Mahmoud Dinar
Mechanical and Aerospace Engineering,
Arizona State University,
Tempe, AZ 85287
e-mail: mdinar@asu.edu

Andreea Danielescu
School of Computing, Informatics and
Decision Systems Engineering,
Arizona State University,
Tempe, AZ 85821

Christopher MacLellan
Human-Computer Interaction Institute,
Carnegie Mellon University,
Pittsburgh, PA 15124
e-mail: cmaclell@cs.cmu.edu

Jami J. Shah
Mechanical and Aerospace Engineering,
Arizona State University,
Tempe, AZ 85287
e-mail: jami.shah@asu.edu

Pat Langley
Department of Computer Science,
University of Auckland,
Private Bag 92019,
Auckland 1142, New Zealand
e-mail: patrick.w.langley@gmail.com

Problem Map: An Ontological Framework for a Computational Study of Problem Formulation in Engineering Design

Studies of design cognition often face two challenges. One is a lack of formal cognitive models of design processes that have the appropriate granularity: fine enough to distinguish differences among individuals and coarse enough to detect patterns of similar actions. The other is the inadequacies in automating the recourse-intensive analyses of data collected from large samples of designers. To overcome these barriers, we have developed the problem map (P-maps) ontological framework. It can be used to explain design thinking through changes in state models that are represented in terms of requirements, functions, artifacts, behaviors, and issues. The different ways these entities can be combined, in addition to disjunctive relations and hierarchies, support detailed modeling and analysis of design problem formulation. A node–link representation of P-maps enables one to visualize how a designer formulates a problem or to compare how different designers formulate the same problem. Descriptive statistics and time series of entities provide more detailed comparisons. Answer set programming (ASP), a predicate logic formalism, is used to formalize and trace strategies that designers adopt. Data mining techniques (association rule and sequence mining) are used to search for patterns among large number of designers. Potential uses of P-maps are computer-assisted collection of large data sets for design research, development of a test for the problem formulation skill, and a tutoring system. [DOI: 10.1115/1.4030076]

1 Introduction

Many studies of design cognition have employed protocol analysis [1], which is both resource consuming and ambiguous in that the same data can lead to different analyses by different researchers [2,3]. One remedy to this problem is to use a common modeling framework. Different frameworks have been proposed to study design thinking, which will be discussed and evaluated with respect to our research objective; to show differences in problem formulation among designers.

Characteristics of design problems have consequences for choosing an appropriate formalism for studying design thinking. Design problems are ill-defined (with vague or incomplete goals), ill-structured (with conflicting goals, evident or explicit dependencies), and dynamic (with changing requirements). Therefore, a representation of the design space in early stages of conceptual design, when the problem is formulated, should accommodate incomplete, conflicting, and changing problem definitions. At this stage, designers often reframe the problem space [4] and construct multiple representations of the problem [5]. In addition, a representation of problem definition should be able to include elements of the solution space, since the problem and solution spaces co-evolve during design [6].

These reasons motivated creating a framework to study problem formulation with a higher level of detail, and an appropriate formalism for understanding the differences among designers in problem formulation, in addition to improving the process of data collection and analysis. This paper presents the P-maps ontological framework. We show how P-maps can be used to represent and analyze problem formulation in different ways. In the rest of this paper, first the literature is reviewed to see the types of differences that P-maps should capture, and how P-maps should bridge the gaps in existing frameworks. P-maps is then introduced and compared to other design frameworks or modeling formalisms. Different applications of P-maps are then shown for encoding and visualizing differences among designers, tracing adoption of strategies, and discovering common patterns using data mining techniques. The paper concludes with potential uses of the framework in the future.

This paper focuses on describing the modeling abilities of the framework and its potential for improving data analysis. Our previous paper [7] is devoted to an interactive web-based user interface which helps us in collecting, coding, and representing problem formulation data with P-maps. Attention should be paid to the method of the study. With the current number of subjects in our samples, claims are premature. However, the main benefit of using a web-based data collection tool lies in the scalability of further analyses, as programs are written once, and used with relatively cheap computation for larger data sets. Some of the examples in describing the ontology, its formal representation, or possible analyses come from the web-tool data, but some of them are based on previously collected protocols. The P-maps framework models, represents, and supports analyses of problem formulation data regardless of the data collection mode.

2 Related Work

Review of the design literature reveals a few studies that have focused on representing the problem and the solution spaces, as well as some on the process of problem formulation. This section starts with a review of problem formulation in design to highlight the types of data fragments that are present in designers' problem

Contributed by the Design Engineering Division of ASME for publication in the Journal of Computing and Information Science in Engineering. Manuscript received September 13, 2013; final manuscript received February 16, 2015; published online April 24, 2015. Editor: Bahram Ravani.

formulation, and the differences that should be looked for among designers. The review extends to representations that aim at modeling early conceptual design and points out the gaps in these frameworks that should be addressed. In addition, formalisms and models pertinent to a computational study of design problem formulation are reviewed.

2.1 Problem Formulation in Design. Past research has shown a few characteristics of problem formulation, mainly through protocol studies. Designers prefer to treat problems as ill-defined [8,9]. Atman et al. [10] state that senior undergrad design students produce higher quality designs by gathering more information early, considering more alternative solutions, and moving more frequently between design steps. Eisentraut [11], however, maintains that such behavior relates to different styles of problem solving, which are independent of the situation of the design episode.

It is well known that, unlike well-defined problems, design problems continue to evolve throughout the problem solving process. Cross and Cross [9] claim that creative designers, holding experience of previous solutions at the back of their minds, use first principles as stimuli to build bridges between problem and solution space through key concepts. Harfield [12] claims that designers need "protosolutions" to compare the goal and the problem state, and that naive designers make fixed assumptions while creative designers question requirements. Valkenburg and Dorst [4] suggest that a more successful design team frames a design problem more frequently than an unsuccessful one.

In addition, strategies that are adopted in problem formulation have also been studied. Such studies focus on how creative and experienced designers approach design problems differently from noncreative and novice designers. For example, Ball et al. [13] have found that experts lean on experiential abstract knowledge, while novices map the source problem and solution to a target problem. Ho [14] states that expert designers decompose a problem explicitly and directly approach the goal state, while novices decompose a problem implicitly and eliminate a problem when they fail to handle it. Differences among designers do not remain at such coarse levels of creativity and experience. Kruger and Cross [15] categorize designers into problem-driven and solution-driven. Gero and Mc Neill [16] classify the different strategies that designers adopt into micro strategies (analysis, proposition, and making explicit references), and macro strategies (top-down, bottom-up, decomposition, opportunistic, and backtracking).

Furthermore, in addition to the observations that describe the design process, there are some prescriptive models of engineering design that offer different methods and checklists for every step of the design process. The systematic approach of Pahl and Beitz [17] introduced a checklist for developing requirements with a list of examples for geometry, material, ergonomics, assembly, etc., spanning the product life-cycle. Requirements are not only specified individually, but also lead to other requirements, often in a parent–child relation. Developing an objective tree is a common method of eliciting new requirements and determining how they should be synthesized.

Another well-established aspect of problem formulation is the development of function decompositions. Similar to objective trees, function trees are developed to find out what different parts of the design should do to achieve its main purpose. Functions are decomposed into subfunctions until referring to a specific solution becomes inevitable. Many researchers have contributed to a vocabulary for these abstract subfunctions, called function bases. Hirtz et al. [18] combined different function bases to reach a unified vocabulary for a standardized development of function trees. Function decomposition is necessary but not sufficient for describing problem formulation.

