Design Spaces and How Software Designers Use Them:* a sampler

Mary Shaw
School of Computer Science
Carnegie Mellon University
Pittsburgh PA, USA
mary.shaw@cs.cmu.edu

Marian Petre
School of Computing and Communications
Open University
Milton Keynes, UK
m.petre@open.ac.uk

ABSTRACT

Discussions of software design often refer to using "design spaces", which describe the spectrum of available design alternatives. This supports design thinking in many ways: to capture domain knowledge, to support a wide variety of design activity, to analyze or predict properties of alternatives, to understand interactions and dependencies among design choices. We present a sampling of what designers, especially software designers, mean when they say "design space" and provide examples of the roles their design spaces serve in their design activity. This illustrates some of the ways design spaces can serve designers. Overall, it conveys an understanding of design spaces as lenses used to reduce the overall space of possibilities and support systematic design decision making.

CCS CONCEPTS

Software and its engineering → Software creation and management → Designing software → Software design engineering

KEYWORDS

Design spaces, software design

ACM Reference format:

Mary Shaw and Marian Petre. 2024. Design spaces and how software designers use them: a sampler. In *Proceedings of Designing 2024 Workshop* collocated with ICSE 2024. 8 pages. https://doi.org/10.1145/1234567890

1. What are design spaces?

As Simon famously said, "everyone designs who devises courses of action aimed at changing existing situations into preferred ones" [37]. We consider here design spaces, ways of describing existing situations and the possible preferred ones, and the ways design spaces are used by software designers.

Design spaces embody the design alternatives for problem domains or applications. In practical systems, both the design alternatives and the dependencies among the choices for those alternatives are numerous and open-ended. This creates a dilemma for designers: how to reduce the intellectual complexity of the open-ended "space of possibilities" to something manageable by focusing on a subset of that space that is most helpful to the current state of the design, i.e., a "design space". In identifying a design space, a designer chooses some perspective on the "space of possibilities" intended to help reduce the number of alternatives and dependencies [1], and then navigates within that selected subset [2].[24][44][29][10][4]. The different perspectives (like lenses) have a particular "focal range" or orientation (e.g., structured vs integrated, as discussed in the sections that follow) that shapes both the priorities and the process of exploration.

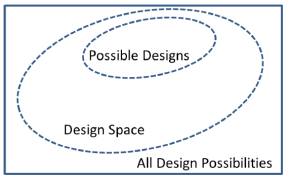

Figure 1: Distinctions among the space of all possibilities, the design space for a subset of that space, and the possible designs that might be identified within that subset. Not to scale. [24]

Woodbury and Burrow [44] recognize the vastness of the space of all design possibilities, with finite but incomprehensibly large numbers of possibilities, but with worthy designs a vanishingly small subset of the space. In such a setting, the problem is not whether a good design exists but whether it is accessible to a designer searching the space. The full set of design alternatives is usually quite rich and quite interconnected, and a decision about one aspect of the design influences choices about other aspects. For example, Schön says "A designer makes things. … Typically this making process is complex. There are more variables—kinds of possible moves, norms, and interrelationships of these—than can be represented in a finite model" [30]. Hence design spaces are helpful—but incomplete.

Engineering practice includes both routine and innovative (also called normal and radical) design tasks; the former involve familiar problems and reuse of large portions of prior work, and the latter calls for novel solutions to unfamiliar problems [31]. Within routine design the design spaces have arguably been well-mapped for

Designing 2024, collocated with ICSE2024

[1] 2: Experts solve simpler problems first. Petre and van der Hoek identified 66 practices that characterize expert designers, based on empirical research. Many of these are evident in our design space examples. We call them out by their numbers [25].

[2] 30: Experts keep options open.

well-scoped domains. Innovative design, by its nature, requires deeper engagement from the designer. Hence the present discussion focuses on innovative design—including engagement with "wicked" problems that resist solution [26], where the choice of design space and the nature of the designer's engagement with design alternatives is instrumental[3].

Designers sometimes address the complexity of the design space by taking a structured approach, making simplifying assumptions to manage the complexity, for example by (provisionally) assuming independence of decisions or focusing on a few principal aspects of the design. At other times, they undertake integrative exploration, keeping the dependencies front-of-mind as they explore their problem understanding and design options. Broadly speaking, structured spaces tend to be associated with explicit knowledge; depth-first search of the design space; treatment of decisions as independent; and reductionist reasoning. On the other hand, integrative design spaces tend to be associated with tacit knowledge; breadth-first search of the design space; attention to dependencies among decisions; and holistic reasoning. In both cases, designers often revise the aspects under consideration as the design evolves[4]. The choice between integrative and structured lenses leads to different approaches to the design space, different representations, and different uses[5].

These lenses are discussed in more detail in the sections that follow, with concrete examples from the literature both in software design and other domains.

2. Structured Design Spaces for Domains

Some design processes create explicit, structured descriptions of the design alternatives. This often focuses primary attention on the design choices rather than the dependencies among these choices. They address the complexity of the design space by making simplifying assumptions such as focusing—for the moment—on a few of the most significant design decisions and their alternatives; they may assume independence among the choices[6]. This is comparable to the engineer's inclination to use linear models whenever possible: they're simple, understandable, tractable, and often good enough[7].

This approach leads to design spaces with descriptions such as "The design space in which a designer seeks to solve a problem is the set of decisions to be made about the designed artifact together with the alternative choices for these decisions. … Intuitively, a design space is a discrete Cartesian space in which the dimensions correspond to design decisions, the values on each dimension are the choices for the corresponding decision, and points in the space correspond to complete design solutions" [34].

