Functional requirements decomposition in set-based design

Minghui Sun, Zhaoyang Chen, Georgios Bakirtzis, Hassan Jafarzadeh, Cody Fleming

Design Science

Functional requirements decomposition in set-based design

Minghui Sun[1], Zhaoyang Chen[2], Georgios Bakirtzis[3], Hassan Jafarzadeh[4] and Cody Fleming[2]

[1]*Nanjing University of Information Science and Technology, China*
[2]*Iowa State University, United States*
[3]*LTCI, Télécom Paris, Institut Polytechnique de Paris, France*
[4]*University of Virginia, United States*

Abstract

Designing systems is typically uncertain and ambiguous at the early stages. Set-based design (SBD) supports alternative exploration and gradual uncertainty reduction during the early lifecycle, making it practical for complex system design. In parallel, functional requirements decomposition helps to advance the design incrementally. However, current literature on SBD lacks formal guidance on how to decompose functional requirements. To bridge this gap, we introduce a four-step method to decompose functional requirements for SBD hierarchically. We systematically define, reason and narrow the sets, breaking down the functional requirements into formal sub-requirements. This method allows parallel abstraction, ensuring the resulting system satisfies the top-level functional requirements.

Keywords: Requirements decomposition, set-based design, parallel development, functional requirements, uncertainty and ambiguity

1. Introduction

Modern engineered systems are usually developed using an interdisciplinary approach focused on defining customer (or stakeholder needs), necessary functionality and documenting requirements early in the development cycle. These requirements are then used to proceed with design synthesis and system validation while considering myriad factors such as operations, cost, schedule, performance, human support and so on (Booton & Ramo 2008; Kapurch 2010; Eisner 2011). Design complexity and the time and cost constraints emanating from competitive market needs often require critical decisions to be undertaken very early in the design process (Al Handawi *et al.* 2024). A critical aspect of this approach is the identification and decomposition of technical requirements, which ultimately form the foundation for the architecture, design, integration and verification of the resulting system (INCOSE 2023).

One of the challenges in requirements decomposition is the inherent uncertainty and ambiguity in the early stages of designing systems (Schrader, Riggs & Smith 1993). For example, design parameters can exhibit high levels of uncertainty, and such uncertainty – both aleatoric and epistemic – must be appropriately characterized to enable its impact to be assessed and subsequently mitigated or tolerated; requirements and their decomposition methods – often expressed in

Received 02 March 2025
Revised 27 January 2026
Accepted 28 January 2026

Corresponding author
Cody Fleming
flemingc@iastate.edu

Des. Sci., vol. 12, e12
journals.cambridge.org/dsj
DOI: 10.1017/dsj.2026.10053

natural language – are notoriously ambiguous, leading to confusion and errors during the decomposition process; and the deliberate ambiguity embedded in high-level requirements, intended to allow flexibility for lower-level creative implementations, introduces additional challenges in maintaining transparency and traceability during cross-level ambiguity refinement. Moreover, developing complex systems (e.g., commercial aircraft) usually involves a number of (globally) distributed design teams. The fact that independent design teams (for subsystems and components) operate as semi-autonomous units with loose inter-team coordination, particularly in large projects (Rismiller 2023) makes addressing this challenge even more daunting. It has been noted that system failures are equivalent to requirements failures (Leveson & Turner 1993; Leveson 2016), with system accidents or other failures often due to missing, flawed or inconsistent requirements (Fleming & Leveson 2015).

Set-based design (SBD) (Ward 1989; Toche, Pellerin & Fortin 2020) arises as a potential candidate for the above challenges and is particularly suited in several ways as it supports robust development of design alternatives, uncertainty reduction and resolution (Shallcross 2021) and subsystem autonomy and optimization, that is the ability for subsystems and parts to be developed separately and in parallel (Ghosh & Seering 2014). Function decomposition features such set-based reasoning (Ghosh & Seering 2014). In SBD, the overall system design problem is typically divided into multiple distinct disciplines, each exploring their own sets of possibilities (Specking *et al.* 2018a). Teams of engineers develop a set of design alternatives in parallel at different levels of abstraction and progressively narrow down the prospective alternatives based on additional information until a final solution is reached (Sobek, Ward & Liker 1999). SBD is defined in greater detail in Section 2. Although the aforementioned requirements engineering challenges broadly, and the function decomposition process specifically, may not be nominally related to SBD, function decomposition has been noted to feature set-based reasoning (Ghosh & Seering 2014).

To tackle uncertainty and ambiguity, one of the most effective approaches is to formalize. However, there has been limited formal guidance in the SBD literature on how to define, reason and narrow sets while improving the level of abstraction of the design (Specking *et al.* 2018a; Dullen *et al.* 2021). Therefore, we propose a four-step method in this paper that applies set-based reasoning to systematically and formally define, reason and narrow the sets, eventually decomposing the functional requirements into sub-requirements. Specifically, this paper has two main objectives:

1. Creating a formal method to decompose functional requirements based on SBD principles to tackle ambiguity and uncertainty in requirements decomposition.
2. Ensuring Objective 1 supports parallel development, that is, autonomy among lower-level design teams.

For Objective 1, in addition to addressing a fundamental challenge in requirements generation, our method makes two significant contributions to the SBD literature:

1. Most SBD approaches formulate functional requirements as ranges within performance spaces, which are applicable to many mechanical components. However, for higher-level systems that may involve a collection of mechanical

systems as well as software and other types of components, functions are typically defined as generic transformations between inputs and outputs. Accordingly, functional requirements must be defined as mappings between input and output ranges. Our method focuses on this latter formulation, addressing requirements development for complex systems and complementing existing SBD literature.

2. In the design community, there are two ways to address uncertainties (Eckert, Isaksson & Earl 2019): "buffer" (the portion of parameter values that compensates for uncertainties) and "excess" (the value over and above any allowances for uncertainties). The current SBD literature does not explicitly distinguish between buffer and excess when addressing uncertainties, lacking specificity in robustness claims. Our method clearly distinguishes between buffer and excess in their respective steps, where buffer is used to reduce controllable uncertainties, and excess accommodates the possibility of underestimated initial uncertainties.

For Objective 2: SBD claims that the autonomy of design teams is an advantage (Matthews *et al.* 2014), but very little SBD work provides a priori proof of this property. In fact, one cannot rigorously justify such a claim without an explicit underlying formalism. Based on the formalism defined in this paper, we can prove that when functional requirements are decomposed in a specific way (i.e., composable and refinement), individual design teams can work independently and the resulting system will satisfy the top-level functional requirements, realizing the promise of parallel development.

Figure 1 is an overall description of the structure of the proposed approach. After introducing the background information and the preliminaries (Sections 2 and 3), we address Objective 2 first in Section 4 (the left half of Figure 1). It was concluded that to ensure parallel development, the sub-requirements at each level of abstraction must be *composable* and able to *refine* the higher-level functional

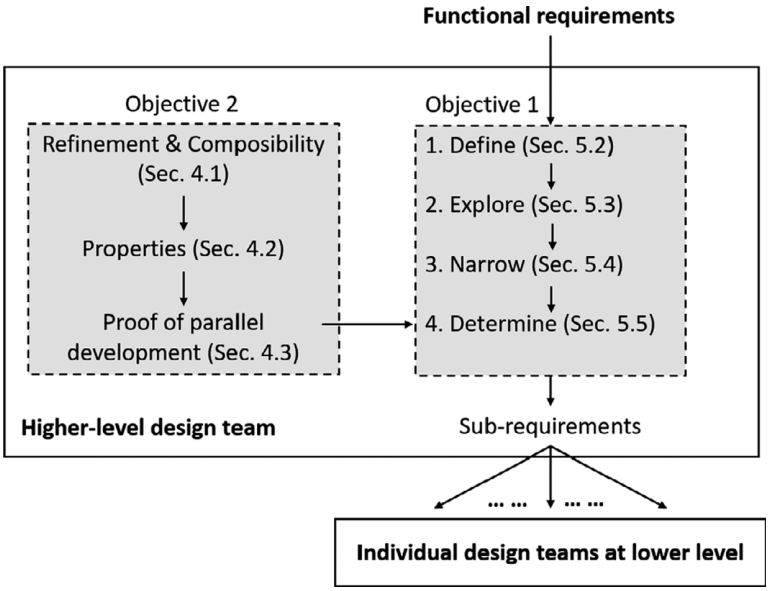

Figure 1. The overall structure of the proposed approach.

requirements as a whole. This finding will be treated as the constraints of the optimization problem of the requirements decomposition process. After that, we explain the requirements decomposition process based on SBD principles (the right half of Figure 1) in Section 5. Given the functional requirements from the customers or a higher-level design team, a four-step process is applied to define, explore and narrow the possible space of alternatives (Sections 5.2–5.4) and eventually determine the sub-requirements that will be assigned to design teams at the lower levels through an optimization program (Section 5.5). We demonstrate the method with an example of designing a cruise control system based on existing computational tools in Section 6.

2. Background

2.1. Set-based design (SBD)

In product development, two main strategies exist for selecting a concept: point-based design (PBD) and SBD. PBD involves choosing a single "best" option early from all possibilities, whereas SBD progressively eliminates infeasible or less attractive alternatives until only one solution remains (Morgan & Liker 2020). PBD often causes problems because decisions are made too early, based on assumptions or incomplete knowledge, leading to rework, correction loops and risk aversion that can stifle innovation (Majerus 2017).

SBD addresses these issues by maintaining multiple feasible options longer and narrowing them down as knowledge grows. SBD is a structured engineering methodology that emphasizes exploring a wide spectrum of design alternatives rather than fixating on a single solution from the outset. Originating in lean product development, most notably at Toyota (Ward *et al.* 1995), this approach promotes delaying commitment until sufficient knowledge is amassed, enabling more robust trade-offs and reducing costly rework (Verma 2023). In SBD, cross-disciplinary teams concurrently maintain multiple feasible design sets, gradually converging through feasibility assessment and intersection of acceptable solution spaces (SEBoK 2025). Unlike traditional point-based or sequential design methods, SBD fosters greater parallelism, improved organizational learning and the capacity to manage uncertainty – especially in early stages with loosely defined requirements or conflicting constraints. Three principles guide SBD: map the design space, integrate by intersection and establish viability before committing (Sobek *et al.* 1999). Mapping the design space involves defining feasibility regions, exploring trade-offs via multiple alternatives and communicating sets (Kerga, Taisch & Terzi 2013). Integration by intersection ensures that acceptable solutions lie in the overlapping feasible sets of all subsystems, promoting compatibility. Finally, narrowing occurs gradually by refining alternatives, remaining within feasible regions and managing uncertainty at decision gates. Together, these principles make SBD a structured approach for fostering robustness and innovation while avoiding premature commitment. A line of research was conducted recently to examine, by applying computational methods, the benefits and drawbacks of SBD, specifically how SBD interacts with various team and problem structures, using problem coupling and varying timespan to further elucidate the impacts of the process (Rismiller, Cagan & McComb 2023, 2024). The results suggest that SBD lowers the need for communication and rework on coupled

problems, allowing projects to achieve greater concurrency and break through barriers in the design space to create new designs (Rismiller 2023).

Recent studies have extended SBD to complex engineering systems, integrating model-based systems engineering (MBSE) to systematically evaluate design options using computational models (Specking *et al.* 2018b). Advanced applications in automotive, aerospace and construction show that combining SBD with optimization and tradespace exploration enables designers to handle uncertainty, balance conflicting requirements and improve performance early in the development process (Ishikawa 2024; Kazuki *et al.* 2024; Oyama & Kato 2024; van Heerden *et al.* 2024). Recent reviews (Castañeda *et al.* 2023; de Oliveira *et al.* 2024) further confirm that, while theoretical development is still evolving, growing research underscores SBD's growing relevance in enabling lean, flexible and resilient design processes.

2.2. SBD for requirements

2.2.1. Qualitative SBD approaches

It was pointed out that "there is limited SBD research contributing to requirements development (Shallcross *et al.* 2020)" and that SBD methodologies applied to complex systems are mostly qualitative (Shallcross 2021). Enhanced function-means modeling (EF-M) is one such method used for function modeling (Müller *et al.* 2019). It has been combined with SBD to manage platform-based product family design (Raudberget, Michaelis & Johannesson 2014). Functional decomposition and solutions are generated according to EF-M, with mappings from functional requirements (FR) to design solutions (DS) at each level of abstraction (Levandowski, Raudberget & Johannesson 2014). These mappings align conceptually with our proposed method. A systematic, knowledge-focused framework was proposed to guide early-phase design exploration in a set-based, front-loaded manner (Gamage & Sobek 2024). This framework offers a step-by-step approach to identifying knowledge requirements in design spaces characterized by multiple objectives and design parameters, ensuring better-informed decisions and fostering innovation in complex product development.

A "wayfaring" model for set-based requirement generation was introduced to discover critical functionalities and create dynamic requirements (Kriesi *et al.* 2016). This model uses prototyping to guide the design process from initial concept to final product, minimizing tooling and production costs. Model-based requirements and architecture (MBRMA), a novel set-based approach, was developed to filter out weak or costly solutions and assist system engineers in trade-off analysis (Borchani *et al.* 2019). Although this approach defines a mathematical formalism, it lacks specific guidance on requirement decomposition. A framework based on set-based engineering allows for building reusable and adaptable engineering methods (Vallhagen *et al.* 2013). These "virtual methods" help create and validate early phase design requirements, ensuring the introduction of novel technologies without compromising risk and cost. Other frameworks include CONGA (Al-Ashaab *et al.* 2013b), DMIV (Ammar *et al.* 2017), MBSS (Yvars & Zimmer 2022), RR-LeanPD model (Al-Ashaab *et al.* 2013a) and a combination of V-model and SBCE (Al-Ashaab *et al.* 2009).

One limitation of these qualitative approaches is the ambiguity surrounding the concrete activities needed for requirements development. This ambiguity makes it

challenging for industry practitioners to replicate these methods. A formal method provides precise, repeatable instructions for decomposing functional requirements and assigning them to lower abstraction levels.

