INTERNATIONAL DESIGN CONFERENCE - DESIGN 2004
Dubrovnik, May 18 - 21, 2004.

DESIGN 2004

PRODUCT DESIGN SCHEMATICS: STRUCTURED DIAGRAMMING FOR REQUIREMENTS ENGINEERING

F. A. Salustri and J. Parmar

Keywords: requirements engineering, diagrammatic representation, design problem analysis, concept maps

1. Introduction and Objectives

A clear understanding of a design problem is essential for designing an effective product. That understanding is reflected in a well-specified set of requirements. Many engineering problems can be traced back to extremely simple requirements problems that seem “obvious” in hindsight (e.g. the fuselage of the Sikorsky S-92 helibus was originally designed without a pass-through for wiring bundles to connect cockpit instrumentation to control surfaces in the tail). Such errors often occur early in the design process, but might not be caught until much later, resulting in significant losses. Analysis of detailed requirements, often the focus of many development projects, rarely treats the holistic world-view of the product’s requirements at the coarsest levels – yet it is at the coarse level where many errors occur.

The issue is more that just requirements analysis. It also involves eliciting the requirements and representing them in a way that stimulates the design of effective products, and the establishment of rational cognitive product models in designers’ minds. Also, all the designers must agree that they are using the same (mental) model of the problem and its solutions.

Design requirements must therefore (a) *focus* the designers’ attention on key issues, (b) help ensure that all product stakeholders (including the client) *agree* on the nature of the problem, (c) *stimulate* innovative thinking, (d) *incur* relatively low administrative overhead, and (e) *facilitate* navigation and searching of requirements information.

The authors’ solution, called a *product design schematic* (PDS), is being developed to meet these criteria. A PDS is a diagram rather than a table or textual description. Diagrams can alleviate the sometimes onerous burden of understanding long, linear, text-based descriptions of requirements. PDSs are part of a larger project, called *design schematics*, intended to develop diagramming tools for all the early stages of design. In this paper, however, only PDSs will be discussed. While it will become obvious that software to draw PDSs is very important, that software is still under development. In this paper, we will focus on the PDS format itself and not on the implementation of a PDS tool.

It is essential to understand that the focus here is on stimulating the human designer to recognise key system-wide requirements, rather than focusing on the (partial) automation of detailed requirements management.

2. Method of Development

PDSs fit into Salustri's overall research framework, Artefact-Centred Modelling (ACM) [Salustri, 1995]. ACM distinguishes between models, languages, and theories with respect to products, designers, and design management agents. This limits the possibility of reflection, which can lead to paradoxical situations. (The classic example of the kind of paradox that can arise from reflective systems is the single statement *This sentence is false*.) PDSs fit into this scheme by addressing only requirements of the product and not of the product's manufacture, the designers who create it, or the organisation where the designers work. While this may seem to be a limitation, PDSs are just one kind of *design schematic*. The complete tool set will cover all these sectors of design engineering. More information on design schematics is available in [Salustri and Parmar, 2003].

The authors examined a number of works in requirements specification and design problem analysis (e.g. Dym and Little, Suh, Pugh, and Steward). Additionally, our work was informed by experiences in teaching design and the observation that a simple but systematic approach to requirements engineering helps students to understand the many impacts of their decisions. Again, the key is to stimulate thinking, not to automate. Based on this research, we have set four broad categories of information for requirements specification. These categories, and how we use them in PDSs, are explained in Section 3.

The authors also researched various diagramming techniques, including concept maps, IDEF, Modelica, UML, and others. We found that there are no existent diagramming forms that satisfied the requirements given in Section 1. However, we did find that *concept maps* provide a sensible starting place because: (a) they support learning experiences; (b) they allow diagrammatic representation of knowledge that closely mimics the networked nature of knowledge in the human mind; and (c) they are supported with readily available software packages.

Concept maps are perhaps best known as learning and assessment tools in pedagogy, they have also found use as brainstorming tools in business and management [Novak, 1998]. Its use in design engineering has not been indicated in any of the reviewed engineering literature. However, this is not a serious problem, because we believe, as detailed elsewhere [Salustri and Parmar, 2003] that the early stages of design are intense episodes of *learning*, especially in studying, eliciting, specifying, and reasoning with requirements – a designer is learning about the design problem to be solved.