A key to the examples in this section is creating an explicit representation of the design decisions, the alternative choices, and the dependencies. The examples in this section show a variety of representations; each is presumably appropriate for its problem and might not be so for other problems[8]. Some are largely Cartesian, some are hierarchical, some are more complex. Some are graphical, some are textual, some are mathematical models. For some the alternatives are ratio-scaled measures, for others some of the dimensions are on nominal or cardinal scales.

Design spaces most often address the problem or requirement space (what the client needs) and the solution or implementation space (how the implementation will accomplish that). They may emphasize functionality, quality attributes, or value information.

The vastness of the design space arises from the open-ended set of possible design decisions. Not only does this lead to combinatorial explosion, it flies in the face of conventional assumptions that specifications are complete and static. We therefore prefer the concept of credentials that capture what you know now, evolve by adding new properties over time, and note the confidence in their correctness[9] [32].

Designers use these design spaces in many ways (discussed briefly in the sections that follow), including:

- Comparing and evaluating existing designs (Section 2.1)
- Capturing domain knowledge (Section 2.2)
- Mapping from problem space to solution space (Section 2.3)
- Analyzing quality attributes (Section 2.4)
- Performing tradeoff analysis in a well-understood domain (Section 2.5)

Defining a design space explicitly entails selecting which dimensions to consider out of all the possible properties that require decisions. The goals of the design and intended use for the space should shape this selection, so the design dimensions of interest are highly dependent on context, and they are likely to change as the designer's understanding of the problem evolves[10]. This is a form of Schön's "reflective conversation with the situation" [29].

2.1. Comparing and evaluating existing designs

Design spaces can be used to compare existing designs, for critique, for evaluation, so that one can select from among products[11]. This is less subject to the risk of oversimplification than other uses, because the current set of elements is known.

At the Software Designers in Action workshop [42], numerous researchers undertook independent analyses of videos of pairs of designers at a whiteboard, each pair addressing the same design brief for traffic signal simulator. (N.B. An analysis from the workshop using a different lens is in Sec 3.1.) As one of the studies in this workshop, Shaw defined a design space to compare the design decisions made by the three teams, the choices implied by the prompt, and the decisions evident in a commercial product [34]. Figure 2 shows the diversity of choices made. For example, all three groups made different decisions about whether traffic signals should be most closely associated with roads, with intersections, or with an entity that connects roads to intersections; a commercial tool associated the signals with intersections.

[3] 36: Experts adjust to the degree of uncertainty present.
[4] 20: Experts reshape the problem space.
[5] 46: Experts explore different perspectives.
[6] 5: Experts design elegant abstractions.
[7] 1: Experts prefer simple solutions.
[8] 28: Experts invent notations.
[9] 26: Experts draw what they need and no more.
[10] 20: Experts reshape the problem space.
[11] 14: Experts look around.

This representation of the design space emphasizes the dimensions. Following Brooks [4], each major group of decisions is represented as a tree with two kinds of branches: choice and substructure. Substructure branches (not tagged) group independent design decisions; choice branches, flagged with "##", provide alternatives. In some cases, the decision is a numeric value, and the choices are implicit.

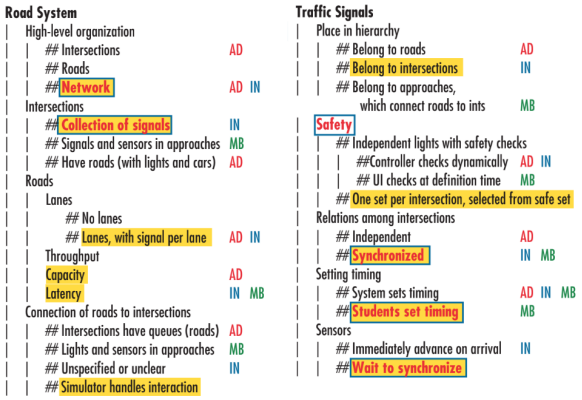

Figure 2: Part of the comparison of several designs for the traffic signal simulator, showing the decisions implied by the task statement (boxed red text), made by the three teams (*AD*, IN, and *MB*), and made by a commercial product (highlighted in yellow) [33]

In an example from another discipline, Römer and Mattern created a design space to compare 15 implementations of wireless sensor networks [28]. They observed that the proliferation of such networks with vastly varying requirements and characteristics was making it increasingly difficult to have useful discussions within the community. They identified 14 major dimensions, most with two or more related subdimensions.

Early superscalar hardware required register renaming to resolve performance bottlenecks. Sima studied a decade of register renaming techniques in 26 RISC and 14 CISC commercial superprocessors and identified a hierarchical design space with four major dimensions to help designers understand and explore this complex space [36].

2.2. Capturing domain knowledge

Design spaces are sometimes defined to capture and explain knowledge about a domain, especially knowledge that will shape many designs of products in the domain[12]. Dimensions of the space are selected to highlight principal distinctions within the space; accordingly, they tend to be fairly static, perhaps evolving as understanding of the domain evolves. However, different aspects of the domain may be significant for different applications, so the dimensions should be selected with that breadth in mind.

In the early development of software architecture styles, Shaw and Clements classified architectural styles in order to establish a uniform descriptive vocabulary, to explain carefully the distinctions among styles, and to lay the groundwork for providing advice about choosing a style appropriate to a problem [35]. The resulting "Boxology" identified six major classes of styles: data flow, call-and-return, interacting processes, data-centered repositories, data-sharing, and hierarchical. Each of these had several specific variants, differing in their constituent parts and their data and control issues. Figure 3 shows a snippet of this space covering one of the major styles.