2.2.2. Quantitative SBD approaches

Quantitative techniques for design space exploration in SBD often feature hierarchical decomposition. For example, SBD was applied to the design of a downhole module to validate a laboratory-developed method in an actual industry setting (Madhavan *et al.* 2008). The module was decomposed into chassis and bumper subsystems, with design targets assigned based on a downhole assembly impact model. A set-based approach was used for a multilevel design problem of negative stiffness metamaterials, progressing from macro-level to micro-level (Matthews *et al.* 2016). Both studies feature hierarchical decomposition but differ from our problem formulation by treating higher-level design spaces as requirements for lower-level design spaces.

Jansson *et al.* combined SBD with axiomatic design to evaluate multiple design alternatives against functional requirements (Jansson, Schade & Olofsson 2013). This work mapped higher-level design spaces to lower-level functional requirements but did not detail the derivation process. Shahan & Seepersad proposed a set-based approach to collaborative design, using Bayesian networks to identify compatibilities and conflicts among distributed subsystems (Shahan & Seepersad 2012). However, this approach focused on optimizing top-level objectives rather than decomposing requirements.

Airplane design requirements can be decomposed into design parameters for various components using SBD principles (Kerga *et al.* 2014). Although educational, this game effectively demonstrated the derivation of lower-level design parameter ranges from higher-level parameters. Panchal *et al.* presented an interval-based constraint satisfaction method for decentralized, collaborative multifunctional design, systematically reducing interval-based design variables to satisfy requirements (Panchal *et al.* 2007). SBD was also used to inform system requirements and evaluate design options for a UAV, demonstrating comprehensive tradespace exploration and valuable insights into requirement development (Parnell *et al.* 2019; Specking *et al.* 2021).

In summary, these approaches treat functional requirements as ranges of performance space variables, unlike our method, which defines requirements as mappings between design and performance spaces. This distinction significantly impacts the requirements decomposition process, with our method focusing on mapping from higher-level function mappings to sub-function mappings.

2.3. Design uncertainty

2.3.1. The SBD approach

SBD is known for its robustness to design uncertainty (Terwiesch, Loch & Meyer 2002; Costa & Sobek 2003; Bogus, Molenaar & Diekmann 2006). The US Naval Sea Systems Command highlighted SBD's suitability for design problems with conflicting requirements and high uncertainty (Parker *et al.* 2017). Trueworthy corroborated this, noting SBD's flexibility in adjusting to changing requirements (Trueworthy *et al.* 2019). Specifically, flexibility is implemented by initially setting

up broad specifications and loose constraints, which can be narrowed down as details emerge (Al Handawi *et al.* 2024).

An approach incorporating Bayesian network classifiers defined "design flexibility" as the subspace size producing satisfactory designs and "performance flexibility" as the feasible performance space size relative to the desired space (Matthews *et al.* 2014). An SBD methodology was proposed to obtain scalable optimal solutions that satisfy changing requirements through remanufacturing, demonstrated on a structural aeroengine component (Al Handawi *et al.* 2021). Another line of work combines SBD with platform-based design to preserve design bandwidth, allowing flexibility in different products (Müller, Siiskonen & Malmqvist 2020). A dynamic platform modeling approach represented production variety streams, reducing the risk of late and costly modifications (Landahl *et al.* 2021). Modeling platform concepts in early phases and eliminating undesired regions of the design space were described by (Levandowski *et al.* 2013; Levandowski *et al.* 2014; Levandowski, Müller & Isaksson 2016). In addition, McKenney *et al.* conducted a design experiment showing that delaying decisions using SBD increases adaptability to requirement changes (McKenney, Kemink & Singer 2011). A method called dynamically constrained SBD provided a dynamic map for feasible design space under varying requirements using parametric constraint sensitivity analysis and convex hull techniques (Kizer & Mavris 2014). This method identified a robust, feasible design space and a flexible family of solutions. A hybrid agent approach for set-based conceptual ship design found the process robust to intermediate design errors (Parsons & Singer 1999). Similar work can be found in (Ross 2017; (Rapp *et al.* 2018).

According to Tackett *et al.*, "excess" is "the quantity of surplus in a system once the necessities are met (Tackett, Mattson & Ferguson 2014). Eckert *et al.* later defined "excess" as "the value over and above any allowances for uncertainties" and "buffer" as "the portion of parameter values that compensates for uncertainties (Eckert *et al.* 2019). Current SBD literature lacks an explicit distinction between buffer and excess when addressing uncertainties, reducing specificity in robustness claims.

2.3.2. The margin-based approach

The aforementioned concepts of "buffer" and "excess" stem from a line of research called "design margin" (Eckert *et al.* 2019). In the seminal papers (Clarkson, Simons & Eckert 2004; Claudia Eckert & Zanker 2004), margin was a concept closely related to change and tolerance of changes. In more recent years, we have seen significant movements in this field. El Fassi *et al.* proposed a graph-theoretical approach – an assumption network – to support the allocation and management of design margins (El Fassi, Guenov & Riaz 2020). A design belief network (DBN) was built to help designers determine whether changes can be absorbed without cascading modifications downstream. A method called the margin value method (MVM) was used to analyze an engineering design, localize the excess margin and quantify it considering change absorption potential in relation to design performance deterioration (Brahma & Wynn 2020). MVM was then adapted in the context of uncertainty in input specifications so that undesirable effects of margins can be minimized while preventing change propagation (Brahma, Wynn & Isaksson 2022). Later, the same team expanded this method to evaluate sets of concepts

that are combinatorially generated from an enhanced function-means tree by using surrogate models and novel metrics for evaluating different conceptual alternatives (Al Handawi *et al.* 2024).

"Resilient objects" provide passive protection against disruptive events so that the necessity for complex margins can be reduced at system interfaces (Panarotto & Fernández 2024). It offered a new perspective on how to solve the trade-offs between resilience and complexity when addressing uncertainties in the dynamic landscape of product development. Five robustness indicators were introduced (Juul-Nyholm & Eifler 2024) based on multi-objective design exploration to tackle two central challenges in engineering design: (1) defining design margins early in the design process, before detailed modeling is possible, and (2) evaluating interdependencies among those margins to prevent cascading effects. As a result, three distinct margin types were identified to help designers assess margin implications even before models are fully defined. A more holistic review of margins throughout the product development process has been explored elsewhere (Brahma *et al.* 2024).

In comparison, although our approach is closely related to margin, our problem is different from those addressed in the current literature on margin. First, the current literature focuses on the margin of design parameters, while our approach aims to introduce margin into the domain and the co-domain of the functions. Furthermore, the concept of hierarchical refinement is relatively rare in the current literature. As recently pointed out, the concept of hierarchy in the margin literature is not well understood (Brahma *et al.* 2024). While Jacobson and Ferguson touched on this concept (Jacobson & Ferguson 2023), the proposed process was largely informal. In fact, the authors admitted at the end of the paper that better modeling approaches around decomposition are still needed. Finally, although this paper is not originally aimed to contribute to the margin community, we recognize that many concepts utilized in this paper indeed resemble margin (see Section 7.2). We believe the formal framework proposed by this paper can contribute to the vision of a design environment that will eventually connect margin to requirements (Ferguson *et al.* 2024).

3. Preliminaries

In general, functions describe the behavior between inputs and outputs of a system, subsystem, or component. Such a description can be an informal summary or statement, or a formal specification, all the way up to a literal mathematical *function* prescribing the desired input–output behavior. In the following, we take the latter perspective: the function of a system is a mathematical input–output transformation that describes the overall behavior of a system (Buede 1997; Larson & Mattson 2012).

$$(x,u,c) \xrightarrow{f} y \tag{1}$$

- f represents the transformation between (x,u,c) and y.
- x represents the input variables that the system takes from the environment or other systems.
- y represents the output variables that the system gives in response to the inputs.

- u represents the uncertain design parameters that are out of the designer's control, such as the weather for an airplane. We call them the uncontrollable parameters.
- c represents the uncertain design parameters that are under the designer's control, such as the weight of an airplane. We call them the controllable parameters.

Note that x,y,u and c are all column vectors of f, as a function likely has multiple input variables, output variables, controllable parameters and uncontrollable parameters.

3.1. Notation

We introduce the notation used throughout this paper.

- f and f_i: f is the higher-level function to be decomposed into multiple sub-functions f_i, where $i = 1,2,\ldots,m$.
- $a(i)$: As mentioned above, x,y,u and c are the four column vectors of f. Similarly, the four column vectors of the sub-function f_i are denoted as $x(i),y(i),u(i)$ and $c(i)$. Abstractly, we denote these column vectors with a unified expression $a(i)$, where $a(i) = (a_1(i),a_2(i),\ldots,a_n(i))^\top$.
- $a_j(i)$, as the j^{th} element of $a(i)$, represents one individual input/output variable or controllable/uncontrollable parameter of sub-function f_i (where $j = 1,2\ldots,n$).
- $R_{a_j(i)}$ is the range of $a_j(i)$, that is, $a_j(i) \in R_{a_j(i)}$.
- $\{a(i)\}$ is a set representation of $a(i)$, that is, $\{a(i)\} = \{a_1(i),a_2(i),\ldots,a_n(i)\}$.
- $R_{a(i)} = \left(R_{a_1(i)},R_{a_2(i)},\ldots,R_{a_n(i)}\right)^\top$ is a column vector of the ranges of $a(i)$.
- $\left\{R_{a(i)}\right\}$ is a set representation of $R_{a(i)}$, that is, $\left\{R_{a(i)}\right\} = \left\{R_{a_1(i)},R_{a_2(i)},\ldots,R_{a_n(i)}\right\}$.

We now define the following set operations, where a and b are two column vectors.

- $\{a\}\sqcup\{b\}$ returns the union of the identifiers (instead of the values) of all the elements in $\{a\}$ and $\{b\}$. The identical variable between $\{a\}$ and $\{b\}$ is merged into one element. By "identical," we mean the elements that have the same physical meaning, the same data source and hence the same value *at all times*.
- $\{a\}\sqcap\{b\}$ returns the identifier of the identical variable(s) between $\{a\}$ and $\{b\}$.
- $\{a\}\sqsubseteq\{b\}$ returns *true* if the identifiers (instead of the values) of all the variables in $\{a\}$ are included in $\{b\}$.
- $\{R_a\}\Cup\{R_b\}$ returns the set of the ranges of $\{a\}\sqcup\{b\}$. Because the identical elements between $\{a\}$ and $\{b\}$ are subject to the ranges in both $\{R_a\}$ and $\{R_b\}$, the resulting ranges for the identical elements are the intersections ($\cap$) of the respective ranges in $\{R_a\}$ and $\{R_b\}$.
- $a_j|\{R_b\}$ identifies the element in $\{b\}$ that is identical to a_j and returns its range in $\{R_b\}$.
- $\mathcal{TR}(\{R_a\})$ transforms the set $\{R_a\}$ into the column vector R_a,that is, $\mathcal{TR}(\{R_a\}) = R_a$.

3.2. Functional requirements

Mathematically, a system function, as defined in Eq. (1), must satisfy Eq. (2), where $\left(R_x^*,R_u^*,R_c^*,R_y^*\right)$ constitute the functional requirements of the system.

$$\forall x \in R_x^*, \forall u \in R_u^*, \exists c \in R_c^* : (x,u,c) \xrightarrow{f} y \in R_y^* \tag{2}$$

- $\forall R_x^*$ implies the system must be able to process all the values from the input set.
- $\forall R_u^*$ is because u is out of the designer's control. The system must be able to process all possible u.
- $\exists R_c^*$ is because c is under the designer's control. The designer only needs to find one c within R_c^* to satisfy the functional requirements.
- R_y^* is the co-domain of f. All possible output values must be bounded within R_y^*.

However, not any random $\left(R_x^*, R_u^*, R_c^*, R_y^*\right)$ can be the requirements of f. $\left(R_x^*, R_u^*, R_c^*\right)$ and R_y^* must satisfy Eq. (3), which implies that the functional requirements of a system are actually a constrained mapping between $\left(R_x^*, R_u^*, R_c^*\right)$ and R_y^*.

$$\left(R_x^*, R_c^*, R_u^*\right) \rightarrow R_y^* \text{ s.t.Eq.}(2) \tag{3}$$

Similarly, the requirements of f_i can be represented as:

$$\left(R_{x(i)}^*, R_{c(i)}^*, R_{u(i)}^*\right) \rightarrow R_{y(i)}^* \text{ s.t.Eq.}(2) \tag{4}$$

Therefore, **the functional requirements decomposition problem** is to decompose the top-level requirements in Eq. (3) into a set of sub-requirements in Eq. (4) at the lower level, repeat the same process independently at each level of abstraction and eventually lead to a system that satisfies the top-level requirements in Eq. (3).

4. Parallel development

We explain in this that refinement and composability are the two constraints to ensure parallel development (i.e., Objective 2).

4.1. Refinement and composability

Refinement and composability play an important role in producing traceable requirements.

4.1.1. Refinement

Refinement defines a relationship between two sets of functional requirements (FR). FR' refining FR implies the reduction of the uncertainties from FR to FR', which can always be reflected in the mapping between the input and output variables. As shown in Figure 2(a), FR' refines FR if the relationship in Eq. (5) is satisfied, where the meanings of $x_i|\left\{R_{x'}^*\right\}$ and $y_i|\left\{R_{y'}^*\right\}$ are defined in Section 3.1.

$$\begin{cases} \{x\} \sqsubseteq \{x'\} \wedge \forall x_i \in \{x\} : R_{x_i}^* \subseteq x_i|\left\{R_{x'}^*\right\} \\ \{y\} \sqsubseteq \{y'\} \wedge \forall y_i \in \{y\} : y_i|\left\{R_{y'}^*\right\} \subseteq R_{y_i}^* \end{cases} \tag{5}$$

Intuitively, Eq. (5) means (1) FR' defines all the input/output variables of FR, and (2) FR' allows the system to take a larger range of x but will generate a smaller range of y than FR.

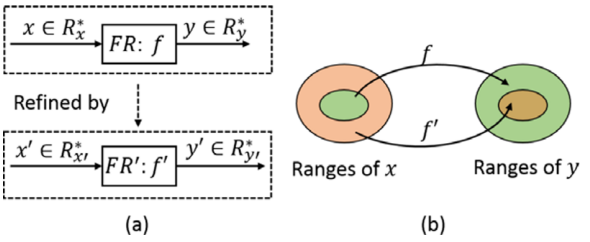

Figure 2. Graphical illustration of *refinement*. The green circles are the ranges defined in FR, and the brown circles are the ranges defined in FR'.