A concept map is a network graph of nodes denoting concepts in a domain of interest, connected by labelled edges denoting relationships between those concepts. In pedagogy, concept maps are considered good if they highly "cross-linked". That is, a strictly hierarchical concept map is considered an indicator of poor learning (shallow understanding) of a subject, whereas a map that is highly cross-linked is an indicator of good learning (deep understanding). The authors extend this to design engineering by hypothesising that a good concept map of a design problem is an indicator of a deep understanding about the problem – a prerequisite for developing good designs to solve the problem.

However, concept maps are too generic to support design engineering. Beyond the notions of concept nodes and relationship links, there is no further grammatical structure to aid in representing complex knowledge. This is done to keep the tool simple and to support users to adopt whatever conventions best express their knowledge in a general case.

In early design engineering these points remain important. Simplicity is fundamental: a complex tool will not be adopted by practising designers and would burden them unnecessarily. Also, using graphical features to augment text descriptions is a well-known and successful way of creating dense yet meaningful descriptions of knowledge for use by other humans. These points notwithstanding, design engineers are not children, and we can extend the structure of concept maps to provide a richer medium for design schematics in general and PDSs in particular.

3. Results

3.1 PDS Information Types

A PDS represents four categories of requirements information.

Product Characteristics. A *product characteristic* (PC) is a description of what a product must or should *be*. PCs are adjectival forms; e.g. a product being *durable*, *light*, or *safe*.

PCs are similar to, but more specialised than, the *design objectives* of [Dym & Little, 2000] in that they cover only characteristics of the product itself. This helps limit reflection and increase formal soundness. The narrow scope of PDSs for products can be made up by having other PDSs to treat manufacturing, process and workflow, etc.

A PC cannot be a negation (e.g. *the product cannot be heavy*) because one cannot prove a negative.

A PC cannot be a boolean combination of characteristics (e.g. *the product must be light and safe*). Such statements hide causal connections between PCs.

All PCs in a PDS must treat only one level of the product's system hierarchy. For example, the PC *the elevator system's cables must be strong* is inappropriate because the *cables* and the *elevator* are at two different levels of the product hierarchy. Exceptions to this rule arise from external constraints.

PCs have a special feature that distinguishes them: they are inherited by subsystems. For example, if we say *The car is safe*, then we imply that every component of the car is also safe, though the degree by which safety is exhibited by sub-systems can vary. As one proceeds from a product-level design to a subsystem-level design, one must carry forward information regarding the PCs of the product and apply them to the subsystems.

As will be shown below, PCs form the roots of a tree of requirements. As such they can be used to group requirements together and label those groups of requirements.

Finally, there are two types of PCs. *Enabling* PCs are generic and (nearly) universal; for mechanical systems, these include functionality, durability, safety, affordability, usability, fabricability, maintainability, sustainability, and quality. From these derive *special* PCs that are particular to given products. For example, consider "*the ladder is usable by being portable*." *Portability* is a special PC of the enabling PC *usability*. Depending on a designer's perspective, one might relate portability to the enabling PC functionality. One may even relate portability to both usability and functionality.

The relationship between generic and specific PCs can be represented with the phrase *by being* as in the example above. The authors have found that providing these kinds of key phases that improves the ease with which both students and practising engineers learn to use PDSs.

Functional Requirements. A *functional requirement* (FR) is a statement of what a product must or should *do*. It is specified as a verb-object pair forming the predicate clause of a sentence whose subject is always the product. To *carry passengers*, *consume little fuel*, and *protect passengers*, are possible FRs of an automobile. [Dym and Little, 2000] distinguish similarly between design objectives and product functions, and the term *functional requirement* is perhaps best known from Axiomatic Design [Suh, 1990]. However, Suh's FRs cover both PCs and FRs in our framework, as well as other characteristics which we exclude.

FRs in our PDS approach share the same restrictions as do PCs.

The most important difference between FRs and PCs regards the propagation of FRs from a system to its subsystems. FRs are related through the system hierarchy by *composition*. That is, subsystem FRs combine to produce emergent functionality at the system level; e.g. the functions of the components of an automobile (such as the engine) are not necessarily functions of the automobile itself; they are required sub-functions that allow the automobile's functions to emerge through their combination.

Constraints. A constraint (C) is a hard quantitative limit on the degree to which a product exhibits a PC or FR. A constraint is a named variable, specified as an adjectival clause (for constraints on PCs) or an adverbial clause (for constraints on FRs). They have three parts: a name (*weight*, *length*, etc.), a limit specifier (one of minimum, maximum, or optimum) and a value (20 kg, 100 m/s, etc.). While constraints can become far more complex in downstream stages of designing, our focus is on the early stages, where such relatively simple representations are entirely adequate.