Following the example of Lane's design space for user interface components [21] (see Section 2.3), the Boxology work identified a few rules of thumb for style selection. For example, it suggested "If a central issue is understanding the data of the application, ... If the data is long-lived focus on repositories. If the input data is noisy and execution order cannot be predetermined, consider a blackboard." [35].

2.3. Mapping problem space to solution space

If both the desired properties of the design (i.e., the requirements) and the implementation alternatives are described by design spaces (that is, if both a problem space and a solution space are under consideration), it is attractive to look for a mapping that guides the designer from a region of the problem space to promising points in the solution space[13].

Lane did this analysis for user interface software structures [21]. He interviewed designers of six software systems to discover the

Style	Constituent parts		Control issues			Data issues				Control/data interaction		Type of reasoning
	Components	Connectors	Topology	Synchronicity	Binding time	Topology	Continuity	Mode	Binding time	Isomorphic shapes	Flow directions	
Data-centered repository styles: Styles dominated by a complex central data store, manipulated by independent computations												Data integrity
Transactional database [Be90, Sp87]	memory, computations	trans. streams (queries)	star	asynch, opp	w	star	spor lvol	shared, passed	w	possibly	if isomorphic, opposite	ACID[5] properties
•Client/server	managers, computations	transaction opns with history[3]	star	asynch.	w, c, r	star	spor lvol	passed	w, c, r	yes	opposite	
Blackboard [Ni86]	memory, computations	direct access	star	asynch, opp	w	star	spor lvol	shared, mcast	w	no	n/a	convergence
Modern compiler [SG96]	memory, computations	procedure call	star	seq	w	star	spor lvol	shared	w	no	n/a	invariants on parse tree

Key to column entries	
Topology	hier (hierarchical), arb (arbitrary), star, linear (one-way), fixed (determined by style)
Synchronicity	seq (sequential, one thread of control), ls/par (lockstep parallel), synch (synchronous), asynch (asynchronous), opp (opportunistic)
Binding time	w (write-time--that is, in source code), c (compile-time), i (invocation-time), r (run-time)
Continuity	spor (sporadic), cont (continuous), hvol (high-volume), lvol (low-volume)
Mode	shared, passed, bdcast (broadcast), mcast (multicast), ci/co (copy-in/copy-out)

Figure 3: Snippet of design space for software architecture styles [35]

[12] 38: Experts address knowledge deficiencies.

[13] 25: Experts draw the problem as much as they draw the solution.

characteristics of the user interface components of their systems and the implementation choices they had made. Based on this, he created detailed functional and structural design spaces. Figure 4 shows the principal structure of these spaces; the actual functional space has 25 dimensions, each with 3 to 5 alternatives, and the actual structural space has 19 dimensions, each with 2 to 7 alternatives. He developed a set of design rules for a recommendation engine that took as input a functional description and produced a ranked set of recommendations for the structural choices; he validated this statistically against the actual designs produced by his subjects. He also produced a set of over three dozen narrative design rules for use by human designers. An example of such a design rule is "High user customizability requirements favor external notations for user interface behavior. Implicit and internal notations are usually more difficult to access and more closely coupled to application logic than are external notations."

1. **Functional dimensions.** These represent the system requirements that are significant in choosing a structure.
   (a) **External requirements.** This group includes requirements of the particular applications, users, and I/O devices, as well as constraints imposed by the surrounding computer system.
   (b) **Basic interactive behavior.** This group includes the key decisions about user interface behavior that fundamentally influence internal structure.
   (c) **Practical considerations.** This group covers development cost considerations; primarily, the required degree of adaptability of the system.
2. **Structural dimensions.** These represent the design alternatives available to satisfy system requirements.
   (a) **Division of functions and knowledge between modules.** This group considers how system functions are divided into modules, the interfaces between modules, and the information contained within each module.
   (b) **Representation issues.** This group considers the representations used for user-interface-related data, including both actual application data (input and output values) and meta-data such as the definition of the user interface.
   (c) **Control flow, communication, and synchronization issues.** This group considers the dynamic behavior of the user interface code.

Figure 4. Overview of functional and structural design spaces for user interface structures [21]

Baum et al. proposed an extension of Lane's model as a design aid for software architecture [3]. They added correlations between dimensions in the functional and structural spaces to show strict dependence, incompatibility, or dependencies that create tradeoff decisions. They also expressed design rules between the functional and structural spaces as correlations.

A design space for self-adaptive systems was developed by Brun et al. to guide designing such a system based on given requirements [5]. It identifies five clusters of design decisions related to control aspects of self-adaptive systems and represents them as questions to be answered by the designer.

2.4. Analyzing quality attributes

In addition to the largely-descriptive, largely-qualitative examples above, design spaces are also created and explored to analyze or predict quantitative or formal attributes of the designs. This is particularly challenging in modern systems that involve the uncertainties that arise from lack of control over third-party components, physical components of cyberphysical systems, or behavior of humans.

Cámara et al. developed a technique for making probabilistic guarantees about such systems [7][8]. Their design space is the set of possible software configurations that are defined by and generated from a formal model that includes structural constraints (architectural style) and application-specific constraints. They explore the design space by further filtering this set of configurations, for example with additional constraints, and quantifying the probabilities of outcomes associated with quality attributes. Unlike most examples in this section, which are represented by explicitly enumerating alternatives, the Cámara design spaces are represented by a formal model.

2.5. Performing tradeoff analysis in a well-understood domain

Disciplines often share and compare results using a consensus model for example[14]. Historically, "stack" served this role for programming languages as did "dining philosophers" for synchronization algorithms and "lift" and "library" for intermediate-level software design. A shared design space can also play this role, but at a more substantive level.