Note that a system satisfying the functional requirements is a special case of *refinement*. As implied by Eq. (2), the system can take all the values in R_x^* and provide the output values that are bounded within R_y^*, which satisfies the relationship of *refinement* in Eq. (5).

4.1.2. Composability

Composability defines the interface between functional requirements. Therefore, only the input and output variables are concerned. Two sets of functional requirements (denoted as FR_j and FR_k in Figure 3(a)) are composable if the relationship in Eq. (6) below is satisfied.

$$\begin{cases} \{z\} = \{y(j)\} \sqcap \{x(k)\} \neq 0 \\ \forall z_i \in \{z\} : z_i | \left\{ R_{y(j)}^* \right\} \subseteq z_i | \left\{ R_{x(k)}^* \right\} \end{cases} \tag{6}$$

Intuitively, *composability* means two things: (1) the output of FR_j and the input of FR_k share at least one common variable (denoted as $\{z\}$), and (2) FR_k can take in more values of z than FR_j can output (Figure 3(b)).

4.2. Properties of refinement and composability

Based on the definitions above, we have the following properties.

Property 1. *If FR' refines FR, and FR'' refines FR', then FR'' refines FR.*

Property 2. *If a set of functional requirements $\{FR_1, FR_2, \ldots, FR_n\}$ are composable, then their refinements $\{FR'_1, FR'_2, \ldots, FR'_n\}$ are also composable.*

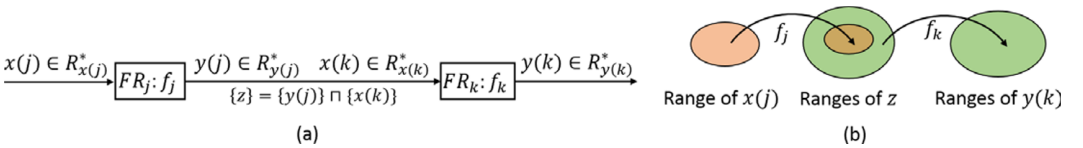

Figure 3. Graphical illustration of *composability*. The green circles are the ranges defined in FR_j, and the brown circles are the ranges defined in FR_k.

Property 3. *Consider a set of composable functional requirements* $\{FR_1, FR_2, \ldots, FR_n\}$. *If* FR_i' *refines* FR_i *(i = 1,2,n), then the composite of* $\{FR_1', FR_2', \ldots, FR_n'\}$ *refines the composite of* $\{FR_1, FR_2, \ldots, FR_n\}$.

Because a system satisfying the functional requirements is a special case of *refinement*, Properties 4, 5 and 6 can be derived similarly to Properties 1, 2 and 3.

Property 4. *If* FR' *refines FR, and a system S satisfies* FR', *then S satisfies FR.*

Property 5. *If a set of functional requirements* $\{FR_1, FR_2, \ldots, FR_n\}$ *are composable, and a set of systems* $\{S_1, S_2, \ldots, S_n\}$ *satisfy the functional requirements, respectively, then* $\{S_1, S_2, \ldots, S_n\}$ *are also composable.*

Property 6. *Consider a set of composable functional requirements* $\{FR_1, FR_2, \ldots, FR_n\}$ *and their composite FR. If* S_i *satisfies* FR_i *(i = 1,2,…,n), then the composite of* $\{S_1, S_2, \ldots, S_n\}$ *satisfies the composite of* $\{FR_1, FR_2, \ldots, FR_n\}$.

4.3. Ensuring parallel development

In this subsection, we illustrate, with a simple system (Figure 4), the hierarchical functional requirements decomposition process and prove the conditions under which the decomposed requirements ensure the top-level functional requirements are satisfied.

First, the given top-level functional requirements *FR* are decomposed into $\{FR_1^1, FR_2^1\}$ and assigns them to the Level-1 suppliers. The Level-1 suppliers then independently decompose FR_1^1 and FR_2^1 into $\{FR_{1,1}^2, FR_{1,2}^2\}$ and $\{FR_{2,1}^2, FR_{2,2}^2\}$, respectively, and assign these to Level-2 suppliers. The Level-2 suppliers implement the functional requirements independently in individual systems $S_{1,1}^2, S_{1,2}^2, S_{2,1}^2$, and $S_{2,2}^2$ that satisfy the respective functional requirements.

We now prove that if at each level of abstraction, (1) the sub-requirements are *composable*, and (2) when composed together, the sub-requirements *refine* the higher-level functional requirements, then $S_{1,1}^2, S_{1,2}^2, S_{2,1}^2$, and $S_{2,2}^2$ are composable and together satisfy *FR*.

Proof:

1. By definition, $\{FR_1^1, FR_2^1\}$ are composable and refine *FR*; $\{FR_{1,1}^2, FR_{1,2}^2\}$ are composable and refine FR_1^1; $\{FR_{2,1}^2, FR_{2,2}^2\}$ are composable and refine FR_2^1; $S_{1,1}^2, S_{1,2}^2, S_{2,1}^2$ and $S_{2,2}^2$ satisfy $FR_{1,1}^2, FR_{1,2}^2, FR_{2,1}^2$ and $FR_{2,2}^2$, respectively.

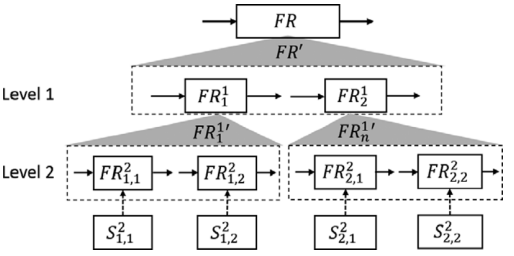

Figure 4. A two-level example system to describe the hierarchical functional requirements decomposition process.

2. According to Property 2, $\{FR^2_{1,1}, FR^2_{1,2}, FR^2_{2,1}, FR^2_{2,2}\}$ are composable. Then, the composite of $\{FR^2_{1,1}, FR^2_{1,2}, FR^2_{2,1}, FR^2_{2,2}\}$ refines the composite of $\{FR^1_1, FR^1_2\}$ according to Property 3. Because $\{FR^1_1, FR^1_2\}$ refine *FR*, the composite of the form $\{FR^2_{1,1}, FR^2_{1,2}, FR^2_{2,1}, FR^2_{2,2}\}$ refines *FR* according to Property 1.
3. Since $\{FR^2_{1,1}, FR^2_{1,2}, FR^2_{2,1}, FR^2_{2,2}\}$ are composable, and $S^2_{1,1}, S^2_{1,2}, S^2_{2,1}$ $S^2_{2,2}$ satisfy $FR^2_{1,1}, FR^2_{1,2}, FR^2_{2,1}$ and $FR^2_{2,2}$, respectively, then $\{S^2_{1,1}, S^2_{1,2}, S^2_{2,1}, S^2_{2,2}\}$ are composable according to Property 5. The composite of $\{S^2_{1,1}, S^2_{1,2}, S^2_{2,1}, S^2_{2,2}\}$ satisfies the composite of $\{FR^2_{1,1}, FR^2_{1,2}, FR^2_{2,1}, FR^2_{2,2}\}$ according to Property 6. We have established that the composite of $\{FR^2_{1,1}, FR^2_{1,2}, FR^2_{2,1}, FR^2_{2,2}\}$ refines *FR*. Then, according to Property 4, the composite of $\{S^2_{1,1}, S^2_{1,2}, S^2_{2,1}, S^2_{2,2}\}$ satisfies *FR*. In other words, the resulting system satisfies the top-level functional requirements.

Therefore, for design teams at different levels of abstraction to work in parallel and obtain a system that satisfies the top-level functional requirements, the sub-requirements at each level of abstraction must be *composable* and *refine* the higher-level functional requirements as a whole.

5. Set-based requirements decomposition

In this section, we explain the set-based requirements decomposition process to tackle ambiguity and uncertainty in requirements decomposition (i.e., Objective 1).

5.1. An overall description

Given the higher-level functional requirements, the decomposition process involves the following steps:

1. *Define and analyze the functional architecture (Section 5.2).* A set of functional architecture (comprising a set of connected sub-functions) for the system-under-design is defined to refine the higher-level functional requirements, based on which the design space and the performance space are derived.
2. *Explore the sub-functions for initial feasible spaces (Section 5.3).* A set of initial feasible spaces of the sub-functions is characterized individually by exploring the different feasible implementations for the corresponding sub-functions.
3. *Narrow the initial feasible spaces(Section 5.4).* The feasible design space and the feasible performance space of the system-under-design are narrowed gradually by taking "intersections" based on the initial feasible spaces of the sub-functions.
4. *Determine the sub-requirements (Section 5.5).* The functional requirements of each sub-function are defined based on the narrowed feasible design space and performance space so that all the sub-requirements are *composable*, and when composed together, they can *refine* the higher-level functional requirements.

After that, the team for each sub-function will improve the level of detail by independently decomposing the sub-functions into the lower levels iteratively until a final solution can be chosen.

5.2. Define and analyze the functional architecture

A functional architecture is composed of a set of connected sub-functions $(f_1, f_2, \ldots, f_n)$ and represents a set of potential design solutions for the higher-level requirements. In SBD, there can be multiple alternative functional architectures. However, from the perspective of formal reasoning, the process to derive the sub-requirements for any functional architecture is the same. Therefore, we assume in this paper that only one functional architecture is defined in this step. In addition, the identification of the functional architecture relies on the designer's understanding of the desired system (Benveniste *et al.* 2015). The techniques to identify functional architecture are out of the scope of this paper, but can be found elsewhere (Komoto & Tomiyama 2011; Penzenstadler 2011; She *et al.* 2022).

Given a functional architecture, we extract the following two pieces of information: (1) which variable/parameter is shared with which function/sub-functions, and (2) the constitution of the design space and the performance space. These will be used to narrow the feasible spaces later.

First, we aggregate $x(i), y(i), c(i)$, and $u(i)$ of the sub-functions into $\{x'\}, \{y'\}, \{c'\}$, and $\{u'\}$ without considering the functional architecture. As defined in Section 3.1, the $\sqcup$ operation merges the identical variables/parameters in multiple sub-functions into one variable/parameter in the new set.

$$\begin{cases} \{x'\} = \{x(1)\} \sqcup \{x(2)\} \sqcup \ldots \sqcup \{x(n)\} \\ \{y'\} = \{y(1)\} \sqcup \{y(2)\} \sqcup \ldots \sqcup \{y(n)\} \\ \{c'\} = \{c(1)\} \sqcup \{c(2)\} \sqcup \ldots \sqcup \{c(n)\} \\ \{u'\} = \{u(1)\} \sqcup \{u(2)\} \sqcup \ldots \sqcup \{u(n)\} \end{cases} \tag{7}$$

Second, for the system to implement the higher-level functional requirements, all the elements defined in the higher-level function f (i.e., $\{x\}, \{y\}, \{c\}$, and $\{u\}$) must also be defined in the functional architecture, i.e.,

$$\{x\} \sqsubseteq \{x'\} \bigwedge \{c\} \sqsubseteq \{c'\} \bigwedge \{u\} \sqsubseteq \{u'\} \bigwedge \{y\} \sqsubseteq \{y'\} \tag{8}$$

Third, we examine where the variables and the parameters are defined. The elements of $\{c'\}$ and $\{u'\}$ may or may not be defined in $\{c\}$ and $\{u\}$. We define the elements of $\{c'\}$ and $\{u'\}$ that are not defined in $\{c\}$ and $\{u\}$ as $\{\tilde{c}\}$ and $\{\tilde{u}\}$, and thus have

$$\begin{cases} \{c'\} = \{\tilde{c}\} \sqcup \{c\} \\ \{u'\} = \{\tilde{u}\} \sqcup \{u\} \end{cases} \tag{9}$$

For $\{x'\}$ and $\{y'\}$, we use Figure 5 to reason about where the elements can possibly be defined. $\{x'\}$ and $\{y'\}$ are the two outer ovals that include $\{x\}$ and $\{y\}$.

As shown in the middle of Figure 5, there are four possible different ways of where $\{x'\}$ and $\{y'\}$ can be defined. $\{x'\}$ and $\{y'\}$ overlap because the sub-functions are connected, i.e., one's input is another's output; $\{x'\}$ and $\{y\}$ overlap if there are feedback loops that feed some elements of $\{y\}$ into the sub-functions. $\{x\}$ does not overlap with $\{y'\}$ or $\{y\}$ because $\{x\}$ are the input variables of the higher-level functional requirements, hence can only come from the external environment rather than being controlled by the system itself. As a result, we

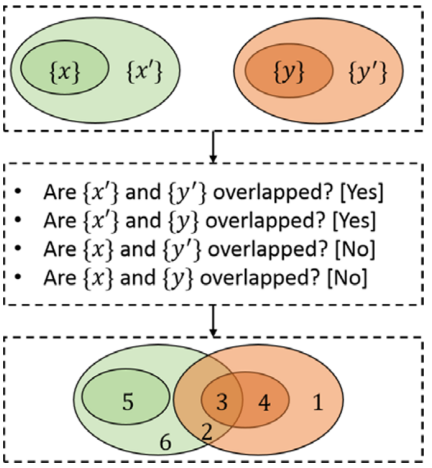

Figure 5. $\{x'\}$ and $\{y'\}$ can be divided into six groups to reason about the design space and the performance space. The ranges of each group will be calculated differently in Section 5.4.

identify six different areas (the bottom of Figure 5) depending on the overlapping between $\{x'\}, \{y'\}, \{x\}$, and $\{y\}$.