Designers often know the name and specifier of a constraint before they know its value. It is therefore useful to distinguish between these parts of a constraint. Constraint names and specifiers can be specified early in the design process, often during the requirements engineering stage, and values can be added later. The importance of early constraint identification is that it can help designers prune design search spaces qualitatively. Early but partial constraint specification can also help design teams identify relationships between PCs and FRs that might otherwise remain hidden or only implied, thus lowering the odds of expensive errors being made early in the design process.
[Dym & Little, 2000] identify constraints, but set them aside during the initial problem analysis stage. Suh treats constraints similarly. [Pugh, 1991] sets a format that incorporates constraints directly. We believe Pugh's approach is best, because integrating constraints within the rest of the problem specification helps ensure that any design alternative will satisfy them. Other approaches, which set constraints aside even temporarily, raise the possibility that either a constraint or an interaction between other problem features arising from a constraint will be overlooked.
Performance Metrics. A *performance metric* (PM) is a quantifiable measure of a product's performance with respect to a constraint. The goal of a PM is to represent the performance capacity of a product in reality. In order to gain some overall sense of which of possibly many alternatives is best, PMs are defined so that a greater positive value indicates a more desirable capacity. Examples are shown in the table below.

Table 1. Examples of performance metrics

	Constraint	Design Value	Performance Metric
Maximal PM	$W_{max} = 1000$ kg	$W = 800$ kg	$PMW = (W_{max}-W)/W_{max} = 0.2$
Minimal PM	$\omega_{min} = 1000$ rpm	$\omega = 1200$ rpm	$PM\omega = (\omega - \omega_{min})/\omega_{min} = 0.17$
Optimal PM	$T_{opt} = 500$ C	$T = 520$ C	$PMT = -\lvert(T_{opt} - T)/T_{opt}\rvert = -0.04$