In the domain of hardware/software co-design, a class of performance-critical signal-processing problems involves configuring the hardware-software allocation and parameterization for system-on-a-chip (SoC) applications. "During system-level design space exploration (DSE), system parameters like, e.g., the number and type of processors, and the mapping of application tasks to architectural resources, are considered. The number of design instances that need to be evaluated during such DSE to find good design candidates is large … techniques are needed to optimize the DSE process" [41]. Solutions must balance tradeoffs among chip area, latency, and power [45].

Approaches to achieving designs with better performance include genetic algorithms [38][41], parameter dependency for high-level pruning with genetic algorithms to find a Pareto-optimal tradeoff subspace [22], and semi-random heuristics combined with integer linear programming [45]. Details of the design spaces may differ, but the point of view that performance optimization is coupled to search space exploration is strong. The research is heavy on formalism and includes empirical results.

Kang et al. addressed a similar problem of mapping tasks to devices with a design space exploration technique that systematically eliminated candidates that are equivalent, based on a user-defined equivalence [20]. This enabled them to generate the relevant candidates from the space in a cost-effective manner.

3. Integrative Design Spaces

Some design processes refer to working in a design space, but they don't represent the rich and complex space of alternatives explicitly[15]. They often are more concerned with the design process that exposes and contrasts alternatives, than with the enumeration or representation of the alternatives per se. Hence Woodbury and Burrow [44] argue that design spaces and their representations are helpful—maybe even necessary—tools, but that they should not

[14] 34: Experts make tradeoffs.

[15] 45: Experts generate alternatives.

dictate the design, which requires reasoning beyond representation details[16]. Integrative designs are concerned with a broad exploration of the design space(s), with a focus on identifying the key design considerations—and with ongoing attention to the interactions among the design requirements and features, and the dependencies among the choices explored. Hence, the exploration prioritizes the "essence" of the problem and defers other elements[17]. Schön characterized this integrative approach [29]: "Let us search ... for an epistemology of practice implicit in the artistic, intuitive processes which some practitioners bring to situations of uncertainty, instability, uniqueness, and value conflict".

3.1. Co-evolution of problem and solution spaces

Dorst and Cross studied nine professional designers working on a design for a litter disposal system for trains [15]. They found that the designers start by exploring the problem space and positing a partial solution[18]. They then explore the partial solution in the solution space, which provides insights about the problem space as in Figure 5. Rinse, lather, repeat.

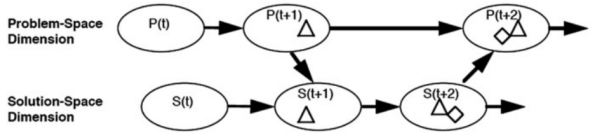

Figure 5. Problem-solution space co-evolution [15]

Curtis reported similar design behavior in software designers [11]. Guindon [18] found that designers bounced back and forth across levels of abstraction and developed mental models of the problem domain and the solution concurrently[19]. Figure 6 shows this, with Curtis' overlay showing the expected attention of a waterfall model. This opportunistic behavior occurred even when a trained software developer attempted to do top-down development. Curtis observed that design insights reorganize the designer's model of the problem domain, which creates new relations between the problem and solution domains[20].

In one of the studies from the Software Designers in Action workshop mentioned previously [42], Tang et al. analyzed the effectiveness of the two teams' design strategies [40]]. (N.B. An analysis from the workshop using a different lens is in Sec 2.1.) One of these strategies is problem-solution co-evolution, in which both the problem space and the solution space are developed and refined together; they associate this with the process of Figure 5.

3.2 Using multiple spaces concurrently to show dependencies

Cai et al. take an integrative approach that uses structured representations. They model software architectures as multiple design spaces, one for each feature and concern, that describe the relations among source[21] [6]. The design spaces encompass design rules, following Baldwin and Clark's design rule theory, in which design decisions create value in a fashion similar to financial options [2].

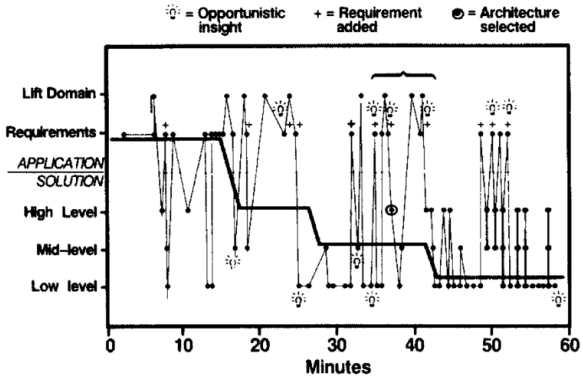

Figure 6. Process map of a software design [18], overlay by [11]

Here design rules are used to capture dependencies between modules, thereby decoupling the components themselves. This yields a representation of the dependencies, called a design structure matrix, in which rows and columns correspond to design decisions and cells show the dependencies, if any. Figure 7 shows a section of a design structure matrix. Clustering of highly interdependent elements show the system architecture. They use the models to evaluate software quality, especially the propensity for bugs.