2.2 Representation Frameworks in Design. A few researchers have developed models for representing the structure of design problems. Maher et al. [19] link problem definition states to solutions in an abstract way. Goldschmidt [20] captures the indeterministic nature of design by providing multiple representations of figural–conceptual modes. In her node–link representation, she equates states and operators in problem solving with nodes, and their sequences with links. Goldschmidt and Tatsa [21] used linkographs to show that intensive interlinking breeds more creative designs. In characterizing the differences between design and nondesign problems, Goel and Pirolli [22] came up with a task-operator-phase model, inspired by information-processing theory of human problem solving [23].

An established framework in representing design thinking is function–behavior–structure [24]. Function–behavior–structure (F–B–S) has been used in modeling the design process [24], as a coding schema in protocol analysis [16,25], and for design automation [26]. Gero [24] identified activities in the design process in terms of transformations from one of the three domains of function, behavior, or structure to another, considering a difference between expected and actual behavior. For example, transforming a function to an expected behavior is considered formulation or specification. Gero and Kannengieser [27] took into account the dynamic character of design by considering the notion of situatedness. Even though F–B–S has been used as a predefined coding schema in protocol analysis [25], it has not been used as a computational framework for searching for strategies because, as Gero and Kannengieser contend [28], F–B–S is a high-level model.

Independetly, Goel et al. [29] have developed the structure–behavior–function (S–B–F) modeling language for a teleological description of complex systems. In this language, structure, behavior, and function are represented in terms of components and their connections, transitions among a sequence of states and pre- and postconditions, respectively. The syntax is similar to notations that are used to represent production rules. The model is a top-down description scheme, in which each fragment of the model is defined by a lower level fragment. At the top, there is an instance of S–B–F, while at the bottom there are building block fragments such as strings and integers.

Besides developing modeling frameworks that can be used commonly in studying different aspects of design cognition, others have tried to employ standard modeling languages. Wölkl and Shea [30] have used SYSML in modeling conceptual design. They follow the prescribed systematic engineering approach by Pahl and Beitz [17]. They propose creating new specifications with requirement diagram, describing functions with use case diagram and activity diagram, and allocating working principles with block diagram. Using such a standard language makes it easier to integrate the often nongeometrical data of conceptual design with later stages of product development. However, Wölkl and Shea [30] concede that the representation is not compact from usability viewpoint, and multiple (and separate) diagrams are required to represent different aspects of the designs. This makes it less likely to see the problem in context, or boost creative ideas which often arise from seeing the interconnections of concepts [31].

2.3 Computational Modeling Formalisms. Section 2.1 reviewed some of the representation models in engineering design. There are inspiring formalisms in computer science that should be mentioned for two reasons: Such formalisms have been used for representing knowledge, and thus (design) thinking (representation aspect); they will be pertinent to automating analyses of design thinking data (computation aspect). Concept map [32] is a representation model which has been used in education as means of providing students with an easy and intuitive way to document and explain taught lessons. Novak and Cañas [33] have proposed the use of concept maps to identify changes in students' understanding over time. Additionally, concept maps have been used to understand the differences between the knowledge of experts and novices. The main advantage is the ability to accommodate fine levels of granularity. Even though concept maps have nodes and

labeled links and can represent hierarchies, they are still relatively unstructured. There is no standard way or ontology and one can label data fragments in any way. This becomes a major shortcoming, especially when one wants to compare different instances of the problem formulation over time or to compare models of different designers. Figure 1 exemplifies a concept map of a problem formulation process of designing a water sampling device.

Semantic networks [34] are a type of graphical network that relate conceptual nodes with binary links. They have been used to represent the meaning of sentences in natural language processing. Nodes are used for representing concepts and links for the types of relationships among them. This is a graphical representation of some static situation, e.g., a person's mental state. Concepts are usually organized in a taxonomic hierarchy and often rely on the use of inheritance [35]. Semantic networks struggle to represent disjunction [35], which is important in representing design problem formulation.

To summarize, we reviewed previous studies in understanding problem formulation. Some of the representations that have been developed for studying design thinking and modeling design processes were also reviewed. In addition, we reviewed a few formalisms that might be used in representing or building a computational model of design thinking. Studies of problem formulation have been fragmented, and representation models that have been proposed in studying design cognition, though have led to interesting findings, do not have the necessary level of detail for studying problem formulation. Therefore, there was a need for a new modeling framework that was fine-grained, and incorporated formalisms that facilitated showing differences among designers' problem formulation with a potential for automating analyses of data collected from a large number of participants.

3 The P-Maps Framework

This section presents the P-maps data model, describes the procedure that led to the creation of the framework, and justifies the modeling characteristics that the framework should possess. It also shows the different aspects that differentiate P-maps from other data modeling frameworks that were reviewed in the literature.

3.1 Developing the Ontological Framework. To develop the P-maps framework, three steps were taken. First, two protocol studies were conducted in the exploratory phase of this research. In the first study [36], data fragments were organized into different categories to highlight contrasts between an expert designer and novice designers. In the second study [37], a state model was defined as a list of instances of different categories and their relations in the order they were added. A sequence of these states revealed temporal changes in designers' formulations and facilitated comparing formulations of different designers for the same problem. The design problem in both studies was designing an inexpensive, reusable water sampling device.

In the second step, an exhaustive list of relevant entities in problem formulation was created. Entities that appeared to be similar concepts in different terminologies were merged. For example, in early stages of an engineering design process, the term *requirement* can be used with no trepidations instead of specifications, demands, customer needs, goals, objectives, or constraints. All of these terms have slight differences which can be either safely ignored, or accommodated with the definition of proper attributes for the entity, e.g., goals can be defined as requirements that one prefers to maximize or minimize (e.g., life or weight). The addition of a *target* attribute for the *requirement* entity accommodates goals.

Third, the P-maps framework emerged from synthesizing the two taxonomies explained above, with respect to a set of desired features in order to reach the main research objective, i.e., to show differences among designers' problem formulation. These criteria and their justification are shown in Table 1.

3.2 P-Maps Data Model. The P-maps framework has incorporated five types of entities to model the early stages of conceptual design, i.e., problem formulation. These five types are *requirement, function, artifact, behavior, and issue*. Each type consists of entities whose instances can be a part of disjunctive hierarchies. Entities can have optional attributes. The entities in the five groups can be organized into hierarchies with parent–child and preceding–succeeding relations. In the *artifact* group, the *solution principle* entity, an abstract concept of a solution, can be combined with *physical embodiments*, tangible components such as gears and motors. All groups except the *issue* group are inter-related with bidirectional relations, while the *issue* group can have a relation to any combination of the rest of the