The variables in Areas 1, 2, 3 and 4 are, therefore, denoted as $\left\{y'^{(1)}\right\}$, $\left\{y'^{(2)}\right\}$, $\left\{y'^{(3)}\right\}$, and $\left\{y'^{(4)}\right\}$ in

$$\begin{cases} \left\{y'^{(1)}\right\} = \{y'\} \sqcap \neg\{y\} \sqcap \neg\{x'\} \\ \left\{y'^{(2)}\right\} = \{y'\} \sqcap \{x'\} \sqcap \neg\{y\} \\ \left\{y'^{(3)}\right\} = \{y\} \sqcap \{x'\} \\ \left\{y'^{(4)}\right\} = \{y\} \sqcap \neg\{x'\} \end{cases} \tag{10}$$

Area 5 is $\{x\}$. Area 6, denoted as $\{\tilde{x}\}$, can then be represented as

$$\{\tilde{x}\} = \{x'\} \sqcap \neg\{y'\} \sqcap \neg\{x\} \tag{11}$$

Therefore, we can classify all the elements of a functional architecture into the exclusive groups below based on where the elements are defined.

- $\{x\}$ is defined in $\{x\}$ and $\{x'\}$; $\{\tilde{x}\}$ is only defined within $\{x'\}$.
- $\{c\}$ is defined in $\{c\}$ and $\{c'\}$; $\{\tilde{c}\}$ is only defined within $\{c'\}$.
- $\{u\}$ is defined in $\{u\}$ and $\{u'\}$; $\{\tilde{u}\}$ is only defined within $\{u'\}$.
- $\left\{y'^{(1)}\right\}$ is only defined in $\{y'\}$; $\left\{y'^{(2)}\right\}$ is defined in $\{x'\}$ and $\{y'\}$; $\left\{y'^{(3)}\right\}$ is defined in $\{x'\}, \{y\}$, and $\{y'\}$; $\left\{y'^{(4)}\right\}$ is defined in $\{y\}$ and $\{y'\}$.

Finally, because the design space is the set of the independent variables/parameters of a system, and the performance space is the set of the dependent

variables (Wood & Antonsson 1989; Nahm & Ishikawa 2006), the design space and the performance space of the system can be summarized in Eq. (12).

$$\begin{cases} \text{Design space}: \{x\} \sqcup \{\tilde{x}\} \sqcup \{c\} \sqcup \{\tilde{c}\} \sqcup \{u\} \sqcup \{\tilde{u}\} \\ \text{Performance space}: \left\{y'^{(1)}\right\} \sqcup \left\{y'^{(2)}\right\} \sqcup \left\{y'^{(3)}\right\} \sqcup \left\{y'^{(4)}\right\} \end{cases} \tag{12}$$

5.3. Explore the initial feasible spaces

The initial feasible spaces of each sub-function are characterized in

$$\forall x(i) \in R_{x(i)}, \forall u(i) \in R_{u(i)}, \exists c(i) \in R_{c(i)} : f_i(x(i), u(i), c(i)) \in R_{y(i)} \tag{13}$$

after exploring the feasible implementations. The specific initial spaces can only be determined on a case-by-case basis.

Note that Eq. (13) is different from Eq. (4). The ranges (without $*$) in Eq. (13) represent the feasible spaces of the sub-functions, while the ranges in Eq. (4) represent the desired spaces (i.e., requirements).

5.4. Narrow the feasible spaces

In this subsection, we derive the feasible design space and performance space of the system by narrowing from the initial feasible spaces.

First, we aggregate the ranges of the sub-functions $\left\{R_{x(i)}\right\}, \left\{R_{c(i)}\right\}, \left\{R_{u(i)}\right\}$ and $\left\{R_{y(i)}\right\}$ $(i = 1, 2, \ldots, n)$ into $\{R_{x'}\}, \{R_{c'}\}, \{R_{u'}\}$ and $\left\{R_{y'}\right\}$, where $\{x'\}, \{u'\}, \{c'\}$ and $\{y'\}$ are defined in Eq. (7). Refer to Section 3.1 for the definition of $\Cup$.

$$\begin{cases} \{R_{x'}\} = \left\{R_{x(1)}\right\} \Cup \left\{R_{x(2)}\right\} \Cup \ldots \Cup \left\{R_{x(n)}\right\} \\ \left\{R_{y'}\right\} = \left\{R_{y(1)}\right\} \Cup \left\{R_{y(2)}\right\} \Cup \ldots \Cup \left\{R_{y(n)}\right\} \\ \{R_{c'}\} = \left\{R_{c(1)}\right\} \Cup \left\{R_{c(2)}\right\} \Cup \ldots \Cup \left\{R_{c(n)}\right\} \\ \{R_{u'}\} = \left\{R_{u(1)}\right\} \Cup \left\{R_{u(2)}\right\} \Cup \ldots \Cup \left\{R_{u(n)}\right\} \end{cases} \tag{14}$$

Second, we calculate the ranges of $\{\tilde{x}\}, \{x\}, \{\tilde{c}\}, \{c\}, \{\tilde{u}\}, \{u\}, \left\{y'^{(1)}\right\}, \left\{y'^{(2)}\right\}, \left\{y'^{(3)}\right\}$ and $\left\{y'^{(4)}\right\}$ based on where they are defined. For example, the elements in $\{x\}$ are defined in both $\{x\}$ and $\{x'\}$. Therefore, the range of each element in $\{x\}$ will be the intersection of the corresponding ranges in R_x^* and $\{R_{x'}\}$, that is, the first item in Eq. (15). Refer to Section 3.1 for the definition of the operation $x_i|\{R_{x'}\}$.

The ranges of $\{c\}, \{u\}, \{\tilde{x}\}, \{\tilde{c}\}, \{\tilde{u}\}, \left\{y'^{(1)}\right\}, \left\{y'^{(2)}\right\}, \left\{y'^{(3)}\right\}$ and $\left\{y'^{(4)}\right\}$ can be calculated in the same way in the rest of Eq. (15).

$$
\begin{cases}
\{R_x\} = \left\{R_{x_i} | \forall x_i \in \{\mathbf{x}\} : R_{x_i} = R^*_{x_i} \cap x_i | \{R_{x'}\}\right\} \\
\{R_u\} = \left\{R_{u_i} | \forall u_i \in \{u\} : R_{u_i} = R^*_{u_i} \cap u_i | \{R_{u'}\}\right\} \\
\{R_c\} = \left\{R_{c_i} | \forall c_i \in \{c\} : R_{c_i} = R^*_{c_i} \cap c_i | \{R_{c'}\}\right\} \\
\{R_{\tilde{x}}\} = \{R_{x_i} | \forall x_i \in \{\tilde{x}\} : R_{x_i} = x_i | \{R_{x'}\}\} \\
\{R_{\tilde{c}}\} = \{R_{c_i} | \forall c_i \in \{\tilde{c}\} : R_{c_i} = c_i | \{R_{c'}\}\} \\
\{R_{\tilde{u}}\} = \{R_{u_i} | \forall u_i \in \{\tilde{u}\} : R_{u_i} = u_i | \{R_{u'}\}\} \\
\left\{R_{y'(1)}\right\} = \left\{R_{y_i} | \forall y_i \in \left\{y'^{(1)}\right\} : R_{y_i} = y_i | \{R_{y'}\}\right\} \\
\left\{R_{y'(2)}\right\} = \left\{R_{y_i} | \forall y_i \in \left\{y'^{(2)}\right\} : R_{y_i} = y_i | \{R_{x'}\} \cap y_i | \{Ry'\}\right\} \\
\left\{R_{y'(3)}\right\} = \left\{R_{y_i} | \forall y_i \in \left\{y'^{(3)}\right\} : R_{y_i} = y_i | \{R_{x'}\} \cap y_i | \{Ry'\} \cap y_i | \left\{R^*_y\right\}\right\} \\
\left\{R_{y'(4)}\right\} = \left\{R_{y_i} | \forall y_i \in \left\{y'^{(4)}\right\} : R_{y_i} = y_i | \{R_{y'}\} \cap y_i | \left\{R^*_y\right\}\right\}
\end{cases}
\tag{15}
$$

In addition, each range in $\{R_x\}, \{R_u\}, \{R_c\}, \{R_{\tilde{x}}\}, \{R_{\tilde{c}}\}, \{R_{\tilde{u}}\}, \left\{R_{y'^{(1)}}\right\}, \left\{R_{y'^{(2)}}\right\}, \left\{R_{y'^{(3)}}\right\}$ and $\left\{R_{y'^{(4)}}\right\}$ cannot be $0/$ because $0/$ implies internal conflicts between the initial ranges, unless the corresponding group of elements is not defined. Furthermore, for $\{R_x\}$ and $\{R_u\}$, the conditions are stricter. Recall $\forall$ is associated with $\left\{R^*_x\right\}$ and $\left\{R^*_u\right\}$ in Eq. (2). The system must operate under all the values defined in $\left\{R^*_x\right\}$ and $\left\{R^*_u\right\}$. For this reason, the ranges of $\{x\}$ and $\{u\}$ defined in $\{R_{x'}\}$ and $\{R_{u'}\}$ must include $\left\{R^*_x\right\}$ and $\left\{R^*_u\right\}$, respectively. As a result, we can update $\{R_x\}$ and $\{R_u\}$ in Eq. (16) based on Eq. (15).

$$
\{R_x\} = \left\{R^*_x\right\} \wedge \{R_u\} = \left\{R^*_u\right\} \tag{16}
$$

Now, let $R^1_D = \{R_x\} \Cup \{R_{\tilde{x}}\} \Cup \{R_c\} \Cup \{R_{\tilde{c}}\} \Cup \{R_u\} \Cup \{R_{\tilde{u}}\}$ and $R^1_P = \left\{R_{y'^{(1)}}\right\} \Cup \left\{R_{y'^{(2)}}\right\} \Cup \left\{R_{y'^{(3)}}\right\} \Cup \left\{R_{y'^{(4)}}\right\}$. The feasible design space and performance space can be easily represented as:

$$
FDS^1 = \mathcal{TR}(R^1_D) \wedge FPS^1 = \mathcal{TR}(R^1_P) \tag{17}
$$

Third, FDS^1 and FPS^1 are coupled through the sub-functions and can be further narrowed by exploiting the coupling. Let $f' = f_1 \wedge f_2 \wedge \ldots \wedge f_n$. f' can be characterized as:

$$
(x, \tilde{x}, c, \tilde{c}, u, \tilde{u}) \xrightarrow{f'} y' \tag{18}
$$

where $(x, \tilde{x}, c, \tilde{c}, u, \tilde{u}) \in FDS^1$ and $y' \in FPS^1$.

Based on Eq. (18), we can narrow R_c and $R_{\tilde{c}}$ to $R^{\downarrow}_c$ and $R^{\downarrow}_{\tilde{c}}$ so that Eq. (19) is satisfied, which is a quantified constraint satisfaction problem (Qureshi *et al.* 2014).

$$
\begin{aligned}
&\text{For } \forall x \in R_x, \forall \tilde{x} \in R_{\tilde{x}}, \forall u \in R_u, \forall \tilde{u} \in R_{\tilde{u}}, \forall c \in R^{\downarrow}_c, \text{ and } \forall \tilde{c} \in R^{\downarrow}_{\tilde{c}}: \\
&(x, \tilde{x}, c, \tilde{c}, u, \tilde{u}) \xrightarrow{f'} y' \in FPS^1
\end{aligned} \tag{19}
$$

Let $R_D^2 = \{R_x\} \Cup \{R_{\tilde{x}}\} \Cup \{R_u\} \Cup \{R_{\tilde{u}}\} \Cup \{R_c^{\downarrow}\} \Cup \{R_{\tilde{c}}^{\downarrow}\}$, then the new feasible design space can be represented as $FDS^2 = \mathcal{TR}(R_D^2)$.

The reason that only R_c and $R_{\tilde{c}}$ are narrowed is because other ranges in Eq. (18) are all associated with ∀, meaning the system must operate under all values in the ranges. Hence, the ranges associated with ∀ cannot be narrowed. Furthermore, the reason that the ranges of the controllable parameters $R_c^{\downarrow}$ and $R_{\tilde{c}}^{\downarrow}$ in Eq. (19) are associated with ∀ is that $R_c^{\downarrow}$ and $R_{\tilde{c}}^{\downarrow}$ will be further narrowed independently by subsystems at the lower levels of abstraction until single values can be determined for the elements in $\{c\}$ and $\{\tilde{c}\}$. "∀" makes sure that whatever values are chosen for c and $\tilde{c}$ at the lower levels, Eq. (19) will always be satisfied.

After Eq. (19), because the feasible design space is narrowed from FDS^1 to FDS^2, FPS^1 can be narrowed into a new feasible performance space FPS^2 based on the new FDS^2, such that Eq. (20) holds:

$$\forall (x, \tilde{x}, c, \tilde{c}, u, \tilde{u})^{\top} \in FDS^2 : (x, \tilde{x}, c, \tilde{c}, u, \tilde{u}) \xrightarrow{f'} y' \in FPS^2 \tag{20}$$

Reachability analysis (G. *et al.*, 2021) can be applied to calculate FPS^2 as an over-approximation so that all the points in FDS^2 will be mapped within FPS^2. Let the resulting reachable sets for $y'^{(h)}$ be $\left\{R_{y'^{(h)}}^{\downarrow}\right\}$ and $R_P^2 = \left\{R_{y'^{(1)}}^{\downarrow}\right\} \Cup \left\{R_{y'^{(2)}}^{\downarrow}\right\} \Cup \left\{R_{y'^{(3)}}^{\downarrow}\right\} \Cup \left\{R_{y'^{(4)}}^{\downarrow}\right\}$. The new feasible performance space can be represented as $FPS^2 = \mathcal{TR}(R_P^2)$.

Finally, the feasible design space and performance space of the system are narrowed to FDS^2 and FPS^2.

5.5. Determine the sub-requirements

In this subsection, we determine the sub-requirements that are *composable*, and when composed together, can *refine* the higher-level functional requirements.

Uncertainty Expansion. Ideally, the ranges in the sub-requirements should be as narrow as possible to reduce uncertainty to the greatest extent. The ranges in FDS^2 and FPS^2 are the narrowest ranges that can be calculated at the current level of abstraction. However, it is always possible that the initial characterizations of the uncertainties (i.e., $R_{u'}$ and $R_{c'}$) are under-estimated, which will inevitably lead to an expansion of FPS^2 at a later time.

To accommodate the possible unexpected expansion, the desired ranges of y' in the sub-requirements must also expand from FPS^2. We denote the desired ranges of y' in the sub-requirements as FPS^*. As shown in Figure 6, to maximize the capacity of the sub-requirements to accommodate the possible expansion, FPS^* must be expanded as close to FPS^1 as possible. Note that FPS^* cannot expand beyond FPS^1 to avoid infeasibility.