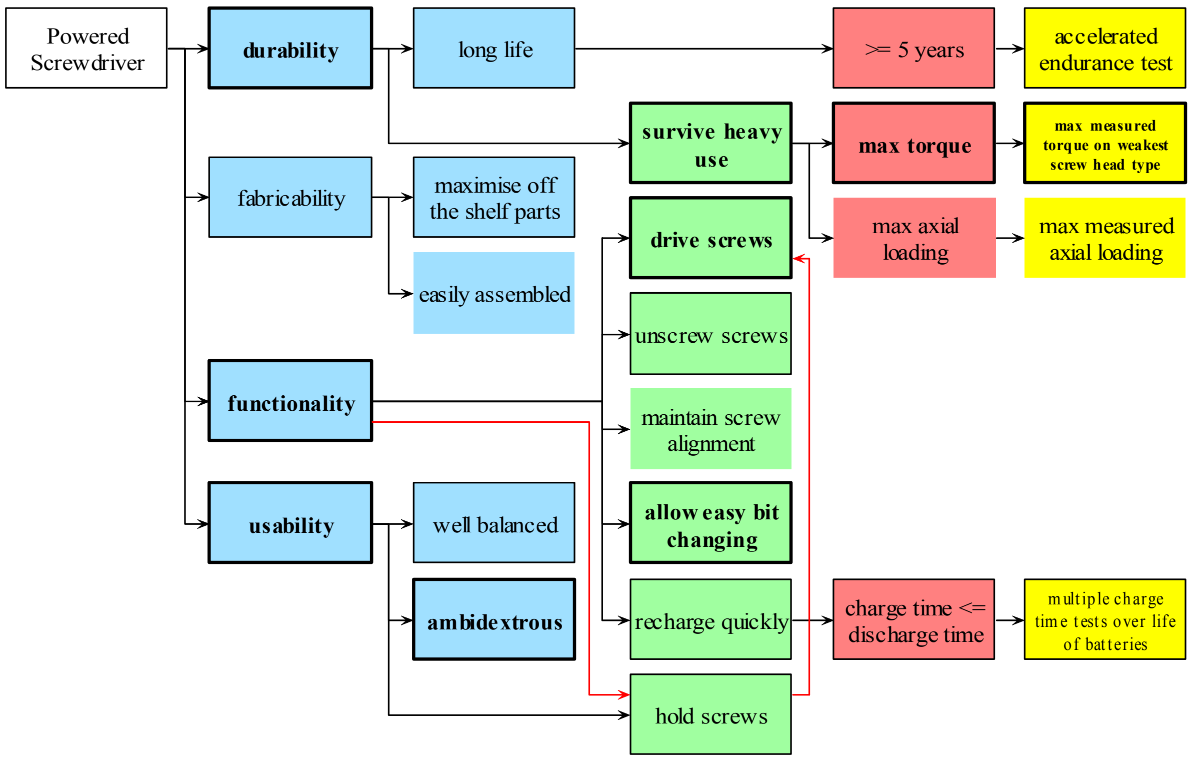

Figure 1. A sample partial PDS for a powered screwdriver

In early design stages, it matters more to know that a PM can be defined than to know its actual value. By knowing how product performance is to be evaluated, test and quality engineers can begin to prepare for the processes they will have to follow, thus helping to shorten time-to-market.
Implementing the principle that the large, positive values are preferred allows one to quickly compare design alternatives by simply scanning tabulated PMs. Normalisation permits one to gain a sense of the "strong points" of a design by finding the most positive PMs. Quantitative tools like weighted decision matrices [Pugh, 1991] can be extended easily to include PMs.
This formulation of PMs applies to measures of performance that are continuously valued. For discrete values, one can assign 1 to cases where a constraint is satisfied and 0 to all other cases, and maintain the principle of preference for the largest positive value of the metric. Other discrete scales, like the Saaty scale, can also be used to evaluate performance with respect to characteristics that are neither quantitative nor continuous.

3.2 Product Design Schematics

The authors believe that requirements engineering is burdened by a dependence on text. We believe that a hybrid text/diagram form can highlight features of design problems that are easily missed in textual descriptions. Most importantly, a "picture" of the problem gives an integrated sense of the problem. The decisions made in the early stages of design are the most crucial, but they occur with the least amount of "solid" quantitative information. To help offset this, it is important to ensure that designers have a deep understanding of the relationships between requirements, which are rarely represented well by linear textual descriptions. Diagrams allow one to view requirements in different ways, which promote clearer and even more innovative thinking about potential solutions.
Based on our framework for requirements specification, the authors have developed PDS diagrams. A PDS is a graph, with PCs connecting to FRs and Cs, and all these types connecting to PMs. A sample PDS for a powered screwdriver is shown in Figure 1. The PDS is incomplete due to space limitations; however, it does show the main features of the diagrammatic form.
The diagram is arranged in rows and columns, like tabular and matrix formats such as the design structure matrix [Steward, 1991] or the product design specifications in [Pugh, 1991]. PCs are shown in blue, FRs in green, constraints in red, and PMs in yellow. Requirements are shown in three *strengths*: essential requirements have thick borders and bold font; requirements considered least important do not have borders; the rest are of middling importance. While PDSs provide no method for designers to set the importance of a particular requirement, they do require that an item of a given importance leads to other items of no greater importance.
Linking between the various nodes with arcs simplifies the representation by eliminating the duplicate entries that appear in conventional tabular formats to represent *cycles*. PCs can connect to FRs or Cs; FRs can also connect to Cs; and Cs can only connect to PMs. Links are directed from left to right.
Colour coding is used to distinguish different types of information. The scheme used here is arbitrary. The key is to use a consistent colouring scheme for all PDSs of a product, because it lets designers identify kinds of requirements information quickly and intuitively. Typically, links are black arrows, but they can be coloured (red in this case) to highlight interactions. Which of all possible links are identified as interaction links can be established from via any of a number of clustering algorithms that group nodes in graphs according to the density of connections between elements. The details of this aspect of PDSs goes beyond the scope of this paper. The point is that a PDS can highlight where information is missing, how functions interact, and the impact of constraints in achieving key product characteristics. For example, it is obvious where constraints and PMs are missing in Figure 1.
PDSs thus provide a simple and concise overview of a design problem, which is essential to develop a good solution. They provide a medium to capture the thinking of designers when they explicate a design problem through requirements specification. By giving "shape" to requirements, they can stimulate the designer to explore issues that might have otherwise remained buried in long textual requirements documents. They also provide a medium to facilitate discussions and collaboration between engineers, and the kind of broad overview that project managers need as well.

3.3 The Importance of Layout

The authors have found that PDS *layout* – the actual arrangement of the graphical entities in a PDS – is a prime determinant of both the amount of information that a PDS can communicate and the speed with which a human can absorb that information. While this is of course generally true for any diagram, it is especially important in design since so much information about products is specified and communicated graphically.