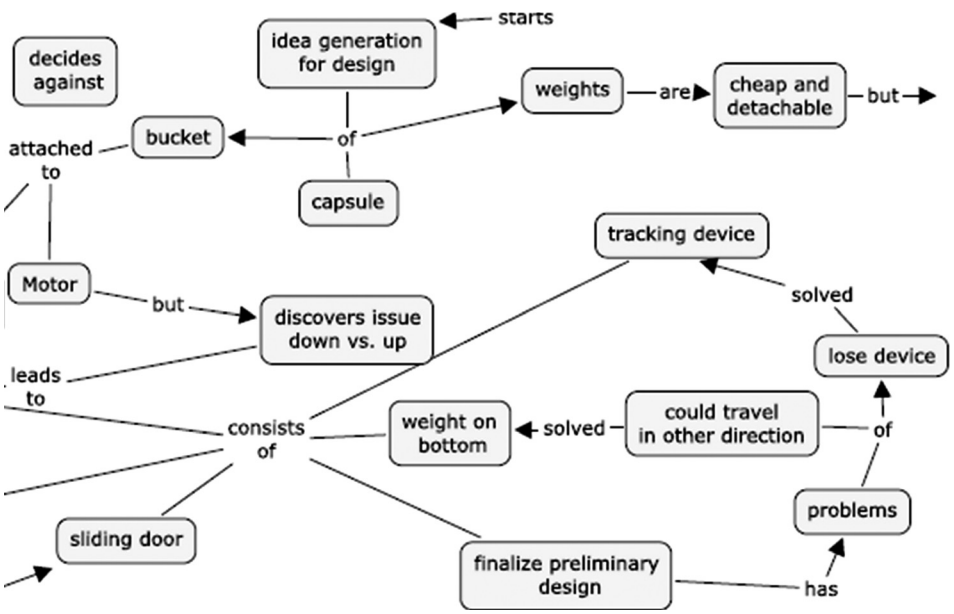

Fig. 1 A part of a concept map used in formulating a design problem

Table 1 A list of desired modeling features considered while developing P-maps

Feature	Justification
Problem and solution oriented	Co-evolution of problem and solution spaces during problem formulation
Hierarchal	Describing compositions and levels of abstraction
Sequential	Showing precedence in one level of abstraction
Disjunctive	Considering alternatives with common or independent fragments
Domain-independent	Describing problems regardless of domain knowledge
Expressive	Discerning differences in the formulations of different designers
Unambiguous	Common understanding among different researchers and users
Flexible	Reflecting unscripted and unpredictable moves in design thinking

entities. Figure 2 shows an abstract model of the five groups of P-maps and the relations that can arise among them.

In P-maps, requirements are the entities that describe the specifications of the design problem. A design problem is usually given as a design brief or problem statement. The design problem is formulated with additional requirements elicited by the designer. The problem is specified by a set of design goals and requirements. The *source* attribute can identify why a requirement is defined; it can have specific values such as "safety," "aesthetics," "ergonomics," "use_environment," and "affordance." It can also specify whether a requirement is explicitly "given" in the problem statement or is implicit and "derived" by the designer. In the P-maps vocabulary, requirements are binary. Goals can be defined with a *target* range. There is usually a relation between the level of satisfaction in using the solution and the degree to which the goal is achieved in terms of a utility function. Another optional attribute is *importance* or *weight*. The *requirement* hierarchy is similar to an objective tree, especially with assigned weights.

Functions refer to what the design does and the actions that the design will execute; examples are "rupture disk," "carry passenger," or "amplify torque." Functions are realized by artifacts and satisfy the requirements. In the *function* group, the hierarchy is evident in functional decompositions. The P-maps model incorporates disjunctive composition, making it possible to have multiple functional decompositions using common subfunctions.

Artifacts realize functions and are the entities that describe the physical components of the design or the concepts the design may be using. In the *artifact* group, the hierarchy of physical embodiments and solution principles is similar to product architecture. The P-maps model also allows partial compositions of solution principles and physical embodiments, since in reality, the designer follows different parts of the subsolutions at different times corresponding to different levels of abstraction.

Behaviors are the physical properties and laws that govern the design. These entities include equations and physical effects, as well as the parameters that are relevant to both artifacts and functions. In the *behavior* group, a physical effect can be represented as a hierarchy of physical laws, which in turn can be shown as a hierarchy of parameters.

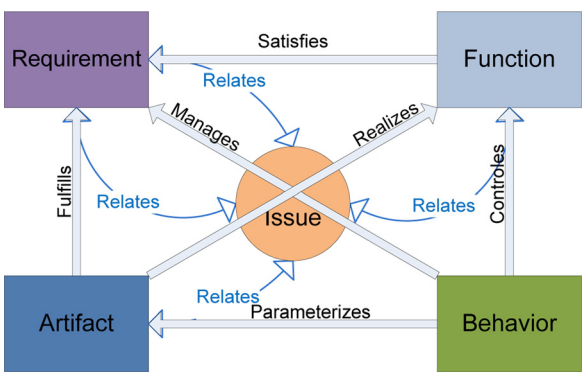

Fig. 2 The entities and the relationships of the P-maps framework

Issues are entities that describe the problems associated with other entities in the design formulation. An issue is often an expression of a point that the designer believes to be pivotal or problematic in achieving a design objective. To name a few, an issue can arise in realizing a function with a specific artifact or behavior, in realizing conflicting design goals such as lower weight and strength of a structure, or in accommodating different components in a product architecture due to incompatible interfaces. The designer gains insight in the discovery of key issues in the design and the areas of the design that should be prioritized. The hierarchy of issues may represent a problem solving strategy. It may also entail the priorities that the designer gives to the issues that should be addressed; such priorities may correspond to a problem solving strategy. The issues may be categorized with an optional attribute of *type* with values such as "question" ("will this structure bear this load?") and "concern" ("experience shows that it is difficult to reach this fuel consumption rate with the desired towing capacity and top speed").

Besides hierarchies and partial orderings, which manifest intragroup relations, intergroup relations are also defined; an underlying property of ontology. This leads to an expressive and flexible model that can show how different designers see the relations among different aspects of a problem and the alternative ways they relate. For example, alternative conceptual designs with common components or different function decompositions can be shown with different branches of an artifact or function hierarchy with nodes that have the same name for the common components or functions, respectively. A specific name is assigned to the relation between any of the two entity groups. For example, an artifact *realizes* a function, and a behavior *manages* a requirement. The P-maps model does not make a distinction between explicitly known relations (e.g., when a designer knows that the power equation of an electric motor *manages* the desired torque), and implicit or qualitative guesses (e.g., when a designer knows that a parameter *manages* a specific goal but does not yet know how exactly).

This section presented the P-maps framework and some of its modeling abilities. The required modeling features of the framework were also discussed. The P-maps framework can be compared the frameworks which were reviewed in Sec. 2 with respect to some of the required modeling features of Table 1. The comparison is listed in Table 2. Comparing some aspects requires additional studies, e.g., unambiguity can be determined by inter-rater agreement, as it was carried out for P-maps, see Sec. 5.1. Section 4 focuses on the different types of analyses that P-maps facilitate.

4 P-Maps Data Analysis Methodology

The graphical and textual formalisms of P-maps make it possible to capture, represent, and analyze problem formulation data in different ways. In this section, we first present a graphical representation of P-maps that provides a way to visualize a designer's progress in terms of successive states, which in turn serves as a means to comparing different designers' formulations. Second, we explain how descriptive statistics of P-maps can be used for

Table 2 Comparison of different modeling frameworks to P-maps

	Covering problem and solution aspects	Capturing hierarchical structure	Representing alternatives	Linking data fragments	Content to view size ratio
F–B–S	Problem and solution	No	No	No	Compact
S–B–F	Solution-oriented	Yes	Implicit	Yes	Large
Linkograph	Solution-oriented	No	Explicit	Yes	Compact
SYSML	Problem and solution	Yes	Implicit	Yes	Large
Concept map	Problem and solution	Yes	Explicit	Yes	Large
P-maps	Problem and solution	Yes	Explicit	Yes	Compact

quantitative analyses of problem formulation. Third, we show how to use P-maps for testing hypotheses about strategies that designers adopt during problem formulation. The section ends by discussing potential for future work in hypothesis generation using data mining techniques, devising a test for designers' problem formulation skills, and creating a tutoring system.