	1	2	3	4	5	6	7	8	9	10	11	12	13
1 mij.Console	(1)	x	x	x			x						
2 mij.FilterException		(2)											
3 mij.Filter		x	(3)										
4 mij.Interpreter		x	x	(4)	x	x	x	x	x	x	x	x	
5 mij.Repository					(5)								
6 mij.ast.FuncExpr						(6)	x			x			
7 mij.ast.Node							(7)			x			
8 mij.ast.Number							x	(8)		x			
9 mij.ast.OperExpr							x		(9)	x			
10 mij.ast.TreeVisitor						x		x	x	(10)	x	x	
11 mij.ast.UnaryOperExpr							x			x	(11)		
12 mij.ast.Variable							x			x		(12)	
13 mij.bnf.AddExpr													(13)

Figure 7. Section of a design structure matrix; the box identifies an inferred module. [6]

3.3 Design spaces with tightly-coupled decisions

Penders et al. [24] describe an exploration-based design strategy that identifies different design-space dimensions and uses inter-dependency rules between those dimensions to make systematic design decisions. "Inter-dependency rules describe the influence design choices in one dimension have on other design dimensions…" and "Inter-dependency rules indicate that, when limiting the choices in one dimension, there is a direct effect on possible design choices in its dependent dimension(s). Thus, inter-dependencies form a navigation guide for the designer, as they imply correlations between dimensions: after inspecting a design dimension a logical next step is to inspect dependent dimensions." [24]

Hence, the interactions between decisions are a principal focus of the design space analysis. Penders et al. argue that "One of the important advantages of the presented exploration-based design

[16] 60: Experts think about what they are not designing.
[17] 37: Experts focus on the essence.
[18] 31: Experts make provisional decisions.
[19] 47: Experts move among levels of abstraction.
[20] 61: Experts re-assess the landscape.
[21] 54: Experts test across representations.

strategy is its ability to enforce a systematic decision for each dimension in the space" [24]. Figure 8 shows such inter-dependency rules in the context of hardware/software co-design.

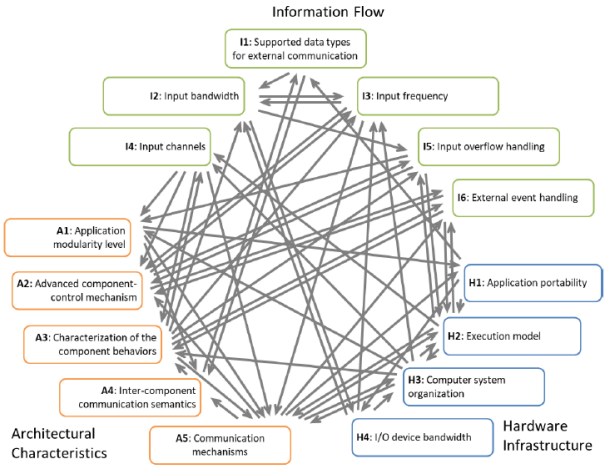

Figure 8. Dependency relationships among decisions [24]

The richness of the interactions among decisions is well illustrated by an example outside of software engineering, the selection of resins for molding plastic parts [19]. Requirements for a molded part might include stiffness, dimensional stability, and resistance to UV; production requirements might include cost of resin, complexity of molding process, and reliability of supplier. "Resin properties influence part performance. … A gain in one property often coincides with a loss in another … Every resin property required for an application influences material selection in varying degrees."

3.4 Implicit design spaces with coupled decisions

Sometimes the important design decisions are related to keeping a system in balance, with an appropriate balance of resources. Here the explicit description of the design space may take a back seat to reasoning about the interactions, so the design space is implicit in the problem analysis[22]. Devereaux [12] considers such a case of logistic considerations for swift operational army maneuvers in the pre-gunpowder period, when "march fast" was the secret to major victories. The essential question is the relation between distance, speed, and force size for moving an army through enemy territory. Devereaux' analysis goes through these steps:

- The limiting factor is meeting nutritional demands; adult men need ~3000 calories/day [identified the key factor: food]
- Historically, this has worked out to ~3 lb of food per person per day, assuming water is available [new parameter: weight per person per day]
- Marching loads are ~80-100 lbs per person with non-food load accounting for all but ~30lb, so ~10 days rations [new factor: food transport]
- The duration can be extended by carrying additional supplies. In addition to food for soldiers, you need food for porters, or horses, or mules. In practice this can only extend the duration to ~40 days [relax the 10-day limit on food transport with a pack train, but at cost of more food and less speed]
- More supplies can be acquired en route by foraging, but this has limited capacity and slows the movement; there is a tradeoff between short operational range at good speed and slower movement with unwieldy supply lines [new parameter for food transport, with associated costs]
- The army will have many non-soldiers performing various duties; this can range from 20% to nearly 100% more human mouths to feed, plus animals [discovered a new parameter, the "tooth to tail" ratio, the ratio of soldiers to support people]
- Bigger armies are slower and harder to coordinate and secure [size affects speed]

The analysis goes on in considerably more detail, but this summary already shows a design space with factors including food, weight of food per day, food transport, ratio of soldiers to noncombatants, and number of horses/mules; these are major determinants of size of army, rate of movement, duration in days, and operational range. These factors interact in ways that sharply constrain the feasible points in the design space.

4. How Designers Use Design Spaces

As discussed, design spaces embody the design alternatives for problem domains or applications, providing different lenses on the design possibilities[23]. What emerges from the sample of design spaces is their role as tools for designers, assisting them:

- To systematize understanding of the alternatives (scope and characteristics) (Section 4.1)
- To systematize navigation of the alternatives (exploration and decision processes) (Section 4.2)
- To support orderly evolution of understanding (through the process of exploration) (Section 4.3)
- To support the orderly reuse of prior art (when existing solutions are appropriate) (Section 4.4).

4.1 To systematize understanding of the alternatives

The examples demonstrate the utility of various design spaces in helping to identify key design considerations, alternatives, and the interactions between them. These may be known considerations—or they may emerge as part of the design space exploration, or in the contrast between different design spaces. Hence, by helping designers externalize (and hence examine) their understanding, design spaces help designers to check for coverage by prompting them to consider alternatives systematically within the space. On the other hand, the systematic exploration can help identify regions of the design space that are not currently relevant, feasible, or prioritized—helping to reduce the alternatives in a reasoned way.