However, y' may also be the input variables of some sub-functions. Expanding the ranges of y' will inevitably lead to the expansion of the input ranges of some sub-functions, which will in turn make the system more difficult or expensive to implement. Therefore, there is an inherent **tension** between accommodating excessive uncertainty and the level of difficulty in implementing the system. This tension dictates that the boundary of FPS^* in Figure 6 cannot be too close either to FPS^1 or to FPS^2.

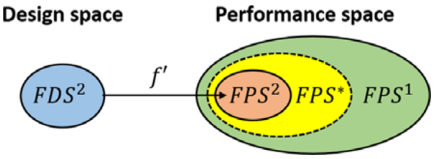

Figure 6. The boundaries of FPS^*.

Trade-off Study. A trade-off needs to be made to decide the distance of FPS^* from FPS^1 and FPS^2. This problem highly resembles the barrier function in optimization. Therefore, we construct an optimization problem based on the barrier function to conduct the trade-off study.

First, we construct the boundaries of the ranges in FPS^*. For $y_j(i)$ (i.e., the jth variable of $y(i)$), let the corresponding range be defined in FPS^* be $R^*_{y_j(i)} = \left[y_j^{l*}(i), y_j^{u*}(i)\right]$. As explained before, the ranges in FPS^* must be bounded by the ranges in FPS^1 and FPS^2. Therefore, the range of $y_j(i)$ in FPS^* is subject to Eq. (21).

$$\begin{cases} y_j^{u2}(i) \le y_j^{u*}(i) \le y_j^{u1}(i) \\ y_j^{l1}(i) \le y_j^{l*}(i) \le y_j^{l2}(i) \end{cases} \tag{21}$$

- $y_j^{u1}(i)$ and $y_j^{l1}(i)$ are the upper and lower bounds of the range of $y_j(i)$ in FPS^1.
- $y_j^{u2}(i)$ and $y_j^{l2}(i)$ are the upper and lower bounds of the range of $y_j(i)$ in FPS^2.

Second, we explain the barrier functions in Eq. (22). $y_j(i)$ is the output of f_i. If $R^*_{y_j(i)}$ is too close to FPS^2, then $h_j(i)$ goes to infinity. If $y_j(i)$ is the input of f_k and $R^*_{y_j(i)}$ is too close to FPS^1, then $h_j^k(i)$ goes to infinity; otherwise, $h_j^k(i) = 0$.

$$\begin{cases} h_j(i) = -a_j(i)\left(\ln|y_j^{u*}(i) - y_j^{u2}(i)| + \ln|y_j^{l*}(i) - y_j^{l2}(i)|\right) \\ h_j^k(i) = -a_j^k(i)\left(\ln|y_j^{u*}(i) - y_j^{u1}(i)| + \ln|y_j^{l*}(i) - y_j^{l1}(i)|\right) \end{cases} \tag{22}$$

The coefficients $a_j(i)$ and $a_j^k(i)$ represent the designers' preference for the ability to tolerate uncertainty, expansion of f_i and the level of difficulty in implementing f_k. Specific values of $a_j(i)$ and $a_j^k(i)$ can only be determined case by case.

$$\min\left(\sum_{i=1}^{n}\sum_{j=1}^{S(i)} h_j(i) + \sum_{i=1}^{n}\sum_{j=1}^{S(i)}\sum_{k=1}^{n} h_j^k(i)\right) \tag{23}$$

Third, the objective function is Eq. (23), where $S(i)$ is the number of the variables in $y(i)$. The optimization problem is to decide $y_j^{u*}(i)$ and $y_j^{l*}(i)$ to minimize Eq. (23), subject to the constraints in Eq. (4) and Eq. (21). As a result, FPS^* can be assembled using the resulting $y_j^{u*}(i)$ and $y_j^{l*}(i)$ where $i = 1, 2, \ldots, n$.

The Sub-Requirements. In general, the sub-requirements must satisfy the following three conditions:

1. They must be within the initial feasible spaces of the sub-functions.

2. They must include all the possible values that the elements of the system will encounter.
3. They must be composable and refine the higher-level functional requirements as a whole.

We intend to assign the ranges in FDS^2 and FPS^* directly to the sub-functions as the sub-requirements. Now, we explain how the ranges satisfy all three conditions above.

First, FDS^1 and FPS^1 are both narrowed from the initial feasible spaces of the sub-functions (see Eq. (15)), and FDS^2 and FPS^* are narrowed from FDS^1 and FPS^1. Therefore, the ranges in FDS^2 and FPS^* satisfy the first condition.

Second, FDS^2 does not narrow the ranges of $\{x\},\{\tilde{x}\},\{u\}$ and $\{\tilde{u}\}$, elements that are associated with $\forall$, thus all the values of the independent variables that the system must accept are included within FDS^2. Moreover, we have established in Eq. (20) that all the possible values of y' are included within FPS^2. Because FPS^* includes FPS^2, all the possible values of y' are also included within FPS^*. Therefore, the ranges in FDS^2 and FPS^* satisfy the second condition.

Third, the ranges of all the elements are only defined once in FDS^2 and FPS^*. If they are assigned to the sub-functions, the identical element will only be assigned the same range. Therefore, the sub-requirements will be composable. In addition, the ranges of $\{x\}$ in FDS^2 are the same as the ranges in the higher-level functional requirements (see Eq. (16)), and the ranges of y are narrowed from the ranges in the higher-level functional requirements (see Eq. (15)). Therefore, the sub-requirements as a whole will also refine the higher-level functional requirement, and the third condition is satisfied.

In conclusion, $R^*_{x(i)}, R^*_{c(i)}$, and $R^*_{u(i)}$ can be directly obtained from FDS^2, and $R^*_{y(i)}$ can be directly obtained from FPS^*. Together, $R^*_{x(i)}, R^*_{c(i)}, R^*_{u(i)}$, and $R^*_{y(i)}$ are the sub-requirements of f_i $(i = 1, 2, \ldots, n)$ decomposed from the R^*_x, R^*_c, R^*_u, and R^*_y of f.

6. Case study

In this section, we design a cruise control system to demonstrate the four-step formal process proposed in Section 5.

Problem description. The task is to decompose functional requirements for a cruise control system. The top-level function can be represented as a mapping below, where the input is the initial speed v_0 and the reference speed v_r; the output is the real speed $v(t)$; no uncertainty parameters are defined. Figure 7 is a graphical representation of the mapping.

$$(v_0, v_r) \xrightarrow{f} v(t)$$

The requirements are summarized in Table 1: for any initial speed $v_0 \in [22.0, 33.0]\,\mathrm{m/s}$ and the reference speed $v_r \in [34.0, 36.0]\,\mathrm{m/s}$, the real speed $v(t)$ shall be bounded by $[20.0, 40.0]\,\mathrm{m/s}$ at all times and converge to $[33.0, 37.0]\,\mathrm{m/s}$ after 20 seconds.

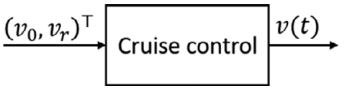

Figure 7. The top-level function.

Table 1. Top-level functional requirements

	Requirements	Meaning
Input (R_x^*)	$v_0 \in [22.0,30.0]\,\mathrm{m/s}$	Given any initial speed in $[22.0,30.0]\,\mathrm{m/s}$
	$v_r \in [34.0,36.0]\,\mathrm{m/s}$	Given any desired speed in $[34.0,36.0]\,\mathrm{m/s}$
Output (R_y^*)	$v(t) \in [20.0,40.0]\,\mathrm{m/s}, \forall t$	The real speed is always bounded, that is, stability
	$v(t) \in [33.0,37.0]\,\mathrm{m/s}, \forall t \in [20,\infty)$	The real speed converges fast enough
Controllable parameters (R_c^*)	Not defined at current level	NA
Uncontrollable parameters (R_u^*)	Not defined at current level	NA

Intuitively, the task of decomposition is breaking down the top-level "box" in Figure 7 into more "boxes" at the lower level (to be explained in Figure 8).

Step 1: Define and analyze the functional architecture. We adopt the functional architecture of a cruise control design from (Aström & Murray 2010) (Figure 8) with each sub-function defined as follows:

$$\begin{cases} f_1 : v(t) = \int_0^t \dot{v}dt + v_0 \\ f_2 : \dot{v}(t) = \dfrac{F(t) - Fr - Fa(t)}{m} \\ f_3 : Fr = mgC_r \\ f_4 : F(t) = \alpha \cdot u(t) T(t) \\ f_5 : Fa(t) = \dfrac{1}{2}\rho C_d A v(t)^2 \\ f_6 : \omega(t) = \alpha \cdot v(t) \\ f_7 : T(t) = T_m \left(1 - \beta\left(\frac{\omega(t)}{\omega_m} - 1\right)^2\right) \\ f_8 : u(t) = p(v_r - v(t)) + i\int_0^t (v_r - v(t))dt \end{cases}$$

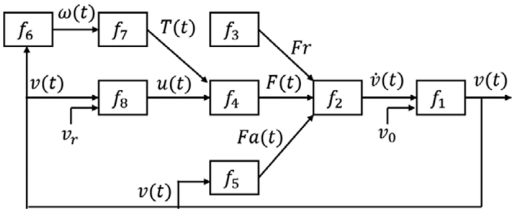

Figure 8. The functional architecture for the top-level functional requirements.

Based on the architecture, we derive the design space and the performance of the cruise control system. For example, according to the characterizations in Section 5.2, $\{y'^{(3)}\}$ are the output variables that are defined in the output of the top-level function, the input of the sub-functions, and the output of the sub-functions. The only variable of such we can find is $v(t)$, and thus $\{y'^{(3)}\} = v(t)$. In addition, as defined in Figure 5, $\{y'^{(1)}\}$ and $\{y'^{(4)}\}$ are the output variables that are not the inputs to any sub-function. In this example, no $\{y'^{(1)}\}$ or $\{y'^{(4)}\}$ is identified. Moreover, as depicted in Figure 5, $\{\tilde{x}\}$ are the variables from the environment but are not defined in the top-level functional requirements. There is no such input variable in the functional architecture in Figure 8. Therefore, $\{\tilde{x}\}$ is empty. Finally, because the uncertain parameters are not defined in the top-level requirements, $\{c\}$ and $\{u\}$ are empty.

- $\{x\} = \{v_r, v_0\}$.
- $\{\tilde{c}\} = \{\omega_m\}$.
- $\{\tilde{u}\} = \{m\}$.
- $\{y'^{(2)}\} = \{\dot{v}(t), Fr(t), F(t), Fa(t), T(t), u(t), \omega(t)\}$.
- $\{y'^{(3)}\} = v(t)$.

As a result, according to Eq. (12), the design space of the cruise control system is,

$$\{v_r, v_0, m, \omega_m\}$$

and the performance space of the cruise control system is,

$$\{\dot{v}(t), Fr(t), F(t), Fa(t), T(t), u(t), \omega(t), v(t)\}.$$

Step 2: Explore the initial feasible spaces. The characterizations of the sub-functions depend on the feasible implementations of the respective sub-function, which can only be determined case by case. For this example, we characterize each sub-function of the functional architecture in Figure 8 as below:

$\mathbf{f_1}$ is a mathematical definition of the speed, and hence will be allocated to no subsystem. $\dot{v}(t)$ has to be bounded to avoid violent acceleration/deceleration by the cruise control; $v(t)$ is bounded to constrain the fastest speed at which the cruise control can be engaged due to safety concerns.

- f_1: $v(t) = \int_0^t \dot{v} dt + v_0$.
- $x(1) = (v_0, \dot{v})^\top$ and,

$$R_{x(1)} = \left([0,40]\text{m/s}, [-1.5,3]\text{m/s}^2\right)^\top.$$

- $y(1) = v(t)$ and $R_{y(1)} = [0,50]\text{m/s}$.

$\mathbf{f_2}$ combines all the forces and yields the acceleration of the car as a whole, where $F(t)$ is the driving force, $Fa(t)$ is the aerodynamic drag and Fr is the tyre friction. The weight of the car m is defined as an uncontrollable parameter.

- f_2: $\dot{v}(t) = (F(t) - Fr - Fa(t))/m$.
- $x(2) = (F(t), Fr, Fa(t))^\top$ and,

$$R_{x(2)} = ([-250,3500]\text{N}, [60,130]\text{N}, [0,1000]\text{N})^\top.$$

- $y(2) = \dot{v}(t)$ and $R_{y(2)} = [-2,4]\text{m/s}^2$;
- $u(2) = m$ and $R_{u(2)} = [990,1100]\text{kg}$.

$\mathbf{f_3}$ represent the rolling friction of the tyre. No input is defined for this function. C_r is the rolling friction coefficient.

- f_3: $Fr = mgC_r$.
- $y(3) = Fr$ and $R_{y(3)} = [70,120]\text{N}$.
- $u(3) = m$ and $R_{u(3)} = [990,1100]\text{kg}$.

$\mathbf{f_4}$ represents the ability of the drivetrain to convert the torque to the driving force, where α is the gear ratio, $u(t)$ is the control input from the cruise controller (not the uncontrollable uncertain parameters defined in this paper), $T(t)$ is the torque.

- f_4: $F(t) = \alpha * u(t)T(t)$.
- $x(4) = (u(t), T(t))^\top$ and,

$$R_{x(4)} = ([-0.5,2],[0,250]\text{N} \cdot \text{m})^\top.$$

- $y(4) = F(t)$ and $R_{y(4)} = [-1250,5000]\text{N}$.

$\mathbf{f_5}$ represents the aerodynamic drag, where ρ is the air density, C_d is the shape-dependent aerodynamic drag coefficient and A is the frontal area of the car.

- f_5: $Fa(t) = \frac{1}{2}\rho C_d A v(t)^2$;
- $x(5) = v(t)$ and $R_{x(5)} = [0,60]\text{m/s}$.
- $y(5) = Fa(t)$ and $R_{y(5)} = [0,2000]\text{N}$.

$\mathbf{f_6}$ represents the ability of the drivetrain to convert from the car speed $v(t)$ to the engine speed $\omega(t)$.