The section exemplifies these approaches with data collected from protocols [37], as well as the Problem Formulator web tool [7]. The protocol data were collected from eight expert designers working on the water sampling problem [37]. Each expert was given an hour. Their protocols were manually encoded into P-maps by two researchers through a process of arbitration. The data collected with the Formulator web tool were from a class of 62 undergrad students of a mechanical design course working on designing a machine to recycle aluminum drink cans. The students learned about the web tool and the ontology through a practice problem.

4.1 Comparing Formulation States Using Node–Link Graphs. A node–link graphical representation of P-maps visually shows the evolution of a designer's problem formulation through changes in states at different times. Figure 3 presents one designer's problem formulation, based on coded protocols [37], at two P-maps states of the same problem. The designer thinks about a triggering function that leads to taking the sample by proposing a flip-flop mechanism that is controlled by a ruptured diaphragm, and then by contemplating the idea of having a scuba diver take the sample which he later finds contradicting the requirement that the device should be detachable, labeled "not self-contained" from the transcript. P-maps can also be used to compare different designers' formulations. We have also used this representation to compare two designers at the end states of P-maps. Comparing successive states of one designer's formulation or that of different designers gives an overview of some differences among designers in problem formulation.

4.2 Descriptive Statistics of P-Maps. P-maps can also be used for a quantitative analysis of differences among designers' problem formulation. This involves calculating descriptive statistics of variables such as the number of instances of each entity, their means and quartiles, and time series of the emergence of the entities. The examples in the following paragraphs are from the coded protocols of the eight expert designers [37]. For the first analysis, the overall number of entities within the five groups were plotted over a normalized timescale to eliminate differences in the length of the design sessions. Each coded predicate equaled one time step. Figure 4 shows the cumulative number of entities for a designer who specified more problem-related aspects of the design by continuously adding new requirements and functions compared to a designer who focused on solution-related aspects, especially in specifying behaviors.

To see whether or not the designers approached their design sessions in a similar order, Fig. 5 shows box plots that clarify when two of the designers defined their requirements, functions,

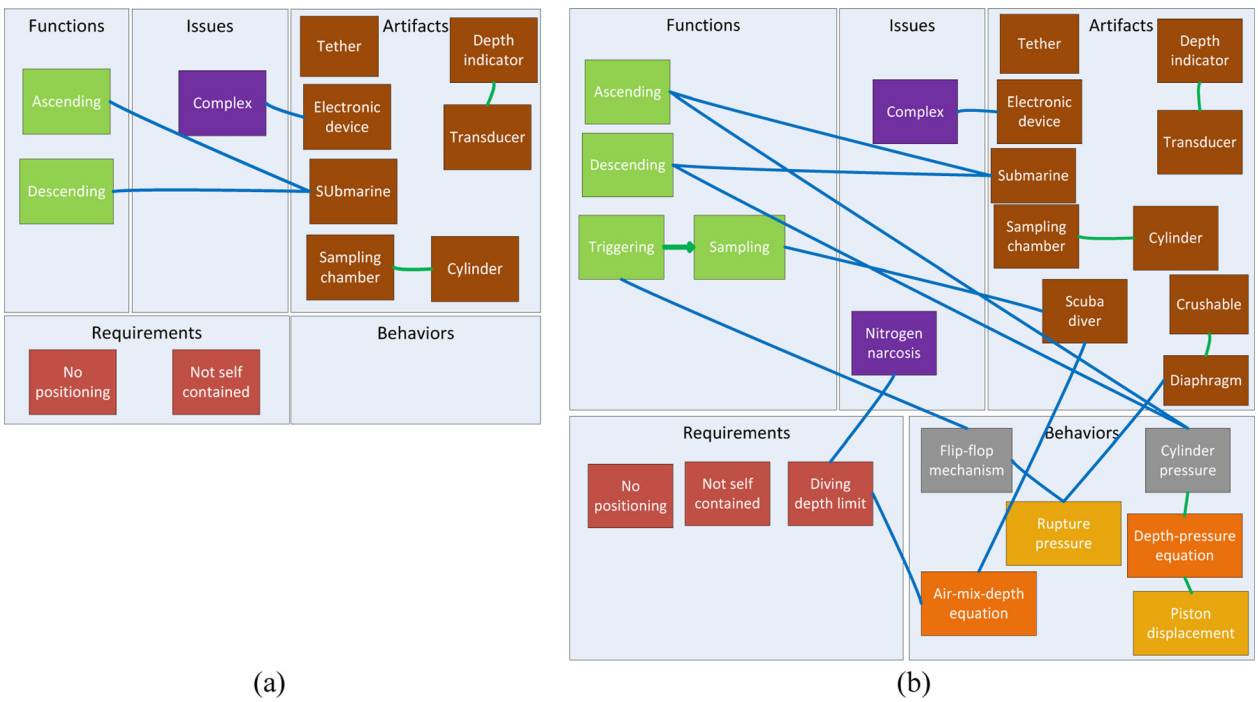

Fig. 3 P-maps states representing changes in a designer's problem formulation through time: (*a*) after 4 min and (*b*) after 8 min

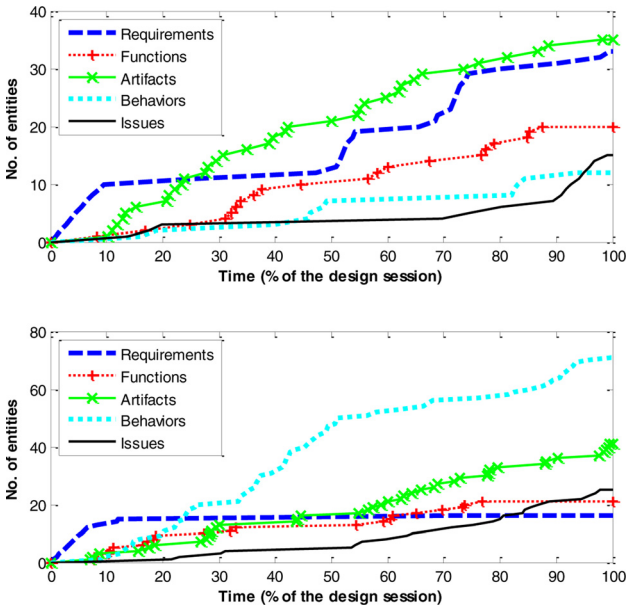

Fig. 4 Differences in thinking about the problem; the designer on the top continuously adds requirements, functions, and artifacts, while the designer on the bottom focuses on behaviors (from Ref. [37])

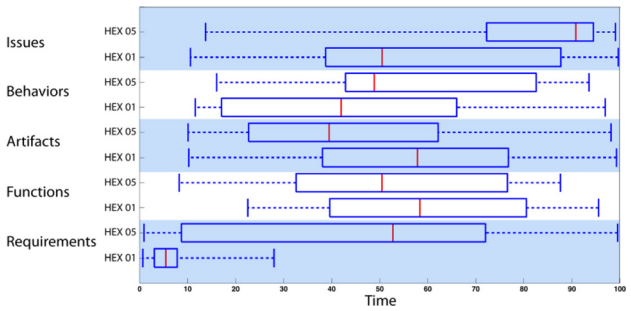

Fig. 5 Comparison of box plots of the five entities for two designers (from Ref. [37])

artifacts, behaviors, and issues. As mentioned previously, the designer that defined requirements throughout the design process was atypical and in fact, the other designers specified requirements towards the beginning of their sessions. Additionally, most designers specified most of their issues towards the end of their design sessions, as they were reviewing the design space they had explored.