Particular care was paid to the layout of the PDS. The layout in Figure 1 was arrived at after considering a number of alternatives, two of which are shown in Figure 2; *the text in the nodes in Figure 2 is not important*. The authors originally thought that colour-coding nodes in a PDS would be sufficient (Figure 2a), but in demonstrations to engineers and engineering students, this was not enough to allow users to identify obvious features of the requirements set *as a whole* that the PDS depicted. We also experimented with algorithms that would minimise the number of links that crossed over one another, and with circular arrangements of nodes rather like the layers of an onion (Figure 2b).

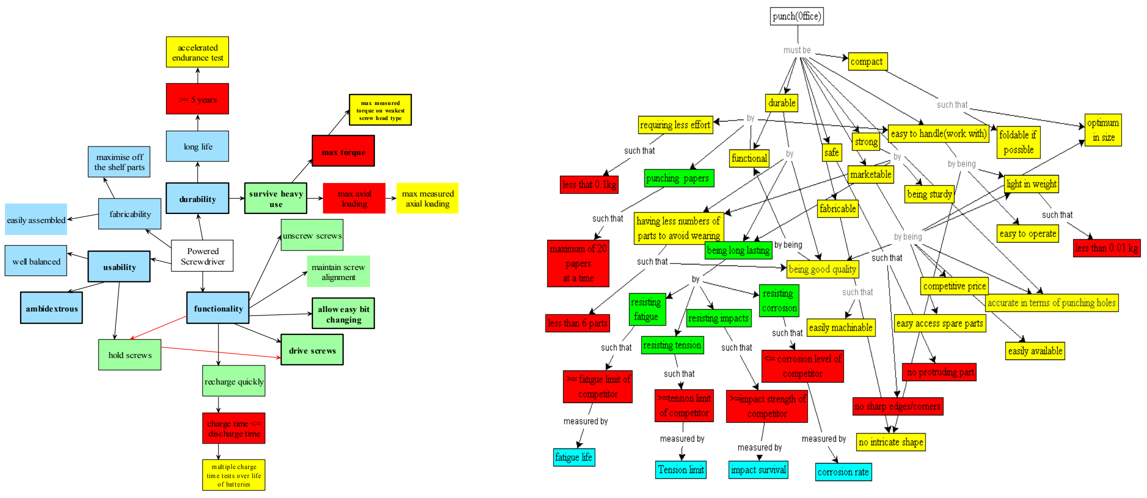

Figure 2. (a) A "random" PDS, (b) a circularly arranged PDS. Neither layout communicates information as well as the semi-tabular layout in Figure 1

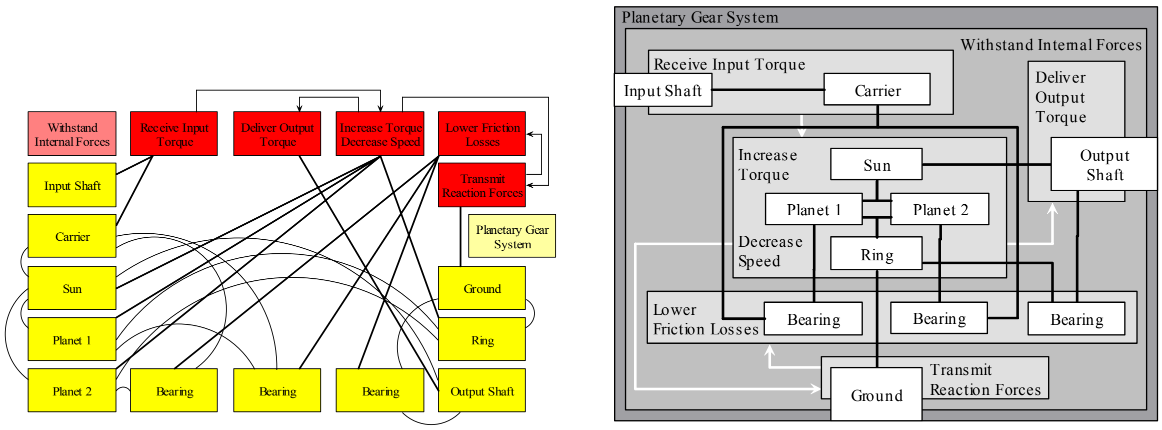

Figure 3. (a) A "random" diagram of the systems, functions, and components of a planetary gear system, (b) The same graphical elements, rearranged according to rules developed by the authors that map engineering relationships to geometrical and topological relations

When presented with such diagrams, however, most people were still unable to extract information easily. Eventually, the authors realised that properly tabulated information can facilitate recognition of

both information *gaps* and information *clusters*. We then developed the “semi-tabular” layout shown in Figure 1. More information on the evaluation of the PDS approach is given in Section 3.5.
Layout is therefore idenified by the authors as an essential characteristic that heavily influences the ability of a diagram to communicate information.
As a final example of the advantages offered by “proper” layout, consider Figure 3, which shows a diagram of systems, functions, and components of a planetary gear system. The colour scheme is irrelevant; the significant factor is the layout of the graphical elements with respect to one another. Again, *the actual text in the nodes is not important*. Figure 3a is essentially a “random” arrangement of nodes. Figure 3b has the same information units present, but some links have been replaced by having nodes overlap. The rules used by the authors to develop Figure 3b are based on mapping engineering relationships to geometrical and topological relationships. For example, interface component (e.g. the *input shaft*) are shown as sharing a border with the node “containing” the whole gear system; and components must overlap at least partially with nodes representing functions that those components help deliver. The authors are currently developing a reference manual of all the mapping rules used in PDSs and other kinds of design schematics such as that shown in Figure 3b. They will be reported in a future publication.

3.4 Software for PDSs

The figures above was laboriously created using a conventional drawing software (Smartdraw). Since the rules for laying out a PDS are simple and deterministic, it is a relatively straightforward programming exercise to create software that will premit the rapid development of tidy, well-organised PDSs. The software will take advantage of the rules given in Section 3.2 for PDSs to present an interface that will be user-centric. For example, the system will allow links to be made only between certain kinds of elements, and new connections will automatically cause the diagram’s layout to be adjusted.
This exercise is not as easy as it seems, however, for two reasons. First, PDSs are just one kind of design schematic. We intend to develop a software system that supports all the different kinds of design schematics in an integrated way. To do this, an *intelligent diagramming engine* will be created. This engine will then be configured at runtime to support each type of design schematic. A single underlying database system will ensure that all kinds of design schematics, including PDSs will be stored uniformly to facilitate data exchange. Second, requirements engineering is a far more complex task than has been suggested here. PDS diagrams are useful for seeing a design problem in the large; to do so requires removing much of the detailed information that is needed to fully design a solution. In this regard, we envision the PDS diagram as a *navigational tool* that, when properly linked to other requirements and project management software will allow designers and project managers to quickly navigate, search for, and control the more substantial detailed information.

3.5 Evaluation of the PDS approach

No rigorous comparisons have been conducted yet between PDSs and other methods of requirements engineering due to the lack of a software platform. Such evaluations would entail experiments with human subjects, since design schematics in general and PDSs in particular are intended to promote better, faster designing by humans. Anecdotal evidence gathered so far has informed the authors that experiments using other media (existent software, white-boards, pencil and paper, and other forms) would not yield useful results in the short term. Our anecdotal evidence includes the following.

- Practising engineers from a large automotive system manufacturer and a robotics company, as well as engineering students, have been shown requirements written as tables, variants of the design structure matrix (Steward, 1991), and PDSs. In most cases, once the form of each specification was explained, subjects were able to extract and voice information about the requirements they were shown more quickly with a PDS diagram than with the other forms.
- PDS diagrams have been used with some success in one author’s senior undergraduate design course. That is, although they reported finding PDS diagrams difficult to construct (again, due to a lack of suitable software) students who used PDS diagrams developed designs that better addressed a broader range of requirements than students who used other methods.

Once the software is available, we will conduct studies to more properly evaluate the PDS approach. Finally, the diagrammatic approach the authors have proposed here does not *require* the information structure described in Section 3.1. The authors believe that PCs, FRs, Cs, and PMs represent a sufficient but not necessary way of structuring requirements information. We envision providing support for other structures eventually, within the diagrammatic environment proposed here.

4. Conclusions

Product design schematics represent a new way to elicit, specify, and reason about design requirements in a holistic way that is especially useful in the *early* stages of designing. PDSs distinguish four different kinds of requirements information; few enough to be easily remembered by users, but rich enough to support properly the requirements engineering stage. By employing a diagrammatic approach, PDSs provide a good overview, thus promoting collaboration and concurrent design practices. A computer based tool is currently under development and will be used to quantitatively assess the PDS approach; in the meantime, anecdotal evidence from industry and teaching settings suggest that PDSs can improve the overall quality of designs.

Acknowledgements

The authors graciously