand the precedents and consequences of the way designers solve problems. Now that we have formalized the dynamics of problem-solving, we can investigate its relation to the complexity of the design model.

3.2. Complexity

We can use Equation (1) to calculate the complexity of the model as the design process unfolds. If the complexity of any network model M can be written as $\zeta(M)$, then we can substitute Equation (3) to define complexity as a function of time:

$$\zeta(M_t) = \zeta(M_0 \cup \{s \mid (p, s, t_d) \in D \text{ and } t_d \leq t\}). \quad (9)$$

Note that solutions s are sets of model elements.

Equation (9) can be used to plot the whole system's complexity development over a sequence of design iterations. But what can this tell us about the contribution of each individual design process d ?

It is helpful to introduce three sections of the model M , as a function of design process $d = (p_d, s_d, t_d)$. First, there is $M^-(d)$, the subset of design elements that has led to the problem p_d . The elements in $M^-(d)$ are added by the solutions in $\mathcal{X}^\infty(p_d)$, i.e., those solutions that have led up to p_d :

$$M^-(d) = M_0 \cup \mathcal{X}^\infty(p_d). \quad (10)$$

Second, there is $M^+(d)$, which adds the design elements that are generated by process d :

$$M^+(d) = M^-(d) \cup s_d. \quad (11)$$

Finally, we can add the design elements that are generated by follow-up processes, for which process d is partly responsible. These are the processes that deal with the problems that manifest themselves due to s_d :

$$M^{++}(d) = M^+(d) \cup \mathcal{Y}^\infty(s_d). \quad (12)$$

We use the collections $M^-(d)$, $M^+(d)$ and $M^{++}(d)$ to characterize the evolution of the complexity as a function of the design choices. We define two impact factors as

$$I_L(d) = \zeta(M^+(d)) - \zeta(M^-(d)) \quad (13)$$

and

$$I_G(d) = \zeta(M^{++}(d)) - \zeta(M^-(d)). \quad (14)$$

The local complexity impact I_L is the difference in system complexity before and after d and therefore reflects the direct contribution of solution s_d . But what about knock-on effects? If s_d is a 'bad' design that leads to many problems, and solving those problems would lead to more complexity, we would like to retrace those effects to d . The global complexity impact I_G does this, by adding the complexity that was added due to manifested problems.

In practice, designers do not know beforehand whether a particular design decision will lead to undesired problems. Such effects are easily underestimated. Only in later stages will the true gravity of early-stage decisions appear (Tan *et al.* 2017). This is nicely captured in I_G , since this metric depends on $\mathcal{Y}^\infty(d)$. This set increases as more and more downstream problems are solved. Therefore, I_G will increase over time. An illustration is provided in the demonstration section.

If we view complexity ζ as just another attribute that depends on the design, the theory becomes more widely applicable. We can replace complexity ζ by any attribute X that depends on the design model, and Equations (9)–(14) can still be applied. Considering, for example, the various Design-for- X studies, the theory presented above allows traceability of any performance indicator over time and can distill desired and undesired contributions of individual design decisions.

We have introduced a theory of how design models develop through a series of problem-solving processes. Problems that manifest themselves due to a prior decision can be easily identified by overlapping design elements. This dynamic defines how the model M develops over time and for different solution alternatives. Additions to the model cause its complexity to increase, as is captured in two derived network metrics.

3.3. Algorithmic approach

We summarize the theory presented into an algorithmic problem-solving method. The following steps include problem identification, causal analysis, complexity assessments and design revisions. They are intended to guide designers through the various decision branches in the design process.

1. **Initialize design model and problem hierarchy.** Initialize the problem hierarchy $H = (E_H, R_H)$. The nodes of H are $E_H = P \cup S$, with problems $P = \emptyset$ and solutions $S = \emptyset$. The relations of H are $R_H = D \cup Q$, with design processes $D = \emptyset$ and manifestations $Q = \emptyset$.
2. **Discover a design problem.** Analyze the design model M . Can a design problem be discovered in M ?

Yes → Describe the discovered problem p in terms of design elements of M and add it to the set of design problems: $p \subseteq M$ and $P = P \cup \{p\}$. Continue with Step 3.

No → The design is finished.

3. **Trace problem causality.** The elements in p might have been introduced by earlier solutions. The causal relations between earlier solutions and the problem p can be identified mathematically and added to Q :

$$Q = Q \cup \{(s, p) \mid s \in S \text{ and } s \cap p \neq \emptyset\}.$$

4. **Avoid the problem.** Investigate whether revising an earlier decision could circumvent

p . First, collect the root problems P_r that have led to the manifestation of p . This collection is $P_r = P \cap \mathcal{X}^\infty(p)$. The problems P_r were addressed in an earlier design stage with limited knowledge about their consequences, namely, in design processes $D_r = \{(p', s', t') \in D \mid p' \in P_r\}$. Given the current knowledge, is there a good alternative to the outcome of any $d_r \in D_r$ that could avoid p ?

Yes → Retrace the design trajectory to d_r by executing Step *. Then, shift from p to p' and continue with Step 5.

No → Continue with Step 5.

5. **Add a solution.** Generate a set of solution candidates, express them in terms of design elements and make a selection. Formalize a new design process $d = (p, s, t)$ for the selected candidate s at current time t . Add the solution s to the model and to the problem hierarchy. If s implies follow-up problems that must be managed, add these *defined* problems P_d to the set P . Update the sets as follows:

$$\begin{aligned} M &= M \cup s, \\ S &= S \cup \{s\}, \\ D &= D \cup \{d\} \text{ and} \\ P &= P \cup \{p_d \in P_d \mid p_d \subseteq M\}. \end{aligned}$$

6. **Evaluate earlier decisions.** The addition of s and the corresponding increase in complexity are a consequence of earlier decisions. It is likely that the complexity was not accounted for when those decisions were made. Therefore, we recommend to evaluate the complexity contribution of prior design processes in the problem hierarchy. The design processes that have indirectly caused the complexity in s are given by $D_r = \{(p', s', t') \in D \mid s' \in \mathcal{X}^\infty(s)\}$. For each process $d_r \in D_r$, compute complexity metrics $I_L(d_r)$ and $I_G(d_r)$. The local $I_L(d_r)$ is the complexity impact that was expected at time t' , while the global $I_G(d_r)$ represents the *actual* impact at this moment. Therefore, a process that underestimated future complexity impact at time t' is indicated by $I_G(d_r) \gg I_L(d_r)$. Given the current complexity impact of decision d_r , could you revise that decision?

Yes → Retrace the design trajectory to d_r by executing Step *. Then, shift from p to p' and continue with Step 5.

No → Continue with Step 7.

7. **Address open problems.** The problems P_d that were defined in Step 5 need to be addressed. Defined but unaddressed problems are given by $P_o = \{p \in P \mid \mathcal{Y}(p) = \emptyset\}$. Are there any open problems, i.e., is P_o non-empty?

Yes → Shift to an open problem $p_o \in P_o$ and continue with Step 3.

No → Continue with Step 2.

- * **Retrace design trajectory.** Steps 4 and 6 can remove some processes from the design trajectory. Any design elements in M associated with those processes need to be removed, and the problem hierarchy needs to be updated. To retrace the trajectory to design process $d = (p, s, t)$, update the sets:

$$\begin{aligned} M &= M \setminus (S \cap \mathcal{Y}^\infty(s)), \\ S &= S \setminus \mathcal{Y}^\infty(s), \\ P &= P \setminus \mathcal{Y}^\infty(s), \\ D &= \{(p', s', t') \in D \mid s' \in S\} \text{ and} \\ Q &= \{(s', p') \in Q \mid p' \in P\}. \end{aligned}$$

where the backslash symbol (\setminus) is the set difference operator, i.e., $A \setminus B = \{a \in A \mid a \notin B\}$. Return to the respective step.

4. Function-behavior-structure

The above theory can in principle use any network model M . We will now frame the theory in a specific modeling paradigm, popularized in design science: FBS. So far we have been using the terms *problem*, *solution*, *design element* and *design process* in a rather general sense. In this section, we will make these terms specific for the FBS paradigm.

Our ideas are based on the situated FBS framework by Gero & Kannengiesser (2004). This framework contains ten specific classes of design elements and twenty design processes between those elements. While this level of detail provides an insightful contribution to design science, a simplified interpretation will be sufficient for the scope of our work. The original situated FBS framework is visualized in Figure 4.

Our simplified interpretation disregards the requirements as separate design elements. Furthermore, we combine the design elements from different contexts

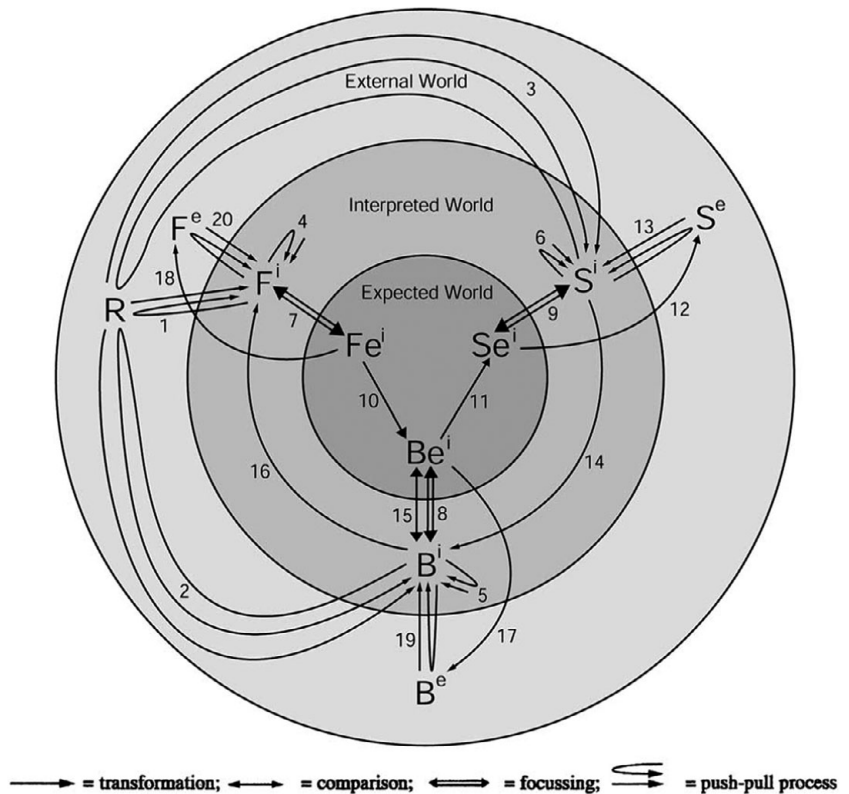


Figure 4. The situated FBS framework classifies design elements as requirements, functions, structures and behaviors. Subsets of these elements play a particular role in different contexts: the external world, the interpreted world and the expected world. Arrows between elements represent twenty classes of a design process (Gero & Kannengiesser 2004).

into functions (F^e, F^i, Fe^i), structures (S^e, S^i, Se^i) and behaviors (B^e, B^i, Be^i). This leads us to a simpler model of design, where there are only three possible processes: *formulation* is the transition from behavior to function, process 16 in Figure 4; *synthesis* is the transition from function to structure, processes 10 and 11 in Figure 4; and *analysis* is the transition from structure to behavior, process 14 in Figure 4.