Although designers have different styles of problem solving that are not dependent on the solution [11], there are some similarities in the ways in which they move among the five groups of entities. Figure 6 compares how two designers moved among the five groups of entities. The iterations show that the process of defining artifacts, behaviors, and functions was strongly intertwined. However, one designer (the top graph) develops an aspect before moving to another aspect of the problem, while the other designer (the bottom graph) quickly shifts attention to different aspects of the problem. In general, the subject designers often went back and forth quickly between defining their artifacts and their functions. For those who also spent substantial effort identifying behaviors, the behaviors were often intertwined with functions and artifacts.

4.3 Formalizing and Tracing Strategies With ASP. P-maps can be used to represent and formalize design strategies that human designers adopt, by defining condition-action rules that describe how states change during the development of P-maps.

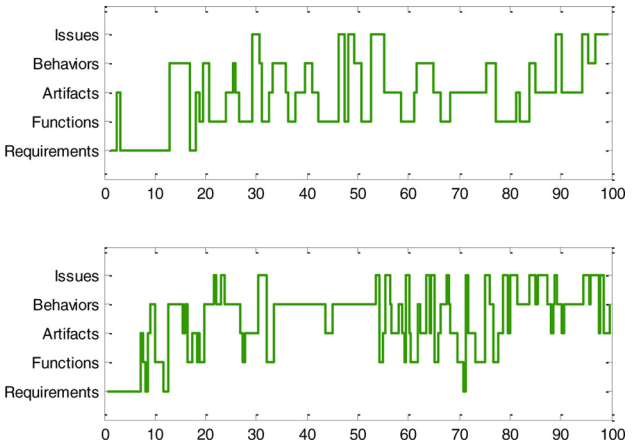

Fig. 6 A comparison of iterations among different entity types for two designers

Strategies can be formalized not only in how entities appear but also with certain conditions where they apply. This section uses three examples to describe the method of employing predicate logic formalism and ASP [38] to formalize and trace problem formulation strategies. The reasons for choosing ASP are:

- ease of analysis in a declarative syntax compared to procedural programming
- simplicity of the logical formalism that makes encoded fragments easily readable and close to natural language
- easy conversion of P-maps data from a conventional database to an ASP representation
- ease of performing automated reasoning over the P-maps predicates

Answer set programs consist of two main components: facts, which are the ground literals over which the system reasons, and rules, which are used to perform logical reasoning over the facts. Predicates are represented with a name followed by braces which contain the values of the attributes that define the predicate. Consider an example of a design of an airplane seat that can be automatically turned into a bed; two functional decompositions can be expressed with these predicates[1]:

fnction(fn_support_sleeping).
fnction(fn_move_to_flat_position).
fnction(fn_support_weight).
parent_of(fn_supporting_sleep, fn_move_to_flat_position, hy_fn1).
parent_of(fn_support_sleeping, fn_support_weight, hy_fn1).
before(fn_move_to_flat_position, fn_support_weight, hy_fn1).
fnction(fn_move_to_vertical_position).
fnction(fn_hang_weight).
parent_of(fn_support_sleeping, fn_move_to_vertical_position, hy_fn2).
parent_of(fn_support_sleeping, fn_hang_weight, hy_fn2).
before(fn_move_to_vertical_position, fn_hang_weight, hy_fn2).

Relations among entities are expressed in a similar way. For the airplane seat example, the following relations describe an issue with a proposed solution to fulfill a requirement:

fulfills(sl_pivoting_recliner, rq_support_250lb_weight).
issue(iu_support_weight_at_flat_position, "load on a cantilever causes high bending stress").
relates(iu_support_weight_at_flat_position,ph_bending_stress).
relates(iu_support_weight_at_flat_position, rq_support_250lb_weight).
relates(iu_support_weight_at_flat_position, sl_pivoting_recliner).

[1]The term function was a reserved word in the ASP solver that we employed, thus fnction is used when representing predicates of functions.

To show how strategies can be traced three examples are explained. For the first example, consider a strategy where designers abstract an aspect of a problem definition. When exploring the design space, a designer can add more detail to an idea or generalize that idea. The ability to abstract concepts is considered a key in creative idea generation [39]. To see whether a designer has employed an abstraction strategy during an interval, one can examine the changes in two state models at the beginning and the end of that interval, then see if the designer added more specific details to a stated thought or if he generalized those parts to more abstract concepts.

To explain the tracing mechanism let us introduce a definition of states and operators in our framework. The definitions may seem to be arbitrary considering the fact that it is difficult to clearly define boundaries of mental states for human subjects. Consider the simple case where any change such as the addition of a new instance of an entity, specifying an attribute of an existing entity or relating two instances is an operator that alters the current state into a new state. Strategies can be traced by comparing two states in an interval during which one expects the strategy to be employed. Going back to the example of the abstraction strategy, we look for the states that include parent–child relations. Then we locate the two states that contain the parent and the child. If the state that has the parent occurs after the state that has the child, it indicates that the designer followed an abstraction strategy. The strategy may be employed with any entity, but we will only show an example with requirements. Three operators in a specific order define the strategy. Two are the addition of the requirements and one relates the parent to the child in an instance where the parent was added before the child. The states will be:

State at T1: requirement(rq1).
State at T2: requirement(rq2).
State at T3: parent_child(rq2,rq1,hy1).

where $T1 < T2 < T3$. Instances of strategies are traced using an ASP solver program. In most ASP solvers, a predicate that ends with a dot represents a fact, the head of a rule is separated from the body by colon and dash, and variables are capitalized while instances are in lower case. The abstraction strategy that was previously illustrated can be traced by using an ASP solver and applying the following rule to all the predicates (facts) that are derived from a P-maps mode (in its textual format in the form of logic predicates):

strategy(upward_abstraction,Entity_parent):- entity(Entity_parent, T_parent),
entity(Entity_child,T_child), parent_of(Entity_parent,Entity_child, T_parent_of),
T_parent > T_child.

The rule matches against a parent entity whose creation is later than that of its child. For any entity that matches against the rule, an answer is generated with a predicate "strategy(upward_abstraction,Entity_parent)." Notice that "upward abstraction" is in lower case and "Entity_parent" is in uppercase. As we noted in earlier comments about the flexibility of P-maps, predicates can be declared at any time and there is no restriction on the order of adding new predicates in relation to others. This means that, in defining a hierarchy, it is not necessary to add children to existing parents.