- f_6: $\omega(t) = \alpha * v(t)$;
- $x(6) = v(t)$ and $R_{x(6)} = [0,55]\text{m/s}$.
- $y(6) = \omega(t)$ and $R_{y(6)} = [0,560]\text{rad/s}$.

$\mathbf{f_7}$ represents the correlation between the torque, the engine speed $\omega(t)$, the maximum torque T_m and the maximum engine speed ω_m. ω_m is the controllable uncertain design parameter that starts with a rough estimation but will eventually be determined by the designers.

- f_7: $T(t) = T_m\left(1 - \beta(\omega(t)/\omega_m - 1)^2\right)$.
- $x(7) = \omega(t)$ and $R_{x(7)} = [0,450]\text{rad/s}$.
- $y(7) = T(t)$ and $R_{y(7)} = [0,250]\text{NM}$.
- $c(7) = \omega_m$ and $R_{c(7)} = [350,480]\text{rad/s}$.

$\mathbf{f_8}$ is the cruise controller, in this case, based on PID control. p and i are the proportional and integral parameters for the PID controller.

- f_8: $u(t) = p(v_r - v(t)) + i\int_0^t (v_r - v(t))dt$.
- $x(8) = (v_r, v(t))^\top$ and,

$$R_{x(8)} = ([0,60]\text{m/s},[0,60]\text{m/s})^\top.$$

- $y(8) = u(t)$ and $R_{y(8)} = [-0.5,4]$.

The coefficients mentioned above are $\alpha = 10, g = 9.8, C_r = 0.01, C_d = 0.32, \rho = 1.3, A = 2.4, \beta = 0.4, T_m = 200, p = 0.1, i = 0.5$. Note that these constant coefficients can also be defined as uncertain parameters, just like m and ω_m, which will make the computation more challenging. However, we intentionally limit the number of uncertain parameters to simplify the presentation.

Step 3: Narrow the feasible spaces. First, we compute FDS^1 and FPS^1 by taking intersections of the ranges of the elements defined in multiple places. For example, m is defined in f_2 and f_3, hence R_m is the intersection of the corresponding ranges defined in f_2 and f_3. v_0 is defined in both the top-level function and f_1, hence R_{v_0} is the intersection of the corresponding ranges defined in f_1 and the top-level function.

As a result, the ranges in FDS^1 can be computed as follows:

$$\begin{aligned} R_{v_r} &= [23.0, 30.0]\,\mathrm{m\,s^{-1}}, \\ R_{v_0} &= [34, 36]\,\mathrm{m\,s^{-1}}, \\ R_{\omega_m} &= [350, 480]\,\mathrm{rad\,s^{-1}}, \\ R_m &= [990, 1100]\,\mathrm{kg} \end{aligned}$$

The ranges in FPS^1 can be computed as follows:

$$\begin{aligned} R_{\dot{v}(t)} &= [-1.5, 3]\,\mathrm{m\,s^{-2}}, \\ R_{Fr(t)} &= [70, 120]\,\mathrm{N}, \\ R_{F(t)} &= [-250, 3500]\,\mathrm{N}, \\ R_{Fa(t)} &= [0, 1000]\,\mathrm{N}, \\ R_{T(t)} &= [0, 250]\,\mathrm{Nm}, \\ R_{u(t)} &= [-0.5, 2], \\ R_{\omega(t)} &= [0, 450]\,\mathrm{rad\,s^{-1}}, \\ R_{v(t)} &= [20, 40]\,\mathrm{m\,s^{-1}} \end{aligned}$$

Second, let $f' = f_1 \wedge f_2 \wedge \ldots \wedge f_8$. FDS^1 and FPS^1 can be further narrowed into FDS^2 and FPS^2 using f'. To achieve FDS^2, the range of the controllable parameter ω_m must be narrowed into $R^{\downarrow}_{\omega_m}$ based on Eq. (19) such that,

$$\text{For } \forall v_r \in R_{v_r}, \forall v_0 \in R_{v_0}, \forall m \in R_m, \forall \omega_m \in R^{\downarrow}_{\omega_m} : (v_r, v_0, \omega_m, m) \xrightarrow{f'} y' \in FPS^1$$

where $y' = (v(t), \dot{v}(t), Fr(t), F(t), Fa(t), T(t), u(t), \omega(t))^{\top}$. In this example, we apply reachability analysis to help find $R^{\downarrow}_{\omega_m}$. We estimate an over-approximation of $R^{\downarrow}_{\omega_m}$, feed it into a reachability solver (i.e., CORA (Althoff 2015) in this example), and adjust the bounds of the over-approximation until the reachable sets of all the output variables are bounded within FPS^1.

As a result, we obtain the ranges in FDS^2 as follows:

$$\begin{aligned} R_{v_r} &= [23.0, 30.0]\,\mathrm{m\,s^{-1}}, \\ R_{v_0} &= [34, 36]\,\mathrm{m\,s^{-1}}, \\ R^{\downarrow}_{\omega_m} &= [365, 450]\,\mathrm{rad\,s^{-1}}, \\ R_m &= [990, 1100]\,\mathrm{kg} \end{aligned}$$

After that, FPS^1 can be narrowed into FPS^2 by calculating the reachable sets according to Eq. (20). We choose CORA as the tool because it can compute the over-approximation of the feasible performance space. The result shows the range of $v(t)$ converges gradually from R_{v_0} to R_{v_r}, meaning the reference speed is achieved by the cruise control system. Eventually, we obtain the ranges of FPS^2:

$$
\begin{aligned}
&R^{\downarrow}_{\dot{v}(t)} = [-0.63, 2.89]\,\mathrm{m\,s^{-2}},\\
&R^{\downarrow}_{Fr(t)} = [88.20, 107.80]\,\mathrm{N},\\
&R^{\downarrow}_{F(t)} = [-9.33, 2997.78]\,\mathrm{N},\\
&R^{\downarrow}_{Fa(t)} = [240.88, 655.25]\,\mathrm{N},\\
&R^{\downarrow}_{T(t)} = [179.94, 200.00]\,\mathrm{Nm},\\
&R^{\downarrow}_{u(t)} = [0, 1.51],\\
&R^{\downarrow}_{\omega(t)} = [219.66, 362.30]\,\mathrm{rad\,s^{-1}},\\
&R^{\downarrow}_{v(t)} = [21.97, 36.23]\,\mathrm{m\,s^{-1}}\ \text{for}\ [0, 100]\,\mathrm{s},\\
&R^{\downarrow}_{v(t)} = [33.65, 36.19]\,\mathrm{m\,s^{-1}}\ \text{for}\ [20, 100]\,\mathrm{s}
\end{aligned}
$$

All the ranges in FPS^2 are subsets of their counterparts in FPS^1. In other words, FPS^2 is successfully narrowed from FPS^1.

Step 4: Determine the sub-requirements. We compute $R^*_{y(i)} = [y^{l*}(i), y^{u*}(i)]$ for $y(i)$ $(i = 1, 2, \ldots, 8)$ by formulating the optimization problem in Eq. (23) subject to the constraints in Eq. (4) and Eq. (21). Note that there is no subscript j associated with the output variable $y(i)$, because each f_i in this example only has one variable at its output side.

First, the constraints described in Eq. (21) can be directly obtained from FDS^1, FPS^1, FDS^2 and FPS^2 calculated in the previous step.

Second, Eq. (4) needs to be satisfied for each individual sub-function, which yields the constraints in Eq. (24). Note that f_1 and f_8 are not included in the constraints. f_1 and f_8 alone are not enough to decide the ranges of the output variables because the operation $\int$ makes the ranges of the output variables correlated with the time t. Their ranges can only be determined after the rest of the system is determined. In this example, f_1 and f_8 belong to the PID controller. These modules are designed after other parts of the car (e.g., engine and tires) are defined, and hence are not part of the constraints in Eq. (24).

$$
\begin{cases}
y^{l*}(6) \le f_6(y^{l*}(1)) \\
y^{u*}(6) \ge f_6(y^{u*}(1)) \\
y^{l*}(7) \le f_7(y^{l*}(6), \overline{\omega_m}) \\
y^{u*}(7) \ge f_7(y^{u*}(6), \underline{\omega_m}) \\
y^{l*}(5) \le f_5(y^{l*}(1)) \\
y^{u*}(5) \ge f_5(y^{u*}(1)) \\
y^{l*}(4) \le f_4(y^{l*}(7), y^{l*}(8)) \\
y^{u*}(4) \ge f_4(y^{u*}(7), y^{u*}(8)) \\
y^{l*}(2) \le f_2(y^{l*}(4), y^{u*}(3), y^{u*}(5), \overline{m}) \\
y^{u*}(2) \ge f_2(y^{u*}(4), y^{l*}(3), y^{l*}(5), \underline{m})
\end{cases}
\tag{24}
$$

Third, based on Eq. (22) and Eq. (23), we define the objective function below:

$$\min\left(\sum_{i=1}^{8} h(i) + \sum_{i=1}^{8}\sum_{k=1}^{8} h^k(i)\right) \tag{25}$$

where the coefficients are defined as $a(1)=0.9, a(2)=0.6, a(3)=0.5, a(4)=0.3, a(5)=0.4, a(6)=0.5, a(7)=0.1, a(8)=0.9, a^5(1)=0.5, a^6(1)=0.1, a^8(1)=0.8, a^1(2)=0.6, a^2(3)=0.4, a^2(4)=0.8, a^2(5)=0.7, a^7(6)=0.6, a^4(7)=0.9, a^4(8)=0.9$.

Finally, the optimization problem is to decide $y^{u*}(i)$ and $y^{l*}(i)$ $(i=1,2,\ldots,8)$ by minimizing Eq. (25). As a result, $y^{u*}(i)$ and $y^{l*}(i)$ $(i=1,2,\ldots,8)$ are calculated. Together with FDS^2 computed in the previous step, the ranges of each sub-function can be summarized in Table 2. Accordingly, the sub-requirements of f_i are the mapping between $\left(R^*_{x(i)}, R^*_{c(i)}, R^*_{u(i)}\right)$ and $R^*_{y(i)}$ in Table 2.

Figure 9 is a graphical representation of the results. The top-level function f is decomposed into eight sub-functions $f_1, f_2, \ldots, f_8$. As a whole, the eight functions have the same input variables and output variables as the top-level function. Other input variables of the sub-functions can be found at the outputs of the sub-functions, and vice versa. As we can see in Table 2, the variables that appear at both the input side and the output side have the same ranges, meaning the sub-functions are *composable*. Furthermore, the sub-functions as a whole *refine* f because the input sets of the former include no less elements than the latter and the output set includes no more elements than the former. Based on our findings of parallel development in Section 4, these sub-functions can be implemented independently. The same process can be repeated to further decompose the sub-functions at even lower levels, which eventually will lead to a system that satisfies the top-level functional requirements.

Table 2. The requirements of the sub-functions

f	$R^*_{x(i)}, R^*_{c(i)}, R^*_{u(i)}$	$R^*_{y(i)}$
f_1	$v_0 \in [34,36]\text{m/s}$ $\dot{v}(t) \in [-1.1,3]\text{m/s}^2$	$v(t) \in [21.8,38.4]\text{m/s}$
f_2	$Fr(t) \in [88.2,107.8]\text{N}$ $F(t) \in [-159.1,3024.0]\text{N}$ $Fa(t) \in [237.6,827.6]\text{N}$ $m \in [990,1100]\text{kg}\backslash$	$\dot{v}(t) \in [-1.1,3]\text{m/s}^2$
f_3	$m \in [990,1100]\text{kg}$	$Fr(t) \in [88.2,107.8]\text{N}$
f_4	$T(t) \in [150,200.1]\text{Nm}$ $u(t) \in [-0.1,1.511]$	$F(t) \in [-159.1,3024.0]\text{N}$
f_5	$v(t) \in [21.8,38.4]\text{m/s}$	$Fa(t) \in [237.6,827.6]\text{N}$
f_6	$v(t) \in [21.8,38.4]\text{m/s}$	$\omega(t) \in [109.8,406.1]\text{rad/s}$
f_7	$\omega(t) \in [109.8,406.1]\text{rad/s}$ $\omega_m \in [365,450]\text{rad/s}$	$T(t) \in [150,200.1]\text{Nm}$
f_8	$v_r \in [23.0,30.0]\text{m/s}$ $v(t) \in [21.8,38.4]\text{m/s}$	$u(t) \in [-0.1,1.511]$

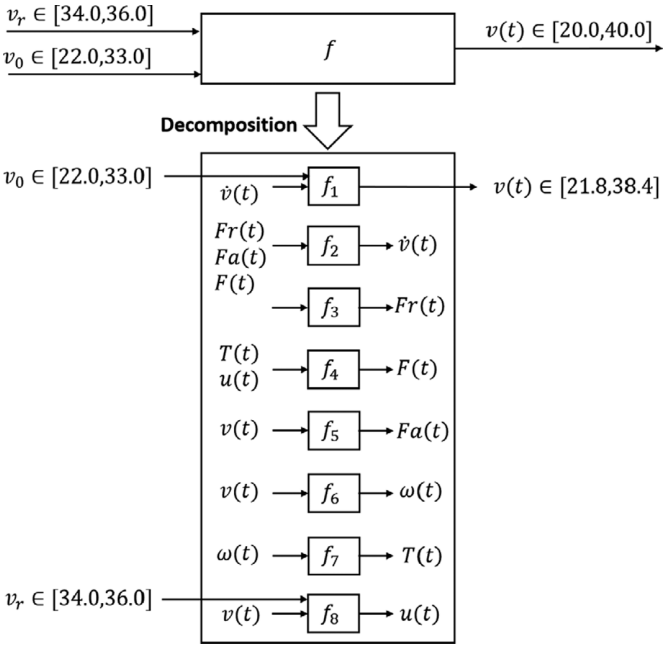

Figure 9. A graphical representation of the results.