Our theory frames each of these processes as a mapping from problem to solution. Most designers start with formulating an intent to influence the behavior of a given system. The formulation process d_f therefore takes a behavioral problem p_f as an input and provides a functional solution s_f as an output. Once a functional specification is established, designers generate the components that will perform functionality. This synthesis process d_s takes a functional problem p_s as an input and provides a structural solution s_s as an output. Finally, designers analyze the generated parts to determine their behavior in the working environment. The analysis process d_a therefore takes a structural problem p_a as an input and provides a behavioral solution s_a as an output. Our framing of the situated FBS framework has led to a cycle of problem-solving, visualized in Figure 5. In the upcoming demonstration section, we will use the colors blue to indicate function, orange to indicate structure and green to indicate behavior.

Problems cannot and should not always be avoided: The outcome of a formulation will always need to be synthesized, and the outcome of a synthesis will always need to be analyzed. Hence, we would classify synthesis and analysis problems as *defined* problems. However, the problems we would want to avoid are those that manifest when an analysis discovers undesired behavior. Those *discovered* problems would need to be addressed in another formulation-synthesis-analysis cycle. This cycle only ends after an analysis shows no new design problems.

In the remainder of this section, we will introduce a network representation of FBS design elements, i.e., nodes and edges of M . Then we propose which of these elements we use to describe which class of problem and solution. Finally, we explain how to visualize an FBS network in a product DSM.

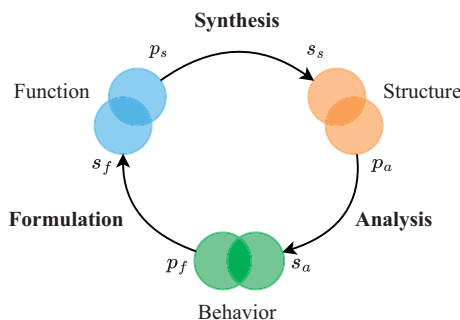


Figure 5. Three FBS problem-solving processes form an iterative design cycle. The overlapping input and output of two processes signifies a causal problem manifestation.

4.1. Network representation

We derive from the situated FBS framework a network model with three classes of nodes and six classes of edges. Figure 6 shows these design elements. We will refer to sets of nodes by single letters and to sets of edges by double letters.

Nodes in our FBS model represent functions F , structures C and behaviors B . We use the letter C for structures, in order to avoid confusion with the set of solutions S . Functions are the tasks that the design should perform, structures are its physical components and behaviors are the physical phenomena it exhibits. In the words of Gero & Kannengiesser (2004): Function describes what a design is for, structure describes what it is and behavior describes what it does.

Edges between nodes of the same class are either functional FF , structural CC or behavioral BB dependencies. A functional dependency is directed, specifying that one function requires another function; a structural dependency specifies the common geometrical features of two objects; and a behavioral dependency specifies the coupling between physical phenomena. Structural and behavioral dependencies are often modeled as undirected edges.

This leaves us with three mappings between nodes of different classes. The mappings FC define which function is performed by which structure, and the mappings CB define which structure exhibits which behavior. Finally, the mappings FB define which function is intended to influence which behavior. We consider the latter as a functional dependency between the behavior of an existing component and the function of a to-be-designed component. For example, the function of a new cooling system (e.g., ‘extract heat’) is to influence the behavior of an existing camera (e.g., ‘thermodynamics’).

As such, design model M is the union of the sets:

$$M = \cup \{F, B, C, FF, BB, CC, FB, FC, CB\}. \tag{15}$$

Each node is a description of function, behavior or structure that can in itself contain various design statements on what is desired or what is expected. As such, functional, behavioral or structural requirements can be part of any of these respective nodes.

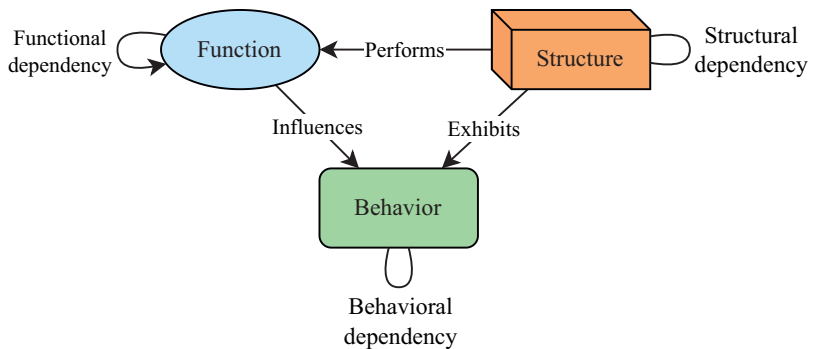


Figure 6. An FBS network model contains three classes of nodes and six classes of edges.

Table 1. Employing the situated FBS framework (Gero & Kannengiesser 2004) in our problem-solving theory has led to a network model with nodes F , B and C , and edges FF , BB , CC , FB , FC and BC . Design processes 10, 11, 14 and 16 are interpreted as three problem-solving processes and defined in terms of the network model

	Situated FBS	Problem-solving
Model	Function F^e, F^i, Fe^i	Function F, FF
	Behavior B^e, B^i, Be^i	Behavior B, BB
	Structure S^e, S^i, Se^i	Structure C, CC
	Requirements R	–
	–	Mappings FB, FC, CB
Process	Process 16	Formulation $d_f(p_f, s_f, t)$, with problem $p_f(B, BB)$ and solution $s_f(F, FF, FB)$
	Processes 10 and 11	Synthesis $d_s(p_s, s_s, t)$, with problem $p_s(F, FF, FB)$ and solution $s_s(C, CC, FC)$
	Process 14	Analysis $d_a(p_a, s_a, t)$, with problem $p_a(C, CC)$ and solution $s_a(B, BB, CB)$

4.2. Problems and solutions

We can now allocate the nodes and edges of our network model to the problems and solutions of our problem-solving cycle:

- The formulation problem p_f is about expressing the behavior B of a contextual system that needs to be changed or improved. The undesired behavior may also arise from interactions between behaviors, BB .
- The formulation solution s_f is defined in terms of new functions F that need to be introduced, the functional dependencies FF and FB indicating which behavior of the contextual system is to be changed by the new functions.
- The synthesis problem p_s is expressed in exactly the terms of a formulation solution: the desired new functionalities F and dependencies FF and FB . These three sets of design elements need to be implemented by structural features.
- The synthesis solution s_s consists of the newly generated components C , the newly introduced structural dependencies CC and the mapping between functions and components FC .
- The analysis problem p_a is to derive the behavior of a set of components C and component dependencies CC .
- The analysis solution s_a consists of the discovered behavior B and behavioral dependencies BB of the system in design, as well as the attribution of behavior to components CB .

Table 1 summarizes the analogy between the situated FBS framework and our interpretation in problem-solving.

4.3. Visualization

We have already introduced the general problem hierarchy in Figure 3 to support designers in their decision-making process. In the case of FBS modeling, the

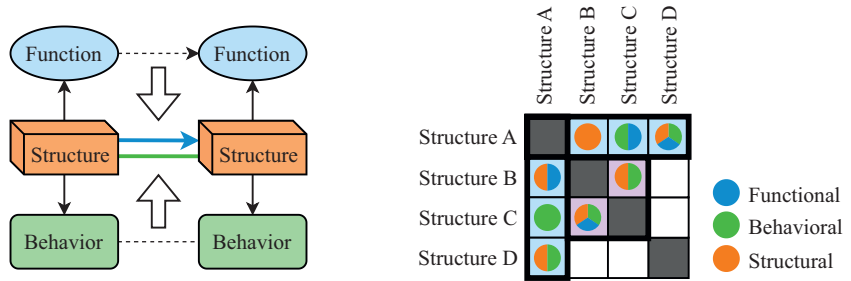


Figure 7. Left: Behavioral and functional dependencies are projected to their respective structures. Right: Proposed DSM to visualize the functional, behavioral and structural interfaces between structures. The DSM visualizes that A is a highly integrative component and that B and C form a cohesive module.

problems and solutions of the hierarchy will represent either formulation, synthesis or analysis.

Additionally, designers need to inspect their FBS model at different times and for alternative decisions. There are many possibilities to visualize such a model, but we propose to use a product DSM (Eppinger & Browning 2012).

The product DSM represents a system as the arrangement of components and their interfaces. We project functional, structural and behavioral dependencies from our FBS model onto this DSM. The leading elements on the axes of the DSM are structural elements *C*. Structural interfaces *CC* connect two structural elements and therefore appear as symmetric off-diagonal entries in the DSM.

Functional and behavioral dependencies *FF*, *FB* and *BB* do not directly connect to structural elements. They do, however, connect indirectly.