A more complex strategy to trace is if designers follow a unidirectional process in building the problem space. One might assume that requirements and functions precede artifacts, behaviors, and issues if the designer follows a prescribed process such as Pahl and Beitz's systematic design approach [17]. One can then consider that functions that satisfy requirements before artifacts and behaviors for those requirements represent a unidirectional forward process. To formally state the strategy in P-maps we should look at each requirement to see if it is satisfied by a function before being related to other entities. We should include all possible combinations of relations for this strategy (depending on what relations exist between a requirement and other entities). We show two combinations here:

strategy(forward_processing,Requirement): -satisfies(Fnction, Requirement,T_satisfies),
fulfills(Artifact,Requirement,T_fulfills), manages(Behavior, Requirement,T_manages),
relates(Issue,Requirement,T_related), $T\_satisfies < T\_fulfills$, $T\_satisfies < T\_manages$,
$T\_satisfies < T\_related$.
strategy(forward_processing,Requirement):- satisfies(Fnction, Requirement,T_satisfies),
relates(Issue,Requirement,T_ related), $T\_satisfies < T\_related$.

The two strategies were traced for both data sets: protocols of experts [37] and students' data collected by the Formulator web tool. The results are shown in Table 3. Making claims about differences between novices and experts from this comparison is premature. However, two interesting observations can be taken for future investigations: (1) experts frame problems more abstractly than novices do; (2) the more balanced distribution of strategies adopted by the students suggest that using a computer tool increases the chances of getting higher hits on the occurrences of design strategies, simply because of the limitations of protocol analysis methods in eliciting verbalized thoughts, as noted by Cross et al. [3]. In addition, strategies with zero medians are good candidates for further studies where we can compare two groups of participants, one of which is deliberately prompted with adopting the strategy.

The approach to tracing formally declared strategies offers a top-down approach to hypothesis testing. One can look in the literature or use one's intuition to form hypotheses about strategies that designers adopt to test for their presence. One can also employ data mining techniques to generate hypotheses from collected data.

4.4 Search for Patterns in Problem Formulation Data. The fine granularity of P-maps makes it possible to extract information about problem formulation in terms of different variables that describe a state at a time, or differences between two states. These variables can be chosen to be independent of each other. For example, hierarchy depth for an entity (i.e., the maximum number of levels of parent–child relations) is independent of the frequency of that entity (the number of instances of that entity). In other words, two designers may have specified 20 requirements in their P-maps, but one is in a wide hierarchy with two levels while the other is in a deep hierarchy with four levels. Table 4 shows an example of a few P-maps variables that were computed for a data set collected with the Formulator web tool [7]. Extracting some variables such as the number of implicit requirements (currently) involves human judgment of the data entered in the tool, while variables such as the deepest entity or the number of relations between artifacts and requirements can be automatically found with querying the database of the tool.

Once a large number of P-maps are collected, data mining can be used to search for patterns. One possible method is the mining of association rules that have high enough scores of confidence and lift [40], representing commonness and high correlation, respectively. Apriori association rule mining [40] of the data set example of Table 4 (which includes 14 more instances, a total of 20 P-maps) reveals some relations among the variables, examples of which can be seen in Table 5. It should be emphasized that one

Table 3 Variations in adopting two strategies among students and experts

	Students		Experts	
	Upward abstraction	Forward processing	Upward abstraction	Forward processing
Mean	2.9	1.1	6.1	0.13
Median	3	0	4.5	0
Standard deviation	2.7	2.5	3.6	0.35

Table 4 Example of a data set of P-maps variables

Implicit requirements	Function depth	Function artifact relations	Deepest entity	Requirement percentage	Artifact percentage	Dominant fourth quartile
4	2	0	Function	0.52	0.11	Requirement
0	2	2	Requirement	0.47	0.32	Artifact
0	2	0	Function	0.19	0.26	Function
4	2	14	Artifact	0.21	0.31	Issue
1	1	6	Behavior	0.19	0.16	Issue
1	2	3	Function	0.26	0.26	Function

should be cautious with interpreting such rules and use them for suggesting hypotheses, not proofs; the rules show concurrency, not necessarily causality. In addition, generalization of such rules to other problems may be incorrect. In the given example data set, number of implicit requirements or function depth depend on the choice of the exercised problem. One remedy is to use normalized variables when applicable.

Another possible method is sequence mining. One can write a sequence of the entities, attributes, and links that are added in P-maps of each designer so that a sequence represents the order of entities that the designer created. The sequences collected from different design sessions can be searched for frequent subsequences with high measure of support; that is to see how frequently a partial order of the entities appeared among different designers. Table 6 shows results from the data set collected with the web tool. The subsequences had a support measure more than 0.5, indicating that they occurred among more than half of the students.

4.5 Potential Applications: Problem Formulation Skill Test and Tutoring System. Possible future applications of the P-maps framework are the development of a test of problem formulation skill and tutoring for that skill. The test development application not only includes collecting data to determine what design problems, questions, and activities are better for identifying differences in the designers' skills but also extends to automating the process of assessing test takers' responses. Once data are collected for various activities, text processing techniques can be used to classify the responses into different bins or categories based on semantic distance. A union of all responses, especially if taken from experts, can form a normative P-maps for each problem, question, or activity as a basis for scoring test takers in the future. This is similar to the process that was carried out for developing Shah's divergent thinking test [41], with the main difference that the latter was developed with pen and paper, a resource intensive method.

Another future application extends the Formulator web tool into a tutoring system. With more findings about how different strategies (dynamic changes in P-maps states) and variables (static snapshots of P-maps states) correlate with creativity, at each step of problem formulation practice tasks, one can generate the right feedback and hints which are common behaviors of tutoring systems [42].

5 Discussion

This section discusses the modeling abilities of P-maps and their limitations in capturing and analyzing problem formulation data. A common measure of assessing a coding schema is finding the inter-rater agreement. Even though P-maps have an acceptable level of agreement among coders, most of the limitations in capturing the data emerge if one uses P-maps as a coding schema to encode protocols. This is a shortcoming of protocol analysis. Collecting data with a computer tool that organizes data according to P-maps ontology will overcome such limitations. We have developed a computer tool that serves this purpose [7]. We also discuss the differences in the top-down vs. bottom-up approaches for analyzing problem formulation data.

5.1 Assessing Unambiguity of the Framework With Inter-Rater Agreement. A common method of determining unambiguousness of an ontology is to find the inter-rater agreement. Two statistical measures of agreement (taking into account agreement occurring by chance) in assigning categorical ratings are Cohen's kappa [43] and Fleiss' kappa [44]. Cohen's kappa is used for two raters while Fleiss's kappa is for any fixed number of raters. To measure the inter-rater agreement in coding protocols within the P-maps ontology, segments of code were given to trained raters who were asked to assign one of the seven categories {requirement, function, artifact, behavior, issue, hierarchy, intergroup} in P-maps: the five entity groups, hierarchical relation, and the intergroup relation. The required sample size was found to be 51, based on Gwet [45] for an expected agreement of 70% among the coders, and an expected 20% error in coding for each

Table 5 Examples of association rules found in a data set of P-maps variables

Rule	Confidence	Lift
Implicit requirements = 0 ==> function depth = 2 and dominant fourth quartile = artifact	0.5	2.1
Function artifact relations = 1 and requirement percent = '(0–0.196)' ==> implicit requirements = 0	1	2.1
Implicit requirements = 1 ==> deepest entity = function	0.83	1.59

Table 6 Frequent subsequences with a support higher than 50%

Sequence	Support
['requirement', 'fnction']	0.59
['fnction', 'fnction', 'fnction']	0.59
['requirement', 'requirement', 'requirement']	0.62
['requirement', 'parent_of_requirement', 'requirement']	0.51
['parent_of_requirement', 'requirement', 'requirement', 'parent_of_requirement', 'parent_of_requirement', 'requirement', 'requirement', 'parent_of_requirement']	0.54

rater. For the eight transcribed protocols of [37], six segments were taken out of each protocol with systematic sampling.

Three coders who had previously used the web tool and were familiar with the ontology coded the 48 selected segments. Fleiss's kappa for the three raters was 0.35, which is fair–moderate agreement. A pairwise comparison with Cohen's kappa resulted in 0.41, 0.36, and 0.28 agreements between the pairs. Coding the relations was inherently more difficult because relations were vaguer to describe verbally and often related to entities which happened distant to each other temporally. After removing {hierarchy, intergroup} from the choices, the agreement would become higher: Fleiss's kappa 0.48 for the three raters; Cohen's kappa, 0.56, 0.47, and 0.43. In addition to the three new coders, inter-rater agreement was measured between two of the authors whose agreement was 0.64 which is substantial [46]. Excluding the relationships, the agreement would be 0.75 which is nearly perfect [47].

5.2 Limitations of Coding Protocols With P-Maps. While the P-maps models allowed us to represent a large part of the problem formulation process the designers went through, there were some things that could not be coded. One of these limitations is that specific groundings, or protosolutions, the designer specified cannot be identified using the P-maps model. Though the P-maps model represents the space the designer could have explored, they may not have necessarily explored all groundings in the space.

Another limitation is that the model is designed to be domain independent. While this is a major strength of the model, this also means that without domain knowledge, the different combinations of possible designs that may be generated from the P-maps may contain artifacts, or other entities that may not combine well or at all in reality. This is a specific example of the limitation introduced in the previous paragraph. In order to allow for this information to be entered, the P-maps model would need to allow the designer to specify when two entities cannot be combined into one protosolution.

There is no way to specify whether the children of a parent are both required or if they are disjunctive when interpreting the transcriptions. For example, a device may either require a regular valve or a one-way valve, or both may be required in different parts of the device.

Finally, we found that designers will often specify information about what does not need to be considered in the design space. For example, one designer concluded that, since the device was intended for freshwater use only, salt erosion, oxidation, or any contamination of the materials could be safely ignored. There is currently no clear way to code this information. On the other hand, the model does allow for statement such as "the device should be made out of materials that do not become contaminated and that should be resistant to salt erosion or oxidation."

5.3 Paths of Discovery: Top-Down Versus Bottom-Up Approaches. P-maps offer a variety of analyses modes which we should approach with caution. We presented two major types of analysis which we will extend in our future work. In the top-down approach, we have an idea about the patterns we want to look into beforehand. We specifically are interested in understanding what strategies designers use in early conceptual design, and how they influence creative outcome. We can use outcome-based measures to determine which designs are more creative. Then we can find out which strategies correlate well with more creativity. We have traced a few strategies in the encoded protocols which we did not report here since their occurrences were insignificant in the protocols. We believe that once we collect data in a computer tool (with subjects who are familiarized with our ontology), they will add more detail to their P-maps models than the ones we created from protocols. Then we will find more traces of strategies that might have been missed because of the absence of the tool and ontology.

In the bottom-up approach, we elicit quantitative information from P-maps to correlate them to a measure of creative outcome. The similarity of P-maps to graph networks (see Fig. 3) makes it also possible to define graph-based measures such as cycles and node degrees. Many P-maps variables have been defined but with vector analysis, one may find a few of these variables to be linearly independent. There is not much known about design thinking and problem formulation, and past studies mostly rely on limited observations. Thus, there is potential to find new and surprising results in data that is captured with respect to our ontology; however, as with conventional data mining, we are prone to finding correlations that are meaningless in the context of design.

6 Conclusion

We are interested in understanding differences in designers thinking, especially when they formulate problems at the early stages of conceptual design. To that end, we have defined an ontological framework that can represent design thinking with temporal states. Each state of the P-maps framework shows how requirements, functions, artifacts, behaviors, and issues relate to each other. P-maps support an unprecedented level of detail. They allow multiple disjunctive relations among instances of the same or different entities. They also have a common hierarchical structure with partial orderings.

We showed how we can use a graphical representation of P-maps to understand the process of a designer's problem formulation, through changes in the states of a co-evolving problem–solution space. We also compared different designers' P-maps and discussed how we gained insight into their differences by coding and analyzing collected protocols with the P-maps ontology as a predefined coding schema. Using ASP, we explained how we can formalize and trace instances of strategies that designers adopt. Finally, we explained how we can use data mining methods in search for common patterns among sequences of P-maps data fragments and in the association of variables that define a P-maps state.

We discussed some limitations of the framework and argued that some can be overcome if a computer tool is used for collecting the data where there is no interpretation by a human coder, and instead the user of the tool categorizes the piece of thought into P-maps as he chooses. One interesting thread to follow will involve two variables each of which has two levels: data modality and the coder. Data can be entered in the Formulator tool or expressed verbally. The coder can be an expert human judge or a computer program.

We should add that we have built the web-based computer tool, Problem Formulator, based on the P-maps framework to help us collect problem formulation data on a large scale. We have collected data from a class of student designers. Examples of different types of possible analyses within the P-maps framework were presented throughout the paper. Even though the collected data forms too small a sample for drawing any significant conclusions, the immediate payoff in using the tool has been the fact that data collection has been much faster than protocol analysis. Future work should also include more design problems with different levels of required domain knowledge, and in different domains. One aspect to study is to find variables and patterns that are independent of the design problem. We believe that P-maps open new avenues for empirical research in design. The methods that were mentioned in this paper, such as data mining and model tracing, set the stage for radical changes in the field of design studies.

Acknowledgment

This study was supported by CMMI Grant No. 1002910 from the National Science Foundation.

References

[1] Ericsson, K. A., and Simon, H. A., 1992, *Protocol Analysis: Verbal Reports as Data*, MIT, Cambridge, MA.
[2] Cross, N., Dorst, K., and Roozenburg, N., 1992, "Research in Design Thinking," Proceedings of a Workshop Meeting Held at the Faculty of

Industrial Design Engineering Delft University of Technology, The Netherlands, May 29–31, 1991, Delft University Press, Delft, The Netherlands.
[3] Cross, N., Christiaans, H., and Dorst, K., 1996, *Analysing Design Activity*, Wiley, Chichester, UK.
[4] Valkenburg, R., and Dorst, K., 1998, "The Reflective Practice of Design Teams," Des. Stud., **19**(3), pp. 249–271.
[5] Coyne, R., 2005, "Wicked Problems Revisited," Des. Stud., **26**(1), pp. 5–17.
[6] Dorst, K., and Cross, N., 2001, "Creativity in the Design Process: Co-Evolution of Problem–Solution," Des. Stud., **22**(5), pp. 425–437.
[7] Maclellan, C. J., Langley, P., Shah, J. J., and Dinar, M., 2013, "A Computational Aid for Problem Formulation in Early Conceptual Design," ASME J. Comput. Inf. Sci. Eng., **13**(3), p. 031005.
[8] Thomas, J. C., and Carroll, J. M., 1979, "The Psychological Study of Design," Des. Stud., **1**(1), pp. 5–11.
[9] Cross, N., and Cross, A. C., 1998, "Expertise in Engineering Design," Res. Eng. Des., **10**(3), pp. 141–149.
[10] Atman, C. J., Chimka, J. R., Bursic, K. M., and Nachtmann, H. L., 1999, "A Comparison of Freshman and Senior Engineering Design Processes," Des. Stud., **20**(2), pp. 131–152.
[11] Eisentraut, R., 1999, "Styles of Problem Solving and Their Influence on the Design Process," Des. Stud., **20**(5), pp. 431–437.
[12] Harfield, S., 2007, "On Design 'Problematization': Theorising Differences in Designed Outcomes," Des. Stud., **28**(2), pp. 159–173.
[13] Ball, L. J., Ormerod, T. C., and Morley, N. J., 2004, "Spontaneous Analogising in Engineering Design: A Comparative Analysis of Experts and Novices," Des. Stud., **25**(5), pp. 495–508.
[14] Ho, C., 2001, "Some Phenomena of Problem Decomposition Strategy for Design Thinking: Differences Between Novices and Experts," Des. Stud., **22**(1), pp. 27–45.
[15] Kruger, C., and Cross, N., 2006, "Solution Driven Versus Problem Driven Design: Strategies and Outcomes," Des. Stud., **27**(5), pp. 527–548.
[16] Gero, J. S., and Mc Neill, T., 1998, "An Approach to the Analysis of Design Protocols," Des. Stud., **19**(1), pp. 21–61.
[17] Pahl, G., and Beitz, W., 1996, *Engineering Design: A Systematic Approach*, Springer, London.
[18] Hirtz, J., Stone, R. B., McAdams, D. A., Szykman, S., and Wood, K. L., 2002, "A Functional Basis for Engineering Design: Reconciling and Evolving Previous Efforts," Res. Eng. Des., **13**(2), pp. 65–82.
[19] Maher, M. L., Poon, J., and Boulanger, S., 1996, "Formalising Design Exploration as Co-Evolution: A Combined Gene Approach," Advances in Formal Design Methods for CAD: Proceedings of the IFIP WG5.2 Workshop on Formal Design Methods for Computer-Aided Design, J. S. Gero, and F. Sudweeks, eds., June, Springer, pp. 3–30.
[20] Goldschmidt, G., 1997, "Capturing Indeterminism: Representation in the Design Problem Space," Des. Stud., **18**(4), pp. 441–455.
[21] Goldschmidt, G., and Tatsa, D., 2005, "How Good are Good Ideas? Correlates of Design Creativity," Des. Stud., **26**(6), pp. 593–611.
[22] Goel, V., and Pirolli, P., 1992, "The Structure of Design Problem Spaces," Cogn. Sci., **16**(3), pp. 395–429.
[23] Newell, A., and Simon, H. A., 1972, *Human Problem Solving*, Prentice-Hall, Upper Saddle River, NJ.
[24] Gero, J. S., 1990, "Design Prototypes: A Knowledge Representation Schema for Design," AI Mag., **11**(4), pp. 26–36.
[25] Pourmohamadi, M., and Gero, J. S., 2011, "LINKOgrapher: An Analysis Tool to Study Design Protocols Based on FBS Coding," International Conference on Engineering Design, Copenhagen, Denmark, pp. 1–10.
[26] Anthony, L., Regli, W. C., John, J. E., and Lombeyda, S. V., 2001, "An Approach to Capturing Structure, Behavior, and Function of Artifacts in Computer-Aided Design," ASME J. Comput. Inf. Sci. Eng., **1**(2), pp. 186–192.
[27] Gero, J. S., and Kannengiesser, U., 2004, "The Situated Function-Behaviour-Structure Framework," Des. Stud., **25**(4), pp. 373–391.
[28] Gero, J. S., and Kannengiesser, U., 2007, "Locating Creativity in a Framework of Designing for Innovation," *Trends in Computer Aided Innovation*, N. León-Rovira, ed., Springer, Boston, pp. 57–66.
[29] Goel, A. K., Rugaber, S., and Vattam, S., 2009, "Structure, Behavior and Function of Complex Systems: The Structure, Behavior, and Function Modeling Language," Artif. Intell. Eng. Des. Anal. Manuf., **23**(1), pp. 23–35.
[30] Wölkl, S., and Shea, K., 2009, "A Computational Product Model for Conceptual Design Using SysML," ASME Paper No. DETC2009-87239.
[31] Boden, M. A., 2004, *The Creative Mind: Myths and Mechanisms*, Routledge, London/New York.
[32] Novak, J. D., and Gowin, D. B., 1984, *Learning How to Learn*, Cambridge University, New York.
[33] Novak, J. D., and Cañas, A. J., 2008, "The Theory Underlying Concept Maps and How to Construct and Use Them," Florida Institute for Human and Machine Cognition, Technical Report IHMC CmapTools 2006-01 Rev 01-2008.
[34] Quillian, R., 1966, "Semantic Memory," Ph.D. thesis, Carnegie Institute of Technology, Pittsburgh, PA.
[35] Lehmann, F., 1992, "Semantic Networks," Comput. Math. Appl., **23**(2–5), pp. 1–50.
[36] Dinar, M., Shah, J. J., Langley, P., Campana, E., and Hunt, G. R., 2011, "Towards a Formal Representation Model of Problem Formulation in Design," ASME Paper No. DETC2011-48396.
[37] Danielescu, A., Dinar, M., Maclellan, C. J., Shah, J. J., and Langley, P., 2012, "The Structure of Creative Design: What Problem Maps Can Tell us About Problem Formulation and Creative Designers," ASME Paper No. DETC2012-70325.
[38] Gelfond, M., 2008, "Answer Sets," *Handbook of Knowledge Representation*, F. van Harmelen, V. Lifschitz, B. Porter, eds., Elsevier, Oxford, UK, pp. 285–316.
[39] Ward, T. B., Patterson, M. J., and Sifonis, C. M., 2004, "The Role of Specificity and Abstraction in Creative Idea Generation," Creat. Res. J., **16**(1), pp. 1–9.
[40] Tan, P.-N., Steinbach, M., and Kumar, V., 2005, *Introduction to Data Mining*, Pearson Addison Wesley, Boston.
[41] Shah, J. J., Millsap, R. E., Woodward, J., and Smith, S. M., 2012, "Applied Tests of Design Skills—Part 1: Divergent Thinking," ASME J. Mech. Des., **134**(2), p. 021005.
[42] Vanlehn, K., 2006, "The Behavior of Tutoring Systems," Int. J. Artif. Intell. Educ., **16**(3), pp. 227–265.
[43] Cohen, J., 1960, "A Coefficient of Agreement for Nominal Scales," Educ. Psychol. Meas., **20**(1), pp. 37–46.
[44] Fleiss, J. L., 1971, "Measuring Nominal Scale Agreement Among Many Raters," Psychol. Bull., **76**(5), pp. 378–382.
[45] Gwet, K. L., 2008, "Variance Estimation of Nominal-Scale Inter-Rater Reliability With Random Selection of Raters," Psychometrika, **73**(3), pp. 407–430.
[46] Landis, J. R., and Koch, G. G., 1977, "The Measurement of Observer Agreement for Categorical Data," Biometrics, **33**(1), pp. 159–174.
[47] Fleiss, J. L., Levin, B., and Paik, M. C., 2003, *Statistical Methods for Rates and Proportions*, 3rd ed., Wiley, Hoboken, NJ.