7. Conclusion

7.1. Summary

Functional requirements decomposition at the early stage is challenging due to the uncertainty and ambiguity in design and the loose inter-team coordination, especially in large projects for complex systems (e.g., commercial airlines). SBD has demonstrated excellent potential in addressing these challenges, but with limited formal guidance. To tackle uncertainty and ambiguity, one of the most effective approaches is to formalize. SBD's claim for team autonomy lacks formal proof as well. For this reason, this paper formalizes the functional requirements decomposition process based on SBD principles. We first prove parallel development: as long as the sub-requirements are composable and refine the higher-level functional requirements, the design teams at multiple levels can decompose their functional requirements independently, and the final system will still satisfy the top-level functional requirements. After that, we propose a four-step formal process that supports parallel development to decompose the functional requirements based on SBD principles. Finally, a case study on designing a cruise control system was conducted to demonstrate the effectiveness of the proposed approach.

7.2. Discussion

Toyota's process. According to Specking *et al.*, much of the SBD literature shares similar characteristics with Toyota's process (Specking *et al.* 2018a). We compare

our process with Toyota's approach below. In Toyota's approach (Ward *et al.* 1995):

1. The team defines a set of solutions at the system level.
2. It defines sets of possible solutions for various subsystems.
3. It explores these possible subsystems in parallel to characterize a possible set of solutions.
4. It gradually narrows the set of solutions, converging slowly toward a single solution. In particular, the team determines the appropriate specifications to impose on the subsystems.
5. Once the team establishes the single solution for any part of the design, it does not change it unless absolutely necessary.

The first step above is the same as Step 1 in our process, where the "solution" in our case is the functional architecture. The second and third steps above correspond to Step 2 of our process, which essentially involves defining the feasible implementation of the subsystems individually. The fourth step above corresponds to Steps 3 and 4 in our process. However, our formal process provides more details without ambiguity. With the underlying formalism defined in the process, we can explain how to narrow the sets, determine the specifications on the subsystems, and gradually converge to a single solution using hierarchy. The fifth step above is related to the uncertainty expansion addressed in our process. Specifically, Toyota applies "rigorous effort to avoid changes that *expand*, rather than contract, the space of possible designs" (Ward *et al.* 1995). We believe expansion is sometimes unavoidable due to epistemic uncertainties. Step 4 of our process for "uncertainty expansion" is motivated by this reason and designed to address the possible impacts of unexpected uncertainty expansions.

Implication for design. Requirements decomposition is not a stand-alone activity in design. It is inherently multidisciplinary, integrating principles from systems engineering, software development, human factors and organizational behavior. Our approach makes positive implications for the overall design by addressing the following four concerns while the design is progressing to one level lower: (1) the sub-requirements must not be technically too demanding, making them infeasible or too expensive to implement, (2) the sub-requirements must not be too limited, failing to support the intended system-level function and performance, (3) the sub-requirements must not be too relaxed, wasting too much capability of the subsystems and (4) the sub-requirements should not be too tight, leaving little room for uncertainty. Traditionally, addressing these concerns requires intense communication efforts and conflict resolution between various stakeholders. Our approach solves them with a formal framework, increasing communication efficiency, directing conflict resolution and ensuring successful integration with mathematical rigor. However, it is worth noting that formalism is by no means a silver bullet for solve all the challenges in requirements decomposition. Our approach should be blended with other activities (e.g., requirements negotiation and change management) and methods (e.g., optimization and reachability analysis) to eventually achieve a more effective and efficient requirements decomposition process.

Scope. Our approach enhances requirements decomposition, but does not solve all the challenges in requirements decomposition. First, requirements decomposition entails much more equally important topics than those presented in this

paper, such as requirements elicitation, architecture definition and requirements validation and verification. This paper sets itself apart by focusing on addressing the four concerns discussed above. In the future, we will seek to understand how to integrate our approach with other equally important topics to further advance the research on requirements decomposition. Second, the proposed four-step process is a framework that focuses on articulating *what* needs to be solved rather than *how* to solve them. Multiple computational problems (e.g., optimization) are identified and formally defined without specifying the methods and tools to solve them and prescribing how well they need to be solved. Such ambiguity is left on purpose because (1) these computational problems on their own are still very active research topics, and (2) we intend to leave the most autonomy to the design teams to decide how to solve these problems with what tools. For this reason, we intentionally downplay the computational analysis (e.g., complexity analysis) in the paper as they are not directly relevant to addressing the four concerns discussed above and depend on the specific methods and tools selected by the engineers. We *do not* claim a contribution to the computational aspect of the proposed process. Finally, formalism means precision but does not remove engineer's judgment out of the equation. Formalism in our framework instills precision in the decomposition process and reduces communication efforts by eliminating conflicts with the formally defined procedural requirements in place. But engineer's judgment are still required as input. For example, the coefficients of the barrier function in Eq. (22) reflect engineer's tolerance for uncertainty expansion and their evaluation of the level of difficulty in implementing the sub-requirements. Determining these coefficients is highly subjective, same as many other optimization problems that involve human subjective inputs. We do not study how these coefficients are determined in this paper, but will explore the underlying decision-making mechanism in the future.

Managing complexity. A complex system usually has hundreds, if not thousands, of functions. Such a level of complexity is a main challenge for designing modern systems. As a solution, abstraction is one of the most effective tools to manage complexity (Hill *et al.* 2008; Bencomo *et al.* 2024). The activity of decomposition, on one hand, refines the higher-level abstraction, and on the other hand, creates abstractions for the lower-level system. The reason for applying abstraction is that there is a limit to human cognitive manageability. It is recommended by multiple systems engineering handbooks (Kapurch 2010; INCOSE 2023) that decomposition should maintain a reasonable number of functions per level. This number is generally aligned with the famous Miller's law (Miller 1956: 5–9) functions per level, which is also the choice of the cruise control case study in this paper. Furthermore, the proposed process is agnostic to the number of sub-functions abstracted per level. The sub-functions can be further decomposed into multiple levels of smaller components. In fact, the deep hierarchical abstraction of the complex system makes the SBD-based method a particularly suitable approach to requirements decomposition.

Margin. As mentioned before, many concepts incorporated in this paper are closely related to margin. First, in Eq. (2), u denotes the uncertain parameters that cannot be controlled with R_u^* as a quantification of the uncertainty. While the uncertainty might be reduced due to, for example, a better understanding of the system and the environment, $\forall u \in R_u^*$ instills a margin (more precisely, buffer) into the system design at the current level that no matter whether and how the

uncertainty is reduced, the uncertainty can always be absorbed by the design. Second, the $\in$ in Eq. (2) ensures that all the output values generated from the inputs and parameters can be successfully processed by the system, creating a margin (more precisely, excess) for the system output. Third, c and $\tilde{c}$ of Eq. (19) are controllable parameters, which are associated with $\exists$ in the original system formulation in Eq. (2). But the narrowed bounds $R_c^{\downarrow}$ and $R_{\tilde{c}}^{\downarrow}$ in Eq. (19) are associated with $\forall$. This is because c and $\tilde{c}$ will be further narrowed at the lower level of abstraction. $\forall$ provides a margin (more precisely, buffer) for the lower levels, so that no matter how c and $\tilde{c}$ are decided at a later time, it will never cause a violation at the higher level. Fourth, FPS^2 is computed in Eq. (20) as an over-approximation of FDS^2 due to technical difficulties in calculating the precise space of y'. Such an over-approximation formulates a margin (more precisely, excess) in FPS^2 w.r.t. FDS^2. Fifth and finally, uncertainty expansion in Section 5.5 expands the performance space from FPS^2 to FPS^* introducing a margin (more precisely, excess) for a potential growth of uncertainty.

Future work. Our approach is only applicable to the systems that can be abstracted as input–output transformations (e.g., dynamic systems), while general SBD does not assume any formalism of the system. This is a limitation of the paper. The applicability of other formalisms within the framework should be investigated in the future.

References

Al-Ashaab, A., **Howell, S.**, **Usowicz, K.**, **Anta, P. H.** & **Gorka, A.** 2009 Set-based concurrent engineering model for automotive electronic/software systems development. In *Proceedings of the 19th CIRP Design Conference – Competitive Design,* pp. 464–468.

Al-Ashaab, A., **Golob, M.**, **Attia, U. M.**, **Khan, M.**, **Parsons, J.**, **Andino, A.**, **Perez, A.**, **Guzman, P.**, **Onecha, A.**, **Kesavamoorthy, S.**, **Martinez, G.**, **Shehab, E.**, **Berkes, A.**, **Haque, B.**, **Soril, M.** & **Sopelana, A.** 2013a The transformation of product development process into lean environment using set-based concurrent engineering: a case study from an aerospace industry. *Concurrent Engineering*; doi:10.1177/1063293X13495220.

Al-Ashaab, A., **Golob, M.**, **Noriega, P.**, **Torriani, F.**, **Alvarez, P.**, **Beltran, A.**, **Busachi, A.**, **Ex-Ignotis, L.**, **Rigatti, C.** & **Sharma, S.** 2013b Capturing the industrial requirements of set-based design for CONGA framework. In *Proceedings of the 11th International Conference on Manufacturing Research (ICMR2013)*, pp. 3–83.

Al Handawi, K., **Andersson, P.**, **Panarotto, M.**, **Isaksson, O.** & **Kokkolaras, M.** 2021 Scalable set-based design optimization and remanufacturing for meeting changing requirements. *Journal of Mechanical Design*, **143** (2), 021702.4. doi:10.1115/1.4047908.

Al Handawi, K., **Brahma, A.**, **Wynn, D. C.**, **Kokkolaras, M.** & **Isaksson, O.** 2024 Design space exploration and evaluation using margin-based trade-offs. *Journal of Mechanical Design* **146** (6), 061701.

Althoff, M. 2015 An introduction to CORA 2015. *ARCH*; doi:10.29007/zbkv.

Ammar, R., **Hammadi, M.**, **Choley, J.-Y.**, **Barkallah, M.**, **Louat, J.** & **Haddar, M.** 2017 Architectural design of complex systems using set-based concurrent engineering. *ISSE*; doi:10.1109/SysEng.2017.8088290.

Aström, K. J. & **Murray, R. M.** 2010 *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press.

Bencomo, N., **Cabot, J.**, **Chechik, M.**, **Cheng, B. H. C.**, **Combemale, B.**, & **Zschaler, S.** *et al.* 2024 Abstraction engineering. *Preprint*, arXiv:2408.14074.

Benveniste, A., **Caillaud, B.**, **Nickovic, D.**, **Passerone, R.**, **Raclet, J.-B.**, **Reinkemeier, P.**, **Sangiovanni-Vincentelli, A.**, **Damm, W.**, **Henzinger, T.**, & **Larsen, K. G.** 2015 Contracts for systems design: Theory. [Research Report] RR-8759, Inria Rennes Bretagne Atlantique; INRIA. 2015, p. 86.

Bogus, S. M., **Molenaar, K. R.** & **Diekmann, J. E.** 2006 Strategies for overlapping dependent design activities. *Construction Management and Economics*; doi: 10.1080/01446190600658529.

Booton, R. C. & **Ramo, S.** 2008 The development of systems engineering. *IEEE Transactions on Aerospace and Electronic Systems* (4), 306–310.

Borchani, M. F., **Hammadi, M.**, **Yahia, N. B.** & **Choley, J.-Y.** 2019 Integrating model-based system engineering with set-based concurrent engineering principles for reliability and manufacturability analysis of mechatronic products. *Concurrent Engineering*; doi: 10.1177/1063293X18816746.

Brahma, A. & **Wynn, D. C.** 2020 Margin value method for engineering design improvement. *Research in Engineering Design* **31** (3), 353–381.

Brahma, A., **Wynn, D. C.** & **Isaksson, O.** 2022 Use of margin to absorb variation in design specifications: an analysis using the margin value method. *Proceedings of the Design Society* **2**, 323–332.

Brahma, A., **Ferguson, S.**, **Eckert, C.** & **Isaksson, O.** 2024 Margins in design–review of related concepts and methods. *Journal of Engineering Design* **35** (10), 1193–1226.

Buede, D. M. 1997 Integrating requirements development and decision analysis. *SMC*; doi: 10.1109/ICSMC.1997.638223.

Castañeda, K., **Herrera, R. F.**, **Sánchez, O.** & **Mejía, G.** 2023 Set-based design in construction projects: benefits, difficulties and trends. In *Proceedings of the 31st Annual Conference of the International Group for Lean Construction, Lille*, Vol. **31**, pp. 1092–1103. IGLC.

Clarkson, P. J., **Simons, C.** & **Eckert, C.** 2004 Predicting change propagation in complex design. *Journal of Mechanical Design* **126** (5), 788–797.

Costa, R. & **Sobek, D. K.** 2003 Iteration in engineering design: inherent and unavoidable or product of choices made? In *DETC*; doi:10.1115/DETC2003/DTM-48662.

Dullen, S., **Verma, D.**, **Blackburn, M.** & **Whitcomb, C.** 2021 Survey on set-based design (SBD) quantitative methods. *Systems Engineering*; doi:10.1002/sys.21580.

Eckert, C., **Isaksson, O.** & **Earl, C.** 2019 Design margins: a hidden issue in industry. *Design Science*; doi:10.1017/dsj.2019.7.

Claudia Eckert, P. J. C. & **Zanker, W.** 2004 Change and customisation in complex engineering domains. *Research in Engineering Design* **15** (1), 1–21.

Eisner, H. 2011 *Essentials of Project and Systems Engineering Management*. John Wiley & Sons.

El Fassi, S., **Guenov, M. D.** & **Riaz, A.** 2020 An assumption network-based approach to support margin allocation and management. In *Proceedings of the Design Society: DESIGN Conference*, Vol. **1**, pp. 2275–2284. Cambridge University Press.

Ferguson, S., **Brahma, A.**, **Eckert, C.** & **Isaksson, O.** 2024 Managing and modelling margins in design as a means for confronting the challenges of a disruptive world. *Journal of Engineering Design* **35** (10), 1185–1192.

Fleming, C. H. & **Leveson, N.** 2015 Integrating systems safety into systems engineering during concept development. In *INCOSE International Symposium*, Vol. **25**, pp. 989–1003. Wiley Online Library.

Gamage, D. S. K. & **Sobek II., D. K.** 2024 A set-based knowledge generation framework for product development. In *Proceedings of the International Annual Conference of the*

American Society for Engineering Management, pp. 1–8. American Society for Engineering Management (ASEM).