If two components (c_1 and c_2) are each related to a function ($c_1 \rightarrow f_1$ and $c_2 \rightarrow f_2$), and those functions have a mutual dependency ($f_1 \rightarrow f_2$), then we can presume that there is a functional dependency between the two components ($c_1 \rightarrow c_2$). The presumed directed dependency can then be visualized in the product DSM. Behavioral dependencies are derived in the same way but yield an undirected dependency in the DSM. Figure 7 visualizes this process.

What practical advice would our method give to designers that use the FBS paradigm? The problem hierarchy will show a recurring sequence of formulation, synthesis and analysis problems. This should motivate designers to avoid downstream problems. Revising a synthesis process could lead to geometrical reshaping or even the use of another technology. Reanalyzing a structural system could lead to a more accurate understanding of a behavioral problem. Finally, reformulating function to change a problem-solving intent might allow other solution architectures.

In this section, we have specialized our general theory for the FBS modeling paradigm. Next we demonstrate how the proposed method can be used in a fusion diagnostics-related design problem.

5. Demonstration

The Visible Spectroscopy Reference System (VSRS) is one of the diagnostic subsystems to be integrated in ITER. The VSRS is an optical diagnostic system

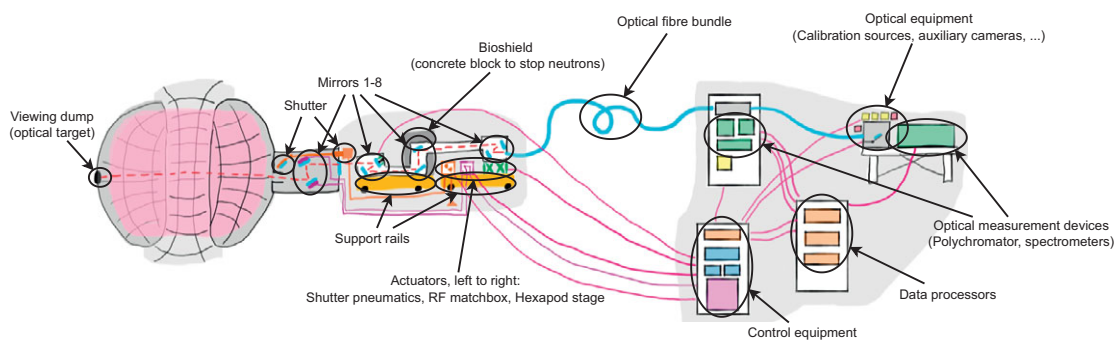


Figure 8. The Visible Spectroscopy Reference System (VSRS) is an optical measurement system in ITER. From left to right: The toroidal fusion chamber emits a pink light. That light is relayed through a sequence of eight mirrors. An optical fibre bundle transports the light to a separate building with normal environmental conditions. The light is analyzed by a polychromator and multiple spectrometers and processed by electronic equipment. The remaining components fulfill auxiliary functionalities, such as controlling a shutter, cleaning the mirrors in-vacuum and aligning and calibrating optics.

that has the role of collecting light emitted by the high-temperature plasma where the fusion process occurs and conducting real-time spectroscopic measurements. These measurements then provide data on the state of the plasma that can be used for machine protection and plasma control. Such functionalities are indispensable for operating ITER.

The VSRS generally consists of three kinds of components that implement a range of technologies: optical elements, such as mirrors, fibers and windows; measurement devices including a polychromator and multiple spectrometers; and electronic devices comprising data processors, analog and digital controllers and network equipment. Figure 8 gives a simple sketch of the VSRS.

In the following sections, we treat two problems. First, with a rather minimal view of the VSRS we show how alternative decisions lead to alternative problems and alternative design models. Second, we increase the granularity of the models. This allows us to focus on the development of the system over time, and we can identify those decisions that have most contributed to its complexity.

5.1. A simple model

This simple model of the VSRS design captures a very typical problem in low-maturity systems development. It revolves around a single critical synthesis decision: whether to use a glass fiber or a metallic mirror to transport light. At this point in time, the designer lacks detailed knowledge about any downstream issues that may occur but has to make a preliminary decision nevertheless. If in the future a problem is discovered, the designer quickly needs to assess the impact of revising the earlier decision. How can our method help the designer?

Let us first define an initial model M_0 from which to explore our alternative design trajectories. Suppose that the designer has selected the metallic mirror: case A. We break down the development process into the six subsequent formulation, synthesis and analysis processes

$$D_A = \left\{ \begin{array}{cc} d_f(p_1, s_1, 1) & d_f(p_4, s_4, 4) \\ d_s(p_2, s_2, 2) & d_s(p_5, s_5, 5) \\ d_a(p_3, s_3, 3) & d_a(p_6, s_6, 6) \end{array} \right\} \quad (16)$$

where $d(p, s, t)$ defines process d in terms of a design problem p , a design solution s and the time t when the process occurred. The elements of M_0 and the processes and solutions in D_A are visualized in Figure 9.

These processes represent two FBS cycles, as shown in Figure 5. We can use Equation (4) to derive problem causality. This reveals a linear problem hierarchy without branches. We refer to this problem hierarchy as the nominal solution path, shown in Figure 10.

After this series of decisions, Equation (3) dictates that the FBS model is the union of all the nodes and edges shown in Figure 9. Equation (9) quantifies the complexity of the system as $\zeta = 277$. At this point, the designer is discontent, particularly because of the burden of developing an auxiliary cooling system. This was not anticipated.

Seeing the importance of their earlier decisions, the designer is triggered to think of an alternative solution. A glass fiber bundle is a working principle that can transport light and could therefore be used instead of a metallic mirror. This decision is part of the synthesis process $p_2 \rightarrow s_2$. An alternative path opens up at p_2 : case B. The designer explores this path through the processes

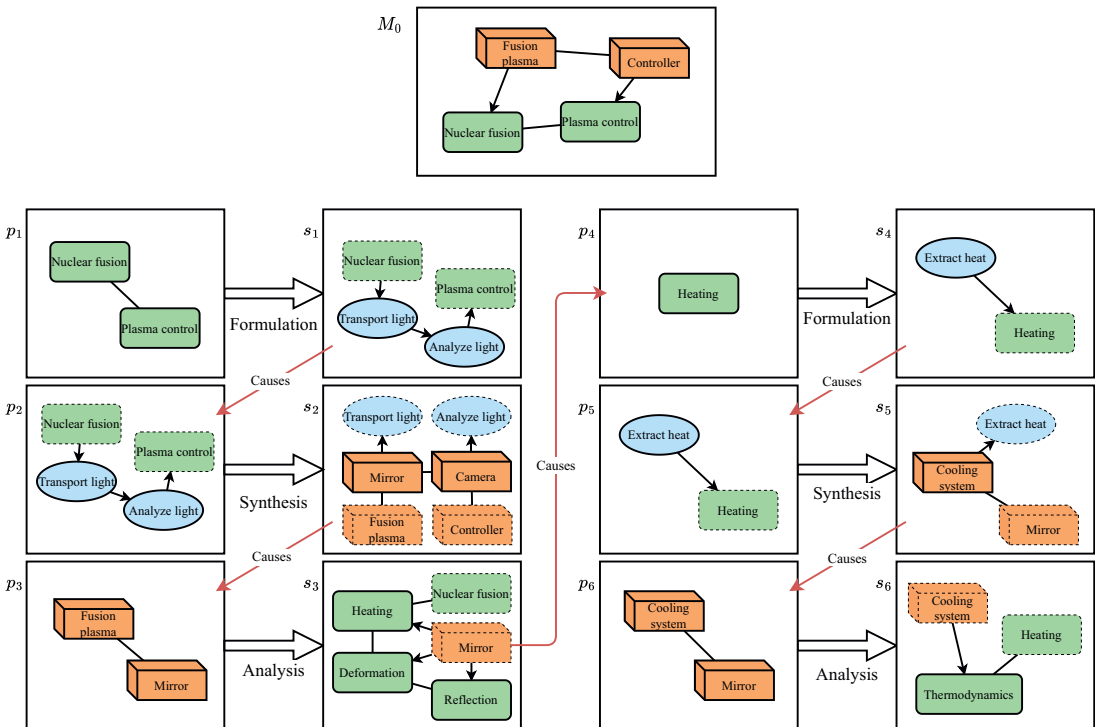


Figure 9. Six problem-solving processes have led to the nominal VSRS design. We discover five causality relations that indicate a linear decision-making sequence.

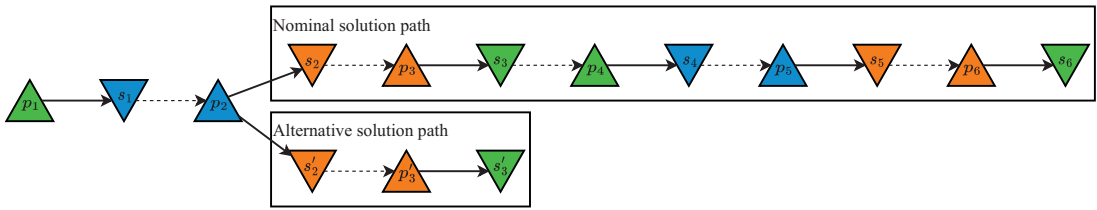


Figure 10. The problem hierarchy shows relations between functional (blue), behavioral (green) and structural (orange) problems and solutions. After a nominal development path, a shorter solution path is discovered by reconsidering the solution to p_2 : to use a fiber bundle instead of a metallic mirror.

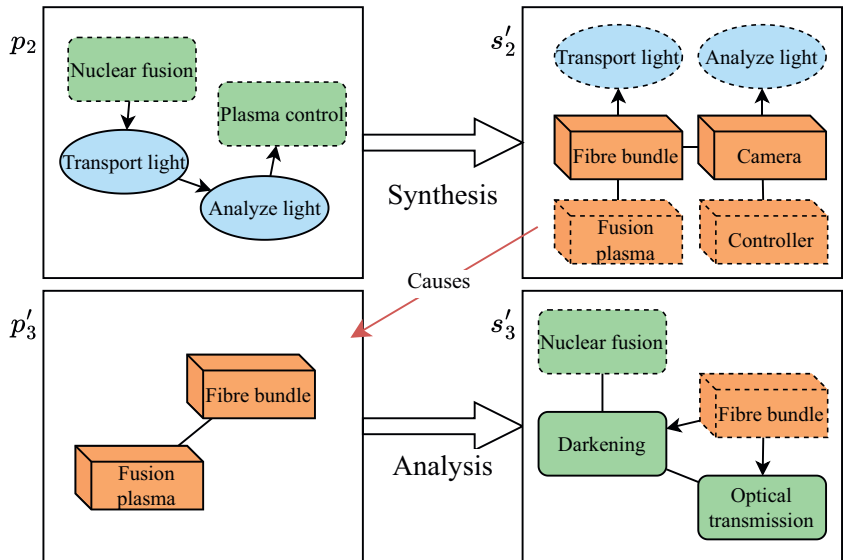


Figure 11. The problem-solving processes of an alternative development path. Using a fiber bundle instead of a metallic mirror leads to a shorter development and a simpler architecture.

$$D_B = \left\{ \begin{array}{l} d_f(p_1, s_1, 1) \\ d_s(p_2, s'_2, 2) \\ d_a(p'_3, s'_3, 3) \end{array} \right\}. \quad (17)$$

Figure 11 defines the model elements of this alternative solution path.

The new mapping $p_2 \rightarrow s'_2$ reflects the decision to implement a fiber bundle instead of a mirror to transport light. Analyzing that system ($p'_3 \rightarrow s'_3$) shows new behavior regarding optical transmission and darkening. In contrast to the mirror, the fiber bundle is not sensitive to thermal displacements that affect its optical behavior. However, glass fibers tend to darken over time when placed in a radioactive environment. This may pose a problem in the future. This example clearly demonstrates how the follow-up complexity of a cooling system can be limited by *avoiding* an identified problem.

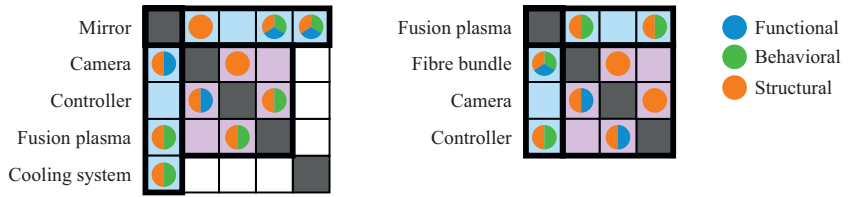


Figure 12. Two DSMs represent the outcome of two solution paths with architectural design alternatives: one using a mirror (left) and one using a fiber bundle (right).

The problem hierarchy is now significantly shorter, see Figure 10. The DSM representation of both alternative solution paths in Figure 12 shows that the cooling system is not anymore in the design. Also, the complexity of this design is considerably less than before: $\zeta = 127$.

There may still be unidentified problems ahead, but for now the designer has more confidence in the fiber bundle solution s_2' . The rejected mirror solution s_2 can still play an important role in design justification. In large development projects, multiple alternative solutions are actually explored in parallel. Our approach can play a supporting role in such efforts.

We have visualized the chain of decisions that has led to the current design, realized the potential of revising an early decision, systematically explored an alternative solution and automatically generated its corresponding design model. Now let's focus on the development of complexity over time.

5.2. A larger model

This demonstration revolves around a much larger FBS model, comprising 71 nodes (20 functional, 18 structural and 33 behavioral) and 134 edges. The model was set up after the conceptual design phase through informal interviews with a system expert. We framed the major design decisions up to that point in terms of formulation, synthesis and analysis processes.

Table A.2 in the Appendix shows the 21 processes that were established, ranked in order of occurrence in the development process. Thus each $t = 1, 2, \dots$ represents a time step when both a problem was identified and a solution was generated. Going through each individual process, the expert explained in detail the objectives and risks of the problem that was identified, and the contents of the solution that was provided. We then defined each problem and solution in FBS design elements. The problems are defined in the Appendix, in Table A.3, and the solutions in Table A.4. The initial model is given in Table A.1, and Table A.5 lists all the nodes in the model by name.

In the following sections, we present the results obtained by our method. First, we show the identified problem hierarchy, then we discuss the time-evolution of complexity and we finish this section by generating intermediate design models.

5.2.1. Problem hierarchy

The nodes of the problem hierarchy are given by problems P and solutions S , both of which are specified in Tables A.3 and A.4. The edges of the problem hierarchy consist of problem-solving dependencies $P \rightarrow S$ and causal dependencies $S \rightarrow P$.

Problem-solving dependencies are also specified by the designer (Table A.2), so we only need to derive the set of causal dependencies $S \rightarrow P$ to finalize the problem hierarchy. Equation (4) identifies 20 of these dependencies. Figure 12 shows the resulting hierarchy.

The hierarchy ranks design decisions from essential (top) to supportive (bottom). It shows us three levels of FBS processes, i.e., subsequent formulation, synthesis and analysis. The level 1 design process brings us from p_1 to s_3 and captures the basic design decisions to solve the root problem. The level 2 processes go from p_4 , p_7 and p_{10} to s_6 , s_9 , s_{12} and s_{15} , dealing with the consequences of our initial design. Finally, level 3 processes deal with leftover problems p_{15} and p_{18} . The visualization will support designers in identifying high-level opportunities for low-level problem avoidance.

5.2.2. Complexity

One of our primary objectives was to monitor the evolution of complexity during the VSRS development process. Refer to Figure 13. We do this by evaluating Equation (9) for every t in Table A.2. At every time step, there is a new solution that increases the complexity (see also Figure 2). The left plot of Figure 14 shows the resulting development of complexity over time. We see a steady increase in complexity as more and more design decisions are made.

A secondary objective is to identify the contribution of individual processes to complexity. In the right plot of Figure 14, we present the development of the global complexity impact I_G (Equation 14) of three solutions: s_1 , s_7 and s_{10} . You will find that these processes are in characteristic places of the problem hierarchy, Figure 13. The lines represent how the I_G of each process was evaluated at different points in time.

The evolution of the global complexity impact I_G of solution s_1 is represented by the blue line. Solution s_1 is the outcome of the first design process and therefore appears at the top of the problem hierarchy. The hierarchy shows that all subsequent problems (in)directly manifest themselves from this outcome. The complexity of every other solution is accounted for in the I_G of solution s_1 , which is why the curve closely follows the trend of the overall complexity. As these solutions are added to the model, the plot shows that solution s_1 has a bigger and bigger contribution to complexity.

The global complexity impact of solution s_{10} as a function of time is represented by the red line. We see that this solution added some elements at $t = 10$ but caused only a single problem. Because solution s_{10} is not responsible for any of the complexity added at $t > 12$, the line flattens. Note that this can be verified by the position in the problem hierarchy.

Finally, the I_G of solution s_7 is represented by the green line. This solution follows a similar trend to solution s_{10} : The curve flattens after some initial complexity, indicating closure of one branch of development. But then, unexpectedly, problem p_{14} was identified. A second increase in I_G shows that the decisions made in solution s_7 lead to more complexity than initially thought.

We conclude from these graphs that it is solution s_1 , obviously, that has the highest impact on the system. This is in line with claims that costs, schedule and technical performance of engineering projects are mostly determined by early-

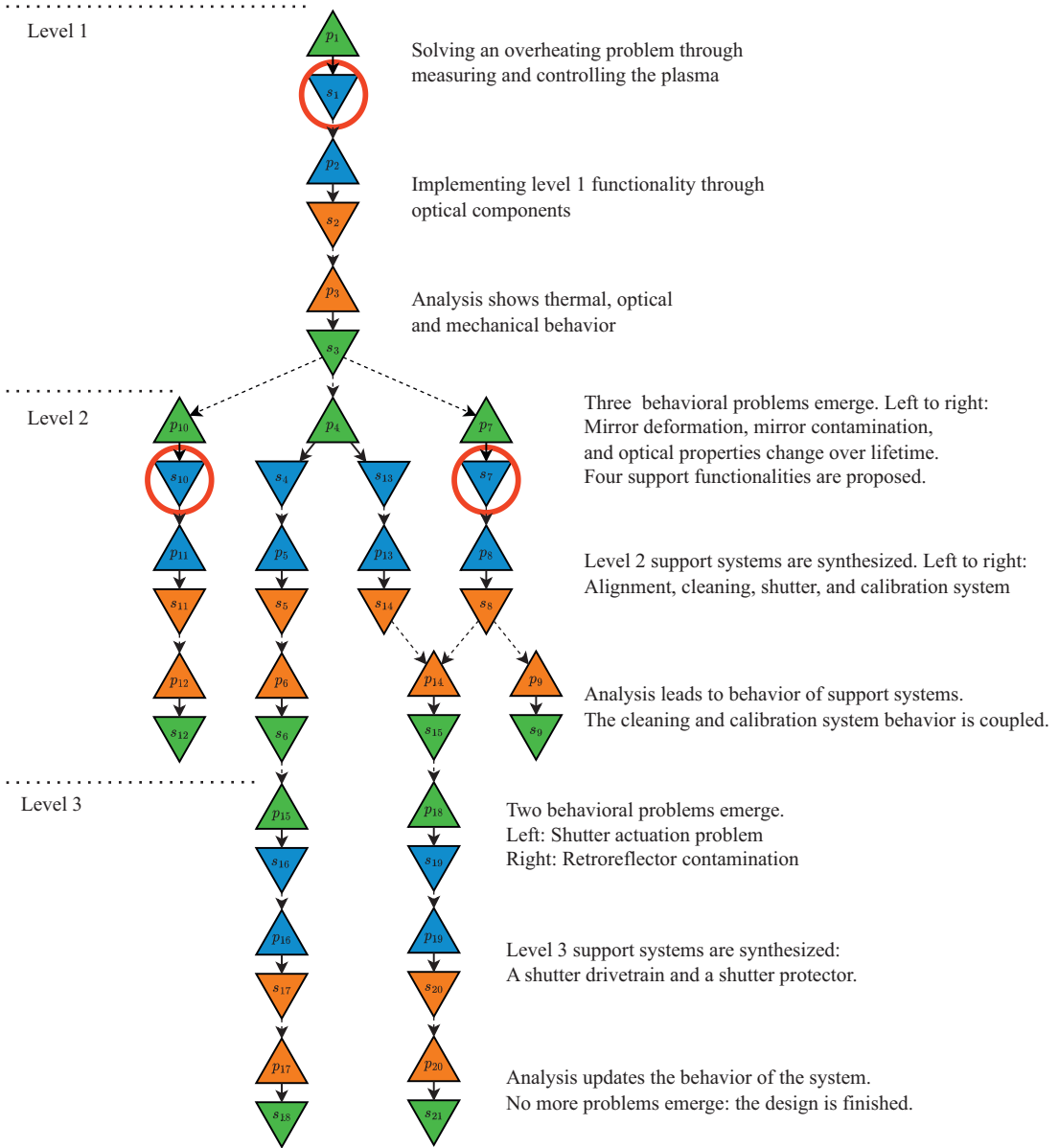


Figure 13. Problem hierarchy of the VSRs design process. Green, blue and orange nodes, respectively, represent behavioral, functional and structural problems and solutions. The problems and solutions are numbered in order of appearance in the design process, as listed in Table A.2. In the following section, the three highlighted solutions will be analyzed in greater depth.

stage decision-making. The course of solution s_{10} is an indication of good decisions that do not impact complexity in later stages. However, the second increase in I_G of solution s_7 should serve as a warning: This process has caused an unforeseen problem.

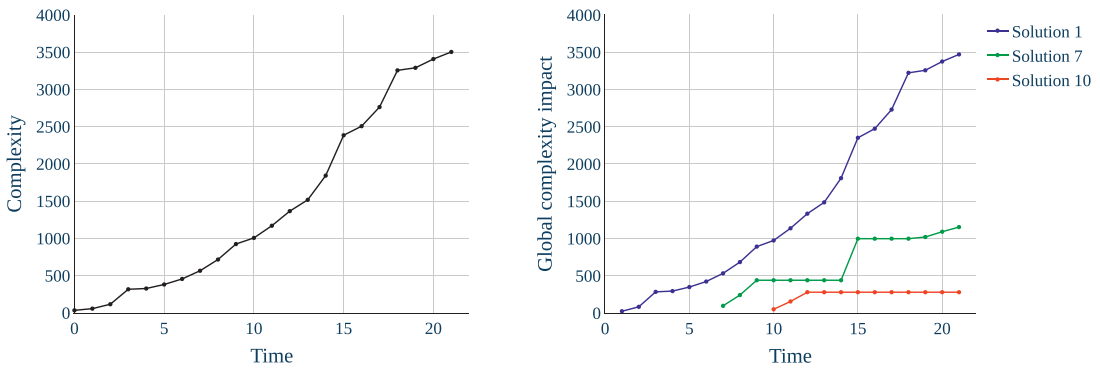


Figure 14. Left: Growth of system complexity over the development process, as computed from Equation (9). Right: Contribution of three characteristic processes, evaluated over time by the global complexity impact (Equation 14). The close similarity between the black line in the left plot and the blue line in the right plot indicates that solution 1 has been highly influential in the evolution of complexity. The difference between these lines is the complexity that can be attributed to the initial model M_0 .

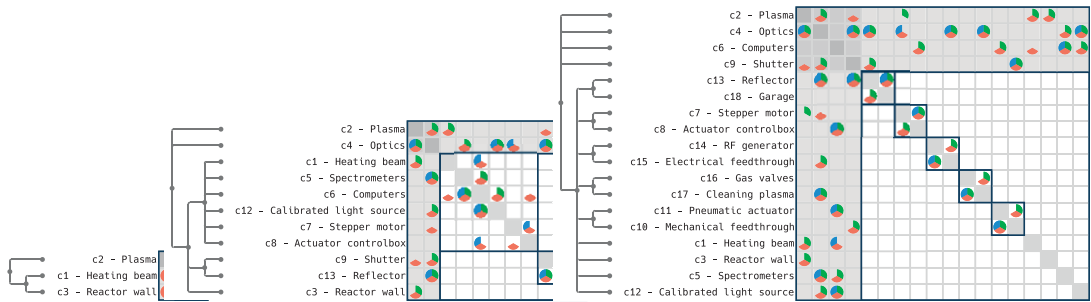


Figure 15. Product DSMs representing the VSRS at the beginning ($t = 0$), middle ($t = 11$) and end ($t = 21$) of the development process.

5.2.3. Model expansion

We know now how the complexity of the VSRS has developed over time. Inspecting the design model at different stages will give us complementary insight into how its architecture grows. We have made three product DSM snapshots at the beginning, middle and end of the development process. Figure 15 compares these DSMs.

We see that the initial DSM contains no functional interfaces. It represents the initial design problem as a behavioral one, before any design intent is formulated. Over time, more components and interfaces contribute to complexity.

The DSMs also show that the modularity of the design changes over time. Consider the differences between the matrix in the middle and the one on the right. The computer c_6 was initially placed in a module with five other components. However, at a later stage the computer became more centrally connected and was therefore moved to the bus. This meant that the heating beam c_1 , the spectrometers c_5 and the light source c_{12} could be removed from the module as well and are now

more independent. Finally, the rightmost matrix introduces more components that form new modules.

Architectural changes in the technical system often have a large impact on the developing organization and its strategies. Analyzing architectural patterns through time provides a systematic means to adapt, for example, by redistributing responsibilities and redefining organizational structures.

6. Closing remarks

Designers can only solve problems effectively if provided with the right tools and techniques. In this paper, we took aim at the case of arising problems. These often unexpected problems appear in late design stages of solution search as a consequence of earlier decisions. We observe that solving such problems can lead to undesired system complexity and, unfortunately, more problems. Our objective is to show designers the cause–effect relations in their decision-making process. This will motivate them to try to *avoid* problems and complexity by reconsidering a prior design decision.

We have presented a theoretical basis for the interplay between problems and solutions. We consider both problems and solutions as elements of a design model, and formalize two causal relations: First, as a conscious design process, a *solution* adds elements to the model in order to solve a *problem*. Second, a *problem* manifests itself due to a *solution* if that solution has added an element that also represents a problem. We visualize these relations in a problem hierarchy. We furthermore introduce two impact factors that quantify the contribution of each decision to the complexity of the design.

Our theory of problem-solving is then merged with the FBS paradigm. The result is a systematic problem-solving method that specializes design processes and problem causality: formulation, synthesis and analysis are the specific design processes that connect functional, behavioral and structural problems and solutions. These elements are visualized in a product DSM. Instances of this DSM can be automatically generated to explore the time-evolution of alternative solution paths.

We have illustrated our design method in two cases of the VSRS, an optical measurement system for nuclear fusion reactors. A simple example with six design steps shows how two alternative solution paths lead to different manifesting problems and different system complexity. The following demonstration contains 21 design steps and focuses on the evolution of the system through time. We are able to monitor the growing complexity throughout the design process and can assess the impact of each individual decision on the overall complexity.

Nuclear fusion reactors are already complex enough, while the search for a viable implementation of this technology is still ongoing. Similar complexity cascades arise also in many other first-of-a-kind development projects in big science and engineering. Let us try to avoid unnecessary problems and manage their complexity.

Acknowledgements

We express our sincere gratitude to the ITER Port Plugs and Diagnostics Department, for providing access to the Interface Database. We would also like to thank

the anonymous reviewers of this journal for their constructive feedback, which has led to significant improvements to this paper.

Nomenclature

M	Design model consisting of design elements
E	Entities of a design model
R	Relations of a design model
ξ	Structural complexity, a scalar attribute of a design model
D	Set of design processes, each process mapping a problem into a solution
P	Set of design problems
S	Set of design solutions
H	Problem hierarchy, representing the causal structure of the design process by problems and solutions
$\mathcal{X}(h)$	Set of input nodes to node $h \in H$
$\mathcal{X}^\infty(h)$	Set of nodes from which through hierarchy H node $h \in H$ can be reached
$\mathcal{Y}(h)$	Set of output nodes from node $h \in H$
$\mathcal{Y}^\infty(h)$	Set of nodes that can be reached through hierarchy H departing from node $h \in H$
$I_L(d)$	Local complexity impact due to design process $d \in D$
$I_G(d)$	Global complexity impact due to design process $d \in D$
F	Functions of a system
B	Behaviors of a system
C	Structural components of a system

Disclaimer

The views and opinions expressed herein do not necessarily reflect those of the ITER Organization.

References

- Adams, G. S., Converse, B. A., Hales, A. H., & Klotz, L. E. (2021). People systematically overlook subtractive changes. *Nature* **592**(7853), 258–261; doi:[10.1038/s41586-021-03380-y](https://doi.org/10.1038/s41586-021-03380-y).
- Bernaert, T., Etman, P., De Bock, M., Classen, I., & De Baar, M. (2021). Tracing the emergence of design problems and their impacts on the complexity of engineering solutions. *Proceedings of the Design Society* **1**, 3229–3238. doi:[10.1017/pds.2021.584](https://doi.org/10.1017/pds.2021.584).
- Bernaert, T., Etman, P., De Bock, M., De Baar, M., & Classen, I. (2022). Challenges of big-science: A matrix-based interface model to manage technical integration risks in multi-organizational engineering projects. In *DS 121: Proceedings of the 24th International DSM Conference (DSM 2022)*, Eindhoven, The Netherlands, October, 11–13, 2022, pp. 1–10. The Design Society; doi:[10.35199/dsm2022.01](https://doi.org/10.35199/dsm2022.01).
- Browning, T. R. (2016). Design structure matrix extensions and innovations: A survey and new opportunities. *IEEE Transactions on Engineering Management*, **63**(1), 27–52; doi:[10.1109/TEM.2015.2491283](https://doi.org/10.1109/TEM.2015.2491283).