Schön [30] discussed this as problem framing: "In order to formulate a design problem to be solved, the designer must frame a problematic design situation: set its boundaries, select particular things and relations for attention, and impose on the situation a coherence that guides subsequent moves."

[22] 59: Experts retain their orientation.

[23] 49: Experts change notation deliberately.

4.2 To systematize navigation of the alternatives

In order to achieve this understanding, designers need to navigate the design space effectively and systematically. Many authors (e.g., Simon, Newell, Brooks) characterize design as search in a design space. Whitworth and Ahmad further characterize that search as choosing a point in the design space: "System design is choosing a point in a multi-dimensional design criterion space with many "best" points, e.g. a cheap, heavy but powerful vacuum cleaner, or a light, expensive and powerful one. The efficient frontier is a set of "best" combinations in a design space. Advanced system performance is not a one dimensional ladder to excellence, but a station with many trains to many destinations." [43]

That navigation—how designers explore and the design space, and how they choose among alternatives—can take different forms, influenced not only by the design objectives but also, for example, by their domains, experience, representations, processes, etc. Computer scientists are naturally inclined to view this search through an algorithmic lens, expecting systematic, orderly, potentially exhaustive traversal of the search space. Brooks [4] notes the appeal of a rational model that systematically searches the design space but also observes that this model is vastly oversimplified: real designers just don't work that way[24]. In particular, designers often satisfice, stopping the search when an acceptable, if no optimal solution is found[25]. Guidon [18] found an opportunistic mix of breadth-first and depth-first search (see Figure 6). Brooks reports critiques by Cross [10], Schön [29], Akin [1], Royce [27] and others that show not only complex navigation of the search space but also evolution of the space itself as design proceeds. The aspiration to systematic and rational reasoning about design, and the pragmatics of working within the constraints of human cognition are the essence of design space exploration; Dorst [13] still reported in 2006 that the rational, systematic model still looms large in the field.

The representation of the design space may also influence the way it's navigated. A hierarchical representation in which each decision shapes the alternatives available for future decisions tends to encourage making decisions in the order embedded in the hierarchy. "What remains unclear is to what extent such strategies are conditioned by the external memory aids available to designers" [44].

4.3 To support orderly evolution of understanding

Designers' understanding of the design space evolves through the process of exploration. As Cross [9] reported: Schön [29] also pointed out that "the work of framing is seldom done in one burst at the beginning of a design process." This was confirmed in Goel and Pirolli's [16] protocol studies of several types of designers (architects, engineers and instructional designers). They found that "problem structuring" activities not only dominated at the beginning of the design task, but also re-occurred periodically throughout the task[26]. This points to the interaction between the problem space and the solution space.

As the understanding of the design space evolves, the process may reveal critical factors (or change the priorities of different factors) and/or reveal assumptions[27]. As a result, new possible solutions may emerge. Some of the examples highlight the contrasts between design spaces, and in particular the interchange between the problem space and the solution space. "Many studies of expert design behavior suggest that designers move rapidly to early solution conjectures, and use these conjectures as a way of exploring and defining problem-and-solution together"[28] [9].

Dorst asserts, "We urgently need a strong descriptive and analytical framework to help us understand what is actually going on in the "upwards jump" from solution to problem, and how we can safeguard against the misuse and manipulation of emergence" [14].

4.4 To support orderly reuse of prior art

Although this discussion has prioritized innovative or radical design, design spaces play a key role in routine or normal design as well, where for each instance the task is selecting an established design from a well-defined family[29].

In typography, a designspace [sic] is an element of a typeface design that controls how a variable font's appearance changes (by interpolation) as its variation axes are adjusted [17][39]. A variable font with weight and width axes has a 2-dimensional space, as in Figure 9; some parametric fonts have 10 or more dimensions of variability. This allows users of the font to tune, but not to otherwise redesign, the font. Font designers, of course, use the designspace to represent the range of variability they intend [39].

Figure 9. Points in a font design-space [17]

In the domain of civil engineering, Pennsylvania mandates the use of the Bridge Automated Design and Drafting (BRADD) software for new and replacement single span bridges [23]. This software automates the bridge design process from problem definition through contract drawings for specific types of simple single-span bridges. The scope is concrete, steel, and prestressed concrete bridges with spans from 18 to 200 ft, with a variety of site-specific parameters It offers 5 types of superstructures and 3 types of abutments. It consists of 4600 Fortran routines (730,000 lines of code), 590 dimensions/parameter/data files (92,500 lines of data), and 260 graphic details containing 976 overlays, and a Visual Basic user interface of 83,000 lines of code.

5. Conclusion

We sampled the literature, primarily in software engineering and principally in system-level design, to find examples of how designers use design spaces. This is not a comprehensive survey, let alone a systematic review. We reached for the low-hanging fruit, a set of examples rich enough to show the diversity of uses in practice. We included examples from other fields to emphasize that design

[24] 17: Experts use design methods (selectively).
[25] 42: Experts know when to stop.
[26] 22: Experts design throughout the creation of software
[27] 57: Experts are alert to evidence that challenges their theory
[28] 31: Experts make provisional decisions
[29] 13: Experts prefer solutions that they know work

spaces are integral to design generally, not specifically a software construct. This serves our main purposes: to raise the visibility of this design tool, to improve communication by clarifying its various meanings, and to exhibit multiple uses that may stimulate further development.

This sampler shows the wide diversity in content, representation, and use. The main point is the idea of structuring knowledge about a design; the specifics are tools that help designers think, not rigid rules that require adherence.[30].

You, the reader, may use "design space" in yet another way. That's fine, and some future extension of this work might include your use. You should, however, be aware of other ways that other people use the term so that you and they can communicate without confusion; the other examples may inspire you to extend your own.