Ghosh, S. & **Seering, W.** 2014 Set-based thinking in the engineering design community and beyond. In *DETC*; doi:10.1115/DETC2014-35597.

Hill, J. H., **Houle, B. J.**, **Merritt, S. M.** & **Stix, A.** 2008 Applying abstraction to master complexity. In *Proceedings of the 2nd International Workshop on the Role of Abstraction in Software Engineering*, pp. 15–21.

INCOSE 2023 *INCOSE Systems Engineering Handbook*. John Wiley & Sons.

Ishikawa, H. 2024 Comparison of design effort across cae simulation, surrogate, and set-based models. *The Proceedings of Design & Systems Conference* **34**, 3103; doi:10.1299/jsmedsd.2024.34.3103.

Jacobson, L. & **Ferguson, S.** 2023 A hierarchical exploration of how design margins enable adaptability. *Proceedings of the Design Society* **3**, 191–200.

Jansson, G., **Schade, J.** & **Olofsson, T.** 2013 Requirements management for the design of energy efficient buildings. *Journal of Information Technology in Construction* **18**, 321–337.

Juul-Nyholm, H. K. & **Eifler, T.** 2024 Multi-objective robustness indicators for evaluation and exploration of design margins. *Journal of Engineering Design* **35** (10), 1227–1257.

Kapurch, S. J. 2010 *NASA Systems Engineering Handbook*. Diane Publishing.

Kerga, E., **Rossi, M.**, **Taisch, M.** & **Terzi, S.** 2014 A serious game for introducing set-based concurrent engineering in industrial practices. *Concurrent Engineering*; doi: 10.1177/1063293X1455010.

Kizer, J. R. & **Mavris, D. N.** 2014 Set-based design space exploration enabled by dynamic constraint analysis. In *Proceedings of the 29th Congress of the International Council of the Aeronautical Sciences*, ICAS.

Komoto, H. & **Tomiyama, T**. 2011 A theory of decomposition in system architecting. In *DS 68–2: Proceedings of the 18th International Conference on Engineering Design (ICED 11), Impacting Society through Engineering Design, Vol. 2: Design Theory and Research Methodology*, 15–19.08.2011.

Kriesi, C., **Blindheim, J.**, **Bjelland, Ø.** & **Steinert, M.** 2016 Creating dynamic requirements through iteratively prototyping critical functionalities. *Procedia CIRP*; doi:10.1016/j.procir.2016.04.122.

Landahl, J., **Jiao, R. J.**, **Madrid, J.**, **Söderberg, R.** & **Johannesson, H.** 2021 Dynamic platform modeling for concurrent product-production reconfiguration. *Concurrent Engineering*; doi:10.1177/1063293X20958938.

Larson, B. J. & **Mattson, C. A.** 2012 Design space exploration for quantifying a system model's feasible domain. *Journal of Mechanical Design*; doi:10.1115/1.4005861.

Levandowski, C., **Forslund, A.**, **Johannesson, H.**, *et al.* 2013 *Using PLM and Trade-off Curves to Support Set-Based Convergence of Product Platforms*. ICED.

Levandowski, C., **Raudberget, D.** & **Johannesson, H.** 2014 Set-based concurrent engineering for early phases in platform development. In *Moving Integrated Product Development to Service Clouds in the Global Economy*. IOS Press; doi: 10.3233/978-1-61499-440-4-564.

Levandowski, C., **Müller, J. R.** & **Isaksson, O.** 2016 Modularization in concept development using functional modeling. In *Advances in Transdisciplinary Engineering*. IOS Press; doi:10.3233/978-1-61499-703-0-117.

Leveson, N. G. 2016 *Engineering a Safer World: Systems Thinking Applied to Safety*. The MIT Press.

Leveson, N. G. & **Turner, C. S.** 1993 An investigation of the therac-25 accidents. *Computer* **26** (7), 18–41.

Madhavan, K., **Shahan, D.**, **Seepersad, C. C.**, **Hlavinka, D. A.** & **Benson, W.** 2008 *An Industrial Trial of a Set-Based Approach to Collaborative Design*. DETC; doi:10.1115/DETC2008-49953.

Majerus, N. 2017 *Lean-Driven Innovation: Powering Product Development at the Goodyear Tire & Rubber Company*. CRC Press.

Matthews, J., **Klatt, T.**, **Seepersad, C. C.**, **Haberman, M.** & **Shahan, D.** 2014 *Bayesian Network Classifiers and Design Flexibility Metrics for Set-Based, Multiscale Design with Materials Design Applications*. DETC; doi:10.1115/DETC2014-34436.

Matthews, J., **Klatt, T.**, **Morris, C.**, **Seepersad, C. C.**, **Haberman, M.** & **Shahan, D.** 2016 Hierarchical design of negative stiffness metamaterials using a Bayesian network classifier. *Journal of Mechanical Design*; doi:10.1115/1.4032774.

McKenney, T. A., **Kemink, L. F.** & **Singer, D. J.** 2011 Adapting to changes in design requirements using set-based design. *Naval Engineers Journal*; doi:10.1111/j.1559-3584.2011.00331.x.

Miller, G. A. 1956 The magical number seven, plus or minus two: some limits on our capacity for processing information. *Psychological Review* **63** (2), 81.

Morgan, J. & **Liker, J. K.** 2020 *The Toyota Product Development System: Integrating People, Process, and Technology*. Productivity Press.

Müller, J. R., **Isaksson, O.**, **Landahl, J.**, **Raja, V.**, **Panarotto, M.**, **Levandowski, C.** & **Raudberget, D.** 2019 Enhanced function-means modeling supporting design space exploration. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*; doi:10.1017/S0890060419000271.

Müller, J. R., **Siiskonen, M. D. I.** & **Malmqvist, J.** 2020 Lessons learned from the application of enhanced function-means modelling. *Design*; doi:10.1017/dsd.2020.87.

Nahm, Y.-E. & **Ishikawa, H.** 2006 Novel space-based design methodology for preliminary engineering design. *The International Journal of Advanced Manufacturing Technology*; doi:10.1007/s00170-004-2463-2.

de Oliveira, M. S., **Forcellini, F. A.**, **Lozano, J. A.** & **Barbosa, J. R.** 2024 Review of models and frameworks for set-based design. *International Journal of Product Development* **28** (1–2), 73–103.

Oyama, S. & **Kato, T**. 2024 A preference set-based design method for qualitative evaluation a case study on automotive EC data. *The Proceedings of Design & Systems Conference* **34** 1310; doi:10.1299/jsmedsd.2024.34.1310.

Panarotto, M. & **Fernández, I. A.** 2024 Local absorption of uncertainty in complex systems using resilient objects. *Journal of Engineering Design* **35** (10), 1292–1310.

Panchal, J. H., **Fernández, M. G.**, **Paredis, C. J. J.**, **Allen, J. K.** & **Mistree, F.** 2007 An interval-based constraint satisfaction (IBCS) method for decentralized, collaborative multifunctional design. *Concurrent Engineering*; doi:10.1177/1063293X07083083.

Parker, M., **Garner, M.**, **Arcano, J.** & **Doerry, N.** 2017 Set-based requirements, technology, and design development for SSN (X). In *Proceedings of Warship 2017: Naval Submarines and UUVs*, 14–15 June 2017.

Parnell, G. S., **Specking, E.**, **Goerger, S.**, **Cilli, M.** & **Pohl, E.** 2019 Using set-based design to inform system requirements and evaluate design decisions. *INCOSE International Symposium* **29** (1), 371–383. doi:10.1002/j.2334-5837.2019.00609.x.

Parsons, M. G. & **Singer, D. J.** 1999 *A Hybrid Agent Approach for Set-Based Conceptual Ship Design*. ICCAS.

Penzenstadler B. 2011 *Desyre-decomposition of systems and their requirements.* PhD thesis, Technische Universität München.

Qureshi, A. J., **Dantan, J.-Y.**, **Bruyere, J.** & **Bigot, R.** 2014 Set-based design of mechanical systems with design robustness integrated. *International Journal of Product Development*; doi:10.1504/IJPD.2014.060037.

Rapp, S., **Chinnam, R.**, **Doerry, N.**, **Murat, A.** & **Witus, G.** 2018 Product development resilience through set-based design. *Systems Engineering*; doi:10.1002/sys.21449.

Raudberget, D. S., **Michaelis, M. T.** & **Johannesson, H. L.** 2014 *Combining Set-Based Concurrent Engineering and Function–Means Modelling to Manage Platform-Based Product Family Design*. IEEM; doi:10.1109/IEEM.2014.7058668.

Rismiller, S., **Cagan, J.** & **McComb, C.** 2023 Exploring the impact of set-based concurrent engineering through multi-agent system simulation. *AI EDAM* **37**, e16.

Rismiller, S., **Cagan, J.** & **McComb, C.** 2024 Understanding collaboration in sub-structured teams through a computational model of set-based concurrent engineering. *Journal of Engineering Design* **35** (5), 543–569.

Rismiller, S. C. 2023 *Using multi agent systems to computationally study set-based concurrent engineering and its interactions with team organization and problem structure*. PhD thesis, Carnegie Mellon University.

Ross, J. E. 2017 *Determining feasibility resilience: set based design iteration evaluation through permutation stability analysis*. PhD thesis, The University of Southern Mississippi.

Schrader, S., **Riggs, W. M.** & **Smith, R. P.** 1993 Choice over uncertainty and ambiguity in technical problem solving. *Journal of Engineering and Technology Management*; doi:10.1016/0923-4748(93)90059-R.

SEBoK 2025 Guide to the systems engineering body of knowledge (sebok) — sebok. https://sebokwiki.org/w/index.php?title=Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK)&oldid=75598.

Shahan, D. W. & **Seepersad, C. C.** 2012 Bayesian network classifiers for set-based collaborative design. *Journal of Mechanical Design*; doi:10.1115/1.4006323.

Shallcross, N., **Parnell, G. S.**, **Pohl, E.** & **Specking, E.** 2020 Set-based design: the state-of-practice and research opportunities. *Systems Engineering*; doi:10.1002/sys.21549.

Shallcross, N. J. 2021 *Quantitative set-based design for complex system development*. PhD thesis, University of Arkansas.

She, J., **Belanger, E.**, **Bartels, C.** & **Reeling, H.** 2022 Improve syntax correctness and breadth of design space exploration in functional analysis. *Journal of Mechanical Design*; doi:10.1115/1.4054875.

Sobek, D. K., **Ward, A. C.** & **Liker, J. K.** 1999 Toyota's principles of set-based concurrent engineering. *MIT Sloan Management Review* **40** (2), 67–83.

Specking, E., **Shallcross, N.**, **Parnell, G. S.** & **Pohl, E.** 2021 Quantitative set-based design to inform design teams. *Applied Sciences*; doi:10.3390/app11031239.

Specking, E. A., **Whitcomb, C.**, **Parnell, G. S.**, **Goerger, S. R.**, **Pohl, E.** & **Kundeti, N. S. A.** 2018a Literature review: exploring the role of set-based design in trade-off analytics. *Naval Engineers Journal* **130** (2), 51–62.

Specking, E., **Parnell, G.**, **Pohl, E.** & **Buchanan, R.** 2018b Early design space exploration with model-based system engineering and set-based design. *Systems* **6** (4), 45.

Tackett, M. W. P., **Mattson, C. A.** & **Ferguson, S. M.** 2014 A model for quantifying system evolvability based on excess and capacity. *Journal of Mechanical Design*; doi:10.1115/1.4026648.

Kerga, E. T., **Taisch, M.** & **Terzi, S.** 2013 Set based concurrent engineering innovation roadmap. In *IFIP International Conference on Advances in Production Management Systems*, pp. 253–261. Springer.

Terwiesch, C., **Loch, C. H.** & **Meyer, A. D.** 2002 Exchanging preliminary information in concurrent engineering: alternative coordination strategies. *Organization Science*; doi: 10.1287/orsc.13.4.402.2948.

Toche, B., **Pellerin, R.** & **Fortin, C.** 2020 Set-based design: a review and new directions. *Design Science*; doi:10.1017/dsj.2020.16.

Trueworthy, A. M., **DuPont, B. L.**, **Maurer, B. D.** & **Cavagnaro, R. J.** 2019 A set-based design approach for the design of high-performance wave energy converters. In *Proceedings of the 13th European Tidal and Wave Energy Conference*, Naples, Italy, pp. 1–6.

Kazuki, T., **Nishikawa, K.**, **Shintani, K.**, **Kawamura, H.**, **Suzuki, D.** & **Tsuchiyama M**. 2024 Application of finite mixture model to set-based design method based differential evolution. *The Proceedings of Design & Systems Conference* **34**, 3206; doi:10.1299/jsmedsd.2024.34.3206.

Vallhagen, J., **Isaksson, O.**, **Söderberg, R.** & **Wärmefjord, K.** 2013 A framework for producibility and design for manufacturing requirements in a system engineering context. *Procedia CIRP*; doi:10.1016/j.procir.2013.07.041.

van Heerden, A. S. J., **Altelarrea, S. J.**, **Riaz, A.**, **Chen, X.**, **Guenov, M. D.** & **Molina-Cristóbal, A.** 2024 Set-based design techniques for evolvability exploration during conceptual aircraft design. In *34th Congress of the International Council of the Aeronautical Sciences*. Florence, Italy: International Council of the Aeronautical Sciences.

Verma, D. 2023 *Systems Engineering for the Digital Age: Practitioner Perspectives*. John Wiley & Sons.

Ward, A., **Liker, J. K.**, **Cristiano, J. J.** & **Sobek, D. K.** 1995 The second Toyota paradox: how delaying decisions can make better cars faster. *Sloan Management Review* **36** (3), 43–61.

Ward, A. C. 1989 *A theory of quantitative inference for artifact sets applied to a mechanical design compiler*. Technical Report, MIT.

Wood, K. L. & **Antonsson, E. K.** 1989 Computations with imprecise parameters in engineering design: background and theory. *Journal of Mechanisms, Transmissions, and Automation in Design* **111** (4), 616–625.

Yvars, P.-A. & **Zimmer, L.** 2022 Towards a correct by construction design of complex systems: the MBSS approach. *Procedia CIRP*; doi:10.1016/j.procir.2022.05.248.