- Brunetti, G., & Golob, B.** (2000). A feature-based approach towards an integrated product model including conceptual design information. *Computer-Aided Design*, *32*(14), 877–887; doi:[10.1016/S0010-4485\(00\)00076-2](https://doi.org/10.1016/S0010-4485(00)00076-2).
- Bussemaker, J. H., Ciampa, P. D., & Nagel, B.** (2020). *System architecture design space exploration: An approach to modeling and optimization*. In AVIAA Aviation 2020 Forum; doi:[10.2514/6.2020-3172](https://doi.org/10.2514/6.2020-3172).
- Cinque, M., De Tommasi, G., De Vries, P. C., Fucci, F., Zabeo, L., Ambrosino, G., Bremond, S., Gomez, I., Karkinsky, D., Mattei, M., Nouailletas, R., Pironti, A., Rimini, F. G., Snipes, J. A., Treutterer, W., & Walker, M. L.** (2020). Management of the ITER PCS design using a system-engineering approach. *IEEE Transactions on Plasma Science* *48*(6), 1768–1778; doi:[10.1109/TPS.2019.2945715](https://doi.org/10.1109/TPS.2019.2945715).
- Costley, A. E., Sugie, T., Vayakis, G., & Walker, C. I.** (2005). Technological challenges of ITER diagnostics. *Fusion Engineering and Design* *74*, 109–119; doi:[10.1016/j.fusengdes.2005.08.026](https://doi.org/10.1016/j.fusengdes.2005.08.026).
- Daly, S. R., McKilligan, S., Studer, J. A., Murray, J. K., & Seifert, C. M.** (2018). Innovative solutions through innovated problems. *International Journal of Engineering Education* *34*(2(B)), 695–707.
- Di Gironimo, G., Lanzotti, A., Marzullo, D., Esposito, G., Carfora, D., & Siuko, M.** (2015). Iterative and participative axiomatic design process in complex mechanical assemblies: Case study on fusion engineering. *International Journal on Interactive Design and Manufacturing (IJIDeM)* *9*(4), 325–338; doi:[10.1007/s12008-015-0270-7](https://doi.org/10.1007/s12008-015-0270-7).
- Dickerson, C. E., & Mavris, D.** (2013). A brief history of models and model based systems engineering and the case for relational orientation. *IEEE Systems Journal* *7*(4), 581–592; doi:[10.1109/JSYST.2013.2253034](https://doi.org/10.1109/JSYST.2013.2253034).
- Dmitriev, A. M., Babinov, N. A., Bazhenov, A. N., Bukreev, I. M., Elets, D. I., Filimonov, V. V., Koval, A. N., Kueskiev, G. S., Litvinov, A. E., Mikhin, E. E., Razdobarin, A. G., Samsonov, D. S., Senitchenkov, V. A., Solovei, V. A., Terechenko, I. B., Tolstyakov, S. Y., Varshavchik, L. A., Chernakov, P. V., Chernakov, A. P., Chernakov, A. P., Tugarionov, S. N., Shigin, P. A., Leipold, F., Reichle, R., Walsh, M. & Pflug, A.** (2019). RF plasma cleaning of water-cooled mirror equipped with notch filter based on shorted $\lambda/4$ line. *Fusion Engineering and Design* *146*, 1390–1393; doi:[10.1016/j.fusengdes.2019.02.090](https://doi.org/10.1016/j.fusengdes.2019.02.090).
- Dongiovanni, D. N., Esposito, B., Marocco, D., & Marzullo, D.** (2018). Design space exploration for architecture selection: Radial Neutron Camera nuclear fusion diagnostic study case. *Fusion Engineering and Design* *137*, 378–389; doi:[10.1016/j.fusengdes.2018.10.020](https://doi.org/10.1016/j.fusengdes.2018.10.020).
- Dorst, K.** (2019). Co-evolution and emergence in design. *Design Studies* *65*, 60–77; doi:[10.1016/j.destud.2019.10.005](https://doi.org/10.1016/j.destud.2019.10.005).
- Drave, I., Rumpe, B., Wortmann, A., Berroth, J., Hoepfner, G., Jacobs, G., Spuetz, K., Zerwas, T., Guist, C., & Kohl, J.** (2020). Modeling mechanical functional architectures in SysML. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, pp. 79–89. IEEE; doi:[10.1145/3365438.3410938](https://doi.org/10.1145/3365438.3410938).
- Eckert, C., Clarkson, P. J., & Zanker, W.** (2004). Change and customisation in complex engineering domains. *Research in Engineering Design* *15*(1), 1–21; doi:[10.1007/s00163-003-0031-7](https://doi.org/10.1007/s00163-003-0031-7).
- Eppinger, S. D., & Browning, T. R.** (2012). *Design Structure Matrix Methods and Applications* (1st ed.). MIT Press.
- Estefan, J. A.** (2007). Survey of model-based systems engineering (MBSE) methodologies. *INCOSE MBSE Focus Group* *25*(8).

- Federici, G., Bachmann, C., Biel, W., Boccaccini, L., Cismondi, F., Ciattaglia, S., Coleman, M., Day, C., Diegele, E., Franke, T., Grattarola, M., Hurzmeier, H., Ibarra, A., Loving, A., Maviglia, F., Meszaros, B., Morlock, C., Rieth, M., Shannon, M., Taylor, N., Tran, M. Q., You, J. H., Wenninger, R. & Zani, L. (2016). Overview of the design approach and prioritization of R&D activities towards an EU DEMO. *Fusion Engineering and Design* **109–111**, 1464–1474; doi:[10.1016/j.fusengdes.2015.11.050](https://doi.org/10.1016/j.fusengdes.2015.11.050).
- Forsberg, K., & Mooz, H. (1991). The relationship of system engineering to the project cycle. *INCOSE International Symposium* **1**(1), 57–65; doi:[10.1002/j.2334-5837.1991.tb01484.x](https://doi.org/10.1002/j.2334-5837.1991.tb01484.x).
- Gero, J. S. (1990). Design prototypes: A knowledge representation schema for design. *AI Magazine* **11**(4), 26–36.
- Gero, J. S., & Kannengiesser, U. (2004). The situated function–behaviour–structure framework. *Design Studies* **25**(4), 373–391; doi:[10.1016/j.destud.2003.10.010](https://doi.org/10.1016/j.destud.2003.10.010).
- Grossetti, G., Brown, R., Franke, T., Gafert, J., Galliard, T., Jenkins, I., Mantel, N., Strauß, D., Tran, M. Q., & Wenninger, R. (2018). Systems engineering perspective to the integration of the heating and current drive system in the EU DEMO: Analysis of requirements and functions. *Fusion Engineering and Design* **136**, 53–57; doi:[10.1016/j.fusengdes.2017.12.023](https://doi.org/10.1016/j.fusengdes.2017.12.023).
- Hamraz, B., Caldwell, N. H. M., Wynn, D. C., & Clarkson, P. J. (2013). Requirements-based development of an improved engineering change management method. *Journal of Engineering Design* **24**(11), 765–793; doi:[10.1080/09544828.2013.834039](https://doi.org/10.1080/09544828.2013.834039).
- Johannesson, H., & Claesson, A. (2005). Systematic product platform design: A combined function-means and parametric modeling approach. *Journal of Engineering Design* **16**(1), 25–43; doi:[10.1080/09544820512331325247](https://doi.org/10.1080/09544820512331325247).
- Jonassen, D., Strobel, J., & Lee, C. B. (2006). Everyday problem solving in engineering: Lessons for engineering educators. *Journal of Engineering Education* **95**(2), 139–151; doi:[10.1002/j.2168-9830.2006.tb00885.x](https://doi.org/10.1002/j.2168-9830.2006.tb00885.x).
- Klema, V., & Laub, A. (1980). The singular value decomposition: Its computation and some applications. *IEEE Transactions on Automatic Control* **25**(2), 164–176; doi:[10.1109/TAC.1980.1102314](https://doi.org/10.1109/TAC.1980.1102314).
- Lanzotti, F. G., Marzullo, D., Imbriani, V., Mazzone, G., You, J.-H., & Di Gironimo, G. (2023). Requirements management in master model development: A case study in fusion engineering. In *Advances on Mechanics, Design Engineering and Manufacturing IV* (eds. S. Gerbino, A. Lanzotti, M. Martorelli, R. M. Buil, C. Rizzi, & L. Roucoules), pp. 466–478. Springer International Publishing; doi:[10.1007/978-3-031-15928-2_41](https://doi.org/10.1007/978-3-031-15928-2_41).
- Leipold, F., Reichle, R., Vorpahl, C., Mukhin, E. E., Dmitriev, A. M., Razdobarin, A. G., Samsonov, D. S., Marot, L., Moser, L., Steiner, R., & Meyer, E. (2016). Cleaning of first mirrors in ITER by means of radio frequency discharges. *Review of Scientific Instruments* **87**(11), 11D439; doi:[10.1063/1.4962055](https://doi.org/10.1063/1.4962055).
- Lin, Y. (1999). *General Systems Theory: A Mathematical Approach* (Softcover reprint of the hardcover 1st edition). Springer Science + Business Media, LLC; doi:[10.1007/978-0-306-46962-6](https://doi.org/10.1007/978-0-306-46962-6).
- Madni, A. M., & Sievers, M. (2018). Model-based systems engineering: Motivation, current status, and research opportunities. *Systems Engineering* **21**(3), 172–190; doi:[10.1002/sys.21438](https://doi.org/10.1002/sys.21438).
- Martinez, T., Škec, S., Lukačević, F., & Štorga, M. (2021). Modelling proportions and sequences of operations in team design activities. *Proceedings of the Design Society 1: ICED 21*, 2187–2196; doi:[10.1017/pds.2021.480](https://doi.org/10.1017/pds.2021.480).
- Mathias, J., Eifler, T., Engelhardt, R., Kloberdanz, H., & Bohn, A. (2011). Selection of physical effects based on disturbances and robustness ratios in the early phases of robust

- design. In *DS 68-5: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, 5: Design for X/Design to X*. The Design Society.
- Marzullo, D., Bachmann, C., Coccorese, D., Di Gironimo, G., Mazzone, G., & You, J. H.** (2017). Systems engineering approach for pre-conceptual design of DEMO divertor cassette. *Fusion Engineering and Design*, **124**, 649–654; doi:[10.1016/j.fusengdes.2017.02.017](https://doi.org/10.1016/j.fusengdes.2017.02.017).
- Moscato, I., Barucca, L., Bubelis, E., Caruso, G., Ciattaglia, S., Ciurluini, C., Del Nevo, A., Di Maio, P. A., Giannetti, F., Hering, W., Lorusso, P., Martelli, E., Narcisi, V., Norrman, S., Pinna, T., Perez-Martin, S., Quartararo, A., Szogradi, M., Tarallo, A., & Vallone, E.** (2022). Tokamak cooling systems and power conversion system options. *Fusion Engineering and Design* **178**, 113093; doi:[10.1016/j.fusengdes.2022.113093](https://doi.org/10.1016/j.fusengdes.2022.113093).
- Obieke, C. C., Milisavljevic-Syed, J., & Han, J.** (2021). Data-driven creativity: Computational problem-exploring in engineering design. *Proceedings of the Design Society I: ICED 21*, 831–840; doi:[10.1017/pds.2021.83](https://doi.org/10.1017/pds.2021.83).
- Pahl, G., Beitz, W., Feldhusen, J., & Grote, K. H.** (2007). *Engineering Design: A Systematic Approach* (3rd ed.). Springer.
- Panarotto, M., Kipouros, T., Brahma, A., Isaksson, O., Strandh Tholin, O., & Clarkson, J.** (2022). Using DSMs in functionally driven explorative design experiments – An automation approach. In *DS 121: Proceedings of the 24th International DSM Conference (DSM 2022), Eindhoven, The Netherlands, October, 11–13, 2022*, pp. 68–77. The Design Society; doi:[10.35199/dsm2022.08](https://doi.org/10.35199/dsm2022.08).
- Paparistodimou, G., Duffy, A., Whitfield, R. I., Knight, P., & Robb, M.** (2020). A network science-based assessment methodology for robust modular system architectures during early conceptual design. *Journal of Engineering Design* **31**(4), 179–218; doi:[10.1080/09544828.2019.1686469](https://doi.org/10.1080/09544828.2019.1686469).
- Potts, M. W., Johnson, A., & Bullock, S.** (2020). Evaluating the complexity of engineered systems: A framework informed by a user case study. *Systems Engineering* **23**(6), 707–723. doi:[10.1002/sys.21558](https://doi.org/10.1002/sys.21558).
- Potts, M. W., Sartor, P. A., Johnson, A., & Bullock, S.** (2020). Assaying the importance of system complexity for the systems engineering community. *Systems Engineering* **23**(5), 579–596; doi:[10.1002/sys.21550](https://doi.org/10.1002/sys.21550).
- Raja, V., Kokkolaras, M., & Isaksson, O.** (2019). A simulation-assisted complexity metric for design optimization of integrated architecture aero-engine structures. *Structural and Multidisciplinary Optimization* **60**(1), 287–300; doi:[10.1007/s00158-019-02308-5](https://doi.org/10.1007/s00158-019-02308-5).
- Ramsaier, M., Stetter, R., Till, M., & Rudolph, S.** (2020). Abstract physics representation of a balanced two-wheel scooter in graph-based design languages. *Proceedings of the Design Society: DESIGN Conference* **1**, 1057–1066; doi:[10.1017/dsd.2020.32](https://doi.org/10.1017/dsd.2020.32).
- Rötzer, S., Schweigert-Recksiek, S., Thoma, D., & Zimmermann, M.** (2022). Attribute dependency graphs: Modelling cause and effect in systems design. *Design Science* **8**, e27; doi:[10.1017/dsj.2022.20](https://doi.org/10.1017/dsj.2022.20).
- Salewski, M., Meo, F., Bindslev, H., Furtula, V., Korsholm, S. B., Lauritzen, B., Leipold, F., Michelsen, P. K., Nielsen, S. K., & Nonbøl, E.** (2008). Investigation of first mirror heating for the collective Thomson scattering diagnostic in ITER. *Review of Scientific Instruments* **79**(10), 10E729; doi:[10.1063/1.2956961](https://doi.org/10.1063/1.2956961).
- Simpson, T. W., Rosen, D., Allen, J. K., & Mistree, F.** (1998). Metrics for assessing design freedom and information certainty in the early stages of design. *Journal of Mechanical Design* **120**(4), 628–635; doi:[10.1115/1.2829325](https://doi.org/10.1115/1.2829325).

- Sinha, K., & de Weck, O. L.** (2013). A network-based structural complexity metric for engineered complex systems. In *2013 IEEE International Systems Conference (SysCon)*, pp. 426–430. IEEE; doi:[10.1109/SysCon.2013.6549917](https://doi.org/10.1109/SysCon.2013.6549917).
- Stephan, U., Steinke, O., Ushakov, A., Verlaan, A., de Bock, M., Moser, L., Maniscalco, M. P., van Beekum, E., & Verhoeff, P.** (2021). RF circuit analysis for ITER visible spectroscopy reference system first mirror plasma cleaning. *Fusion Engineering and Design* **168**, 112654; doi:[10.1016/j.fusengdes.2021.112654](https://doi.org/10.1016/j.fusengdes.2021.112654).
- Suh, N. P.** (1990). *The Principles of Design*. Oxford University Press.
- Summers, J. D., & Shah, J. J.** (2010). Mechanical engineering design complexity metrics: Size, coupling, and solvability. *Journal of Mechanical Design* **132**(2), 021004; doi:[10.1115/1.4000759](https://doi.org/10.1115/1.4000759).
- Tan, J. J. Y., Otto, K. N., & Wood, K. L.** (2017). Relative impact of early versus late design decisions in systems development. *Design Science* **3**, e12; doi:[10.1017/dsj.2017.13](https://doi.org/10.1017/dsj.2017.13).
- Ulrich, K.** (1995). The role of product architecture in the manufacturing firm. *Research Policy* **24**(3), 419–440; doi:[10.1016/0048-7333\(94\)00775-3](https://doi.org/10.1016/0048-7333(94)00775-3).
- Umeda, Y., Takeda, H., Tomiyama, T., & Yoshikawa, H.** (1990). Function, behaviour, and structure. *Applications of Artificial Intelligence in Engineering* **1**, 177–193.
- Ushakov, A., Verlaan, A., Stephan, U., Steinke, O., de Bock, M., Maniscalco, M. P., & Verhoeff, P.** (2020). ITER visible spectroscopy reference system first mirror plasma cleaning in radio-frequency gas discharge – circuit design and plasma effects. *Fusion Engineering and Design* **154**, 111546; doi:[10.1016/j.fusengdes.2020.111546](https://doi.org/10.1016/j.fusengdes.2020.111546).
- Wallis, W. D.** (2012). *A Beginner's Guide to Discrete Mathematics*. Birkhäuser Boston; doi:[10.1007/978-0-8176-8286-6](https://doi.org/10.1007/978-0-8176-8286-6).
- Watson, M., Anway, R., McKinney, D., Rosser, L. A., & MacCarthy, J.** (2019). Appreciative methods applied to the assessment of complex systems. *INCOSE International Symposium* **29**(1), 448–477; doi:[10.1002/j.2334-5837.2019.00614.x](https://doi.org/10.1002/j.2334-5837.2019.00614.x).
- Wilschut, T., Etman, L. F. P., Rooda, J. E., & Vogel, J. A.** (2018). Multi-level function specification and architecture analysis using ESL: A lock renovation pilot study. In *Proceedings of the ASME 2018 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference. IDETC/CIE 2018, Quebec, Canada*. ASME; doi:[10.1115/DETC2018-85191](https://doi.org/10.1115/DETC2018-85191).
- Wolff, D., Brown, R., Curson, P., Ellis, R., Galliard, T., & Harris, M.** (2018). Early lessons from the application of systems engineering at UKAEA (May 2017). *IEEE Transactions on Plasma Science* **46**(5), 1725–1734; doi:[10.1109/TPS.2018.2819726](https://doi.org/10.1109/TPS.2018.2819726).
- Yang, Q., Yang, N., Browning, T. R., Jiang, B., & Yao, T.** (2019). Clustering product development project organization from the perspective of social network analysis. *IEEE Transactions on Engineering Management* **69**, 2482–2496; doi:[10.1109/TEM.2019.2939398](https://doi.org/10.1109/TEM.2019.2939398).
- Zhang, W., & Ma, J.** (2021). Designing in complexity: How solution conjectures inform problem exploration. *Proceedings of the Design Society* **1**: ICED 21, 1153–1162; doi:[10.1017/pds.2021.115](https://doi.org/10.1017/pds.2021.115).

Appendix

This appendix contains the information of the detailed VSRS demonstration. The design process departs from the initial model in Table A.1. Table A.2 defines the subsequent design steps as mappings between a problem and a solution. Tables A.3 and A.4 define those problems and solutions, respectively, in terms of FBS nodes and edges. Table A.5 lists these functional, structural and behavioral nodes by name.

Table A.1. The initial model of the VSRS consists only of structural and behavioral elements C and B

Nodes	Edges
c_1, c_2, c_3	$(c_1, c_2), (c_2, c_3)$
b_1, b_2, b_3, b_4	$(b_1, b_2), (b_2, b_3), (b_2, b_4), (c_1, b_1), (c_2, b_2), (c_3, b_3), (c_2, b_4), (c_2, b_5)$

Table A.2. Problem-solving processes in the development of the VSRS

t	Problem	Solution	Design activity
1	p_1	s_1	Formulation
2	p_2	s_2	Synthesis
3	p_3	s_3	Analysis
4	p_4	s_4	Formulation
5	p_5	s_5	Synthesis
6	p_6	s_6	Analysis
7	p_7	s_7	Formulation
8	p_8	s_8	Synthesis
9	p_9	s_9	Analysis
10	p_{10}	s_{10}	Formulation
11	p_{11}	s_{11}	Synthesis
12	p_{12}	s_{12}	Analysis
13	p_4	s_{13}	Formulation
14	p_{13}	s_{14}	Synthesis
15	p_{14}	s_{15}	Analysis
16	p_{15}	s_{16}	Formulation
17	p_{16}	s_{17}	Synthesis
18	p_{17}	s_{18}	Analysis
19	p_{18}	s_{19}	Formulation
20	p_{19}	s_{20}	Synthesis
21	p_{20}	s_{21}	Analysis

Table A.3. Definition of all design problems in the development of the VSRS

Problem	Nodes	Edges
p_1	b_1, b_2, b_3	$(b_1, b_2), (b_2, b_3)$
p_2	f_1, f_2, f_3	$(f_1, f_2), (f_2, f_3), (f_3, b_1), (b_4, f_1)$
p_3	c_4, c_5, c_6	$(c_1, c_6), (c_4, c_5), (c_5, c_6), (c_2, c_4)$
p_4	b_5, b_7, b_8	$(b_5, b_7), (b_8, b_7)$
p_5	f_7	(f_7, b_7)
p_6	c_9	–
p_7	b_{12}	–
p_8	$f_{11}, f_{12}, f_{13}, f_{14}$	$(f_{11}, f_{12}), (f_{12}, f_{11}), (f_{11}, f_{13}), (f_{13}, f_{11}), (f_2, f_{14}), (f_{14}, b_{12}), (f_7, f_{13})$
p_9	c_{12}, c_{13}, c_4, c_5	$(c_4, c_{13}), (c_4, c_{12}), (c_6, c_{12}), (c_5, c_6), (c_4, c_5)$
p_{10}	b_6, b_8	(b_8, b_6)
p_{11}	f_4, f_5, f_6	$(f_4, f_5), (f_5, f_6), (f_6, b_6)$
p_{12}	c_6, c_7, c_8, c_4	$(c_4, c_7), (c_7, c_8), (c_6, c_8)$
p_{13}	$f_{15}, f_{16}, f_{17}, f_{18}, f_{19}$	$(f_{15}, f_{16}), (f_{16}, f_{18}), (f_{17}, f_{19}), (f_{18}, f_{19}), (f_{19}, b_8)$
p_{14}	$c_{14}, c_{15}, c_{16}, c_{17}, c_{13}, c_4$	$(c_{14}, c_{15}), (c_{15}, c_4), (c_{17}, c_4), (c_{16}, c_{17}), (c_4, c_{13})$
p_{15}	b_{16}	–
p_{16}	f_8, f_9, f_{10}	$(f_8, f_9), (f_9, f_{10}), (f_{10}, b_{16})$
p_{17}	c_6, c_9, c_{11}, c_{10}	$(c_6, c_{11}), (c_{11}, c_{10}), (c_9, c_{10})$
p_{18}	b_{32}	–
p_{19}	f_{20}	(f_{20}, b_{32})
p_{20}	c_{18}, c_{13}	(c_{13}, c_{18})

Table A.4. Definition of all design solutions in the development of the VSRS

Solution	Nodes	Edges
s_1	f_1, f_2, f_3	$(f_1, f_2), (f_2, f_3), (f_3, b_1), (b_4, f_1)$
s_2	c_4, c_5, c_6	$(c_2, c_4), (c_4, c_5), (c_5, c_6), (c_4, f_1), (c_5, f_2), (c_6, f_3), (c_1, c_6)$
s_3	$b_8, b_9, b_6, b_5, b_7, b_{10}, b_{11}, b_{12}$	$(c_2, b_5), (b_9, b_6), (b_8, b_6), (b_8, b_7), (b_5, b_7), (b_9, b_4), (b_8, b_{11}), (b_{10}, b_{11}), (b_{10}, b_{12}), (c_4, b_9), (c_4, b_6), (c_4, b_8), (c_4, b_7), (c_5, b_{10}), (c_5, b_{11}), (c_6, b_{12})$
s_4	f_7	(f_7, b_7)
s_5	c_9	$(c_9, c_4), (c_9, f_7), (c_9, c_2)$
s_6	b_{16}, b_{22}	$(c_9, b_{16}), (c_9, b_{22}), (b_{16}, b_{22}), (b_7, b_{22})$
s_7	$f_{11}, f_{12}, f_{13}, f_{14}$	$(f_{11}, f_{12}), (f_{12}, f_{11}), (f_1, f_{13}), (f_{13}, f_1), (f_2, f_{14}), (f_{14}, b_{12}), (f_7, f_{13})$
s_8	c_{12}, c_{13}	$(c_6, f_{11}), (c_{12}, f_{12}), (c_{13}, f_{13}), (c_6, f_{14}), (c_6, c_{12}), (c_4, c_{12}), (c_4, c_{13}), (c_{13}, c_9)$
s_9	b_{23}, b_{24}, b_{25}	$(c_{12}, b_{23}), (c_{13}, b_{24}), (c_6, b_{25}), (b_{23}, b_8), (b_{23}, b_{25}), (b_8, b_{24}), (b_{10}, b_{25}), (b_{16}, b_{24})$
s_{10}	f_4, f_5, f_6	$(f_4, f_5), (f_5, f_6), (f_6, b_6)$
s_{11}	c_7, c_8	$(c_8, f_5), (c_6, f_4), (c_6, c_8), (c_7, c_8), (c_4, c_7), (c_7, f_6)$
s_{12}	b_{13}, b_{14}, b_{15}	$(c_7, b_{14}), (c_6, b_{13}), (c_8, b_{15}), (b_{14}, b_{15}), (b_{13}, b_{15}), (b_{14}, b_4)$
s_{13}	$f_{15}, f_{16}, f_{17}, f_{18}, f_{19}$	$(f_{15}, f_{16}), (f_{16}, f_{18}), (f_{17}, f_{19}), (f_{18}, f_{19}), (f_{19}, b_8)$
s_{14}	$c_{14}, c_{15}, c_{16}, c_{17}$	$(c_{14}, f_{15}), (c_{15}, f_{16}), (c_{16}, f_{17}), (c_4, f_{18}), (c_{17}, f_{19}), (c_{14}, c_{15}), (c_{15}, c_4), (c_{17}, c_4), (c_{16}, c_{17})$
s_{15}	$b_{26}, b_{27}, b_{28}, b_{29}, b_{30}, b_{31}, b_{32}$	$(c_{14}, b_{26}), (c_{15}, b_{27}), (c_{16}, b_{28}), (c_{17}, b_{29}), (c_4, b_{30}), (c_4, b_{31}), (c_{13}, b_{32}), (b_{26}, b_{27}), (b_{27}, b_{30}), (b_{30}, b_{29}), (b_{28}, b_{29}), (b_{29}, b_{31}), (b_{32}, b_{31})$
s_{16}	f_8, f_9, f_{10}	$(f_8, f_9), (f_9, f_{10}), (f_{10}, b_{16})$
s_{17}	c_{10}, c_{11}	$(c_6, f_8), (c_{11}, f_9), (c_{10}, f_{10}), (c_6, c_{11}), (c_{11}, c_{10}), (c_9, c_{10})$
s_{18}	$b_{17}, b_{18}, b_{19}, b_{20}, b_{21}$	$(c_6, b_{17}), (c_{11}, b_{18}), (c_{11}, b_{21}), (c_{10}, b_{19}), (c_{10}, b_{20}), (b_{16}, b_{19}), (b_{19}, b_{20}), (b_{19}, b_{18}), (b_{18}, b_{21}), (b_{17}, b_{21})$
s_{19}	f_{20}	(f_{20}, b_{32})
s_{20}	c_{18}	$(c_{18}, f_{20}), (c_{13}, c_{18})$
s_{21}	b_{33}	$(c_{18}, b_{33}), (b_{33}, b_{32})$

Table A.5. Functions f , structures c and behaviors b of the VSRS

f_1	Transport light	f_8	Control shutter	f_{15}	Generate voltage
f_2	Analyze light	f_9	Generate force	f_{16}	Transport power
f_3	Raise warning	f_{10}	Transport force	f_{17}	Inject gas
f_4	Determine setpoint	f_{11}	Control light source	f_{18}	Ignite plasma
f_5	Generate signal	f_{12}	Generate reference light	f_{19}	Erode mirror
f_6	Move mirror	f_{13}	Return light	f_{20}	Protect reflector
f_7	Block path	f_{14}	Estimate reflectivity	–	–
c_1	Heating beam	c_7	Stepper motor	c_{13}	Reflector
c_2	Plasma	c_8	Actuator controlbox	c_{14}	RF generator
c_3	Reactor wall	c_9	Shutter	c_{15}	Electrical feedthrough
c_4	Optics	c_{10}	Mechanical feedthrough	c_{16}	Gas valves
c_5	Spectrometers	c_{11}	Pneumatic actuator	c_{17}	Cleaning plasma
c_6	Computers	c_{12}	Calibrated light source	c_{18}	Garage
b_1	Beam injection	b_{12}	Spectroscopy analysis	b_{23}	Gas discharge
b_2	Shinethrough	b_{13}	Alignment algorithm	b_{24}	Reflection
b_3	Melting	b_{14}	Electromagnetics	b_{25}	Reference comparison
b_4	Nuclear fusion	b_{15}	Electronics	b_{26}	Electronics
b_5	Impurity transport	b_{16}	Motion	b_{27}	Conduction
b_6	Displacement	b_{17}	Shutter control	b_{28}	Pneumatics
b_7	Particle deposition	b_{18}	Pneumatics	b_{29}	Plasma dynamics
b_8	Reflection	b_{19}	Motion	b_{30}	Charge
b_9	Heating	b_{20}	Vacuum leaking	b_{31}	Erosion
b_{10}	Photodetection	b_{21}	Valve movement	b_{32}	Redeposition
b_{11}	Dispersion	b_{22}	Blocking	b_{33}	Particle blocking