The designer's principal responsibility is to understand the client's needs and commit to finding a solution that satisfies those needs. All of the models, methods, notations, representations, frameworks, and processes of software engineering are tools to that end, not ends in themselves. The designer should select the ones that help with the problem at hand. These examples show that design spaces deserve space in this toolkit, with details of their role in any particular design subject to the judgment of the designer.

ACKNOWLEDGMENTS

We thank our colleagues who have contributed to this sampler through their research and discussion. This work was partially supported by the Alan J. Perlis chair of Computer Science at Carnegie Mellon University.

REFERENCES

[1] Omer Akin. Variants and invariants of design cognition. *Design Thinking Research Symposium* 7, Taylor and Francis 2008. Reported in [Br10].

[2] Carliss Y. Baldwin and Kim B. Clark. *Design Rules: The Power of Modularity, volume 1*. MIT Press 2000. ISBN 9780262267649 doi: 10.7551/mitpress/2366.001.0001

[3] Lothar Baum et al. Architecture-Centric Software Development Based on Extended Design Spaces. In: van der Linden, F. (eds) *Development and Evolution of Software Architectures for Product Families*. ARES 1998. Lecture Notes in Computer Science, vol 1429, Springer. doi: 10.1007/3-540-68383-6_28

[4] Frederick P. Brooks, Jr. *The Design of Design*. Pearson 2010 ISBN 978-0-201-36298-5

[5] Yuri Brun, et al. A Design Space for Self-Adaptive Systems. In: Rogerio de Lemos et al. (eds) *Software Engineering for Self-Adaptive Systems II*. Lecture Notes in Computer Science, vol 7475, 2013, Springer. doi: 10.1007/978-3-642-35813-5_2

[6] Yuanfang Cai, Lu Xiao, Rick Kazman, Ran Mo and Qiong Feng, Design rule spaces: a new model for representing and analyzing software architecture," *IEEE Transactions on Software Engineering*, vol. 45, no. 7, pp. 657-682, 1 July 2019, doi: 10.1109/TSE.2018.2797899

[7] Javier Cámara, David Garlan, and Bradley Schmerl. Synthesizing tradeoff spaces with quantitative guarantees for families of software systems. *Journal of Systems and Software*, Volume 152, 2019, pp 33-49, doi: 10.1016/j.jss.2019.02.055.

[8] Javier Cámara. HaiQ: Synthesis of software design spaces with structural and probabilistic guarantees. *Proc. 8th International Conference on Formal Methods in Software Engineering* (FormaliSE '20). ACM, pp 22–33. doi: 10.1145/3372020.3391562

[9] Nigel Cross (2004) Expertise in design: an overview. *Design Studies*, 25 (5) September 2004, pp. 427-441.

[10] Nigel Cross. *Designerly Ways of Knowing*. Birkhauser Architecture 2007. ISBN 978-3764384845

[11] Bill Curtis. Insights from empirical studies of the software design process. *Future Generation Computer Systems*, Volume 7, Issues 2–3, 1992, pp 139-149, ISSN 0167-739X, doi: 10.1016/0167-739X(92)90002-S

[12] Bret Devereaux. Logistics, How Did They Do It. [blog, three parts] *A Collection of Unmitigated Pedantry*, July 2022. https://acoup.blog/2022/07/15/collections-logistics-how-did-they-do-it-part-i-the-problem/ https://acoup.blog/2022/07/29/collections-logistics-how-did-they-do-it-part-ii-foraging/ https://acoup.blog/2022/08/12/collections-logistics-how-did-they-do-it-part-iii-on-the-move/

[13] Kees Dorst. Design problems and design paradoxes. *Design Issues* 22: 4-17. Reported in [Br10]

[14] Kees Dorst, 2019, Co-evolution and emergence in design, *Design Studies*, 65,Pages 60-77, https://doi.org/10.1016/j.destud.2019.10.005. (https://www.sciencedirect.com/science/article/pii/S0142694X19300614)

[15] Kees Dorst and Nigel Cross. Creativity in the design process: co-evolution of problem–solution. *Design Studies*, Volume 22, Issue 5, 2001, Pages 425-437. https://doi.org/10.1016/S0142-694X(01)00009-6.

[16] Goel, V and Pirolli, P The Structure of Design Problem Spaces. *Cognitive Science* Vol 16 (1992) 395–429

[17] Google. Designspace. *Google Fonts knowledge glossary*, Retrieved November 2023. https://fonts.google.com/knowledge/glossary/designspace

[18] Guindon, R. Designing the Design Process: Exploiting Opportunistic Thoughts. *Human-Computer Interaction*, 5, 305-344. doi: 10.1207/s15327051hci0502&3_6

[19] Michael Hansen. Material Selection: The Right Resin for Your Design. *MD&DI Medical Device and Diagnostic Industry Magazine*, February 2008, unpaginated. https://www.mddionline.com/materials/material-selection-right-resin-your-design

[20] Eunsuk Kang, Ethan Jackson, and Wolfram Schulte. An Approach for Effective Design Space Exploration. In: Calinescu, R., Jackson, E. (eds) *Foundations of Computer Software. Modeling, Development, and Verification of Adaptive Systems*. Monterey Workshop 2010. Lecture Notes in Computer Science, vol 6662. Springer. doi: 10.1007/978-3-642-21292-5_3

[21] Thomas G. Lane. *User Interface Software Structures*. PhD thesis, Carnegie Mellon University, May 1990. Somewhat more accessible are two technical reports, *Studying Software Architecture Through Design Spaces and Rules*, CMU/SEI-90-TR-18 and CMU-CS-90-175, and *Design Space and Design Rules for User Interface Software Architecture*, CMU/SEI-90-TR-22 and CMU-CS-90-176.

[22] Maurizio Palesi and Tony Givargis. Multi-objective design space exploration using genetic algorithms. *Proc tenth international symposium on Hardware/software codesign* (CODES '02). ACM, 67–72. doi: 10.1145/774789.774804

[23] Pennsylvania Department of Transportation. *Design Manual, Part 4 Structures*. PDT Pub No 15M December 2019. https://www.dot.state.pa.us/public/PubsForms/Publications/PUB%2015M.pdf Software support site for PennDOT BRADD Software, https://bradd.engrprograms.com/home/. Overview in executive summary, https://bradd.engrprograms.com/home/BRADD%20Executive%20Summary.pdf

[24] Ate Penders, Ana Lucia Varbanescu, Gregor Pavlin, and Henk Sips.. Design-Space Exploration for Decision-Support Software. *Proc. 37th IEEE/ACM International Conference on Automated Software Engineering* (ASE '22). ACM, Article 134, 1–6. https://doi.org/10.1145/3551349.3559502

[25] Marian Petre and André van der Hoek. *Software Design Decoded: 66 Ways Experts Think*. MIT Press 2016. ISBN 978-0-262-03518-7

[26] Horst W. J. Rittel and Melvin M Webber. Dilemmas in a general theory of planning. *Policy Sciences* 4 (1973), pp. 155-169. http://www.jstor.org/stable/4531523 This is the "wicked problems" paper.

[27] Winston W. Royce, Managing the development of large software systems: concepts and techniques. *Proc. 9th international conference on Software Engineering* (ICSE '87). IEEE Computer Society, 328–338.

[28] Kay Römer and Friedemann Mattern, The design space of wireless sensor networks, *IEEE Wireless Communications*, vol. 11, no. 6, pp. 54-61, Dec. 2004, doi: 10.1109/MWC.2004.1368897.

[29] Donald A. Schön. *The Reflective Practitioner: How Professionals Think in Action*. Basic Books 1983.

[30] Donald A. Schön. Designing: rules, types and worlds. *Design Studies* Vol 9 (1988) 181–190.

[31] Mary Shaw, Prospects for an engineering discipline of software, *IEEE Software*, vol. 7, no. 6, pp. 15-24, Nov. 1990, doi: 10.1109/52.60586.

[32] Mary Shaw, Truth vs. knowledge: the difference between what a component does and what we know it does, *Proc 8th International Workshop on Software*

[30] 23: Experts do not feel obliged to use things as intended.
21: Experts use notations as lenses, rather than straightjackets

Specification and Design, Schloss Velen, Germany, 1996, pp. 181–185. doi: 10.1109/IWSSD.1996.501165

[33] Mary Shaw. The role of design spaces. *IEEE Software*, vol. 29, no. 1, pp. 46-50, Jan.-Feb. 2012, doi: 10.1109/MS.2011.121.

[34] Mary Shaw. The role of design spaces in guiding a software design. In *Software Designers in Action: a Human-Centric Look at Design Work*. Marian Petre and André van der Hoek (ed). Chapman and Hall/CRC, 2013. ISBN 978-1466501096

[35] Mary Shaw and Paul Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. *Proc. COMPSAC97*, 21st Int'l Computer Software and Applications Conference, August 1997, pp.6–13. doi: 10.1109/CMPSAC.1997.624691

[36] Dezsö Sima, "The design space of register renaming techniques," *IEEE Micro*, vol. 20, no. 5, pp. 70-83, Sept.-Oct. 2000, doi: 10.1109/40.877952.

[37] Herbert A. Simon. *The Sciences of the Artificial*. 1st ed, MIT Press, 1969. ISBN 978-0262190510

[38] Vinoo Srinivasan, Shankar Radhakrishnan and Ranga Vemuri. Hardware software partitioning with integrated hardware design space exploration. *ProcDesign, Automation and Test in Europe*, Paris, France, 1998, pp. 28-35, doi: 10.1109/DATE.1998.655833.

[39] Superpolator. Designspace Theory: some ideas on construction and use. Retrieved November 2023. https://superpolator.com/designspace.html

[40] Antony Tang, Aldeida Aleti, Janet Burge, and Hans van Vliet. What makes software design effective?. *Design Studies* 31 (2010) 614-640. doi:10.1016/j.destud.2010.09.004

[41] Mark Thompson and Andy D. Pimentel. Exploiting domain knowledge in system-level MPSoC design space exploration. *J. Syst. Archit.* 59, 7 (August, 2013), 351–360. doi: 10.1016/j.sysarc.2013.05.023

[42] André van der Hoek & Marian Petre (eds). *Software Designers in Action: a Human-Centric Look at Design Work*. Chapman and Hall/CRC, 2013. ISBN 978-1466501096

[43] Brian Whitworth and Adnan Ahmad. *The Social Design of Technical Systems: Building technologies for communities*. 2nd Edition, The Interaction Design Foundation. ISBN 978-8792964021 https://www.interaction-design.org/literature/book/the-social-design-of-technical-systems-building-technologies-for-communities-2nd-edition

[44] Robert Woodbury and Andrew L. Burrow. Whither design space? *AI EDAM*, 20(2), 63-82. doi:10.1017/S0890060406060057

[45] Wei Zuo, Louis-Noel Pouchet, Andrey Ayupov, Taemin Kim, Chung-Wei Lin, Shinichi Shiraishi, and Deming Chen. Accurate High-level Modeling and Automated Hardware/Software Co-design for Effective SoC Design Space Exploration. *Proc 54th Annual Design Automation Conference* 2017 (DAC '17). ACM, Article 78, 1–6. doi: 10.1145/3061639.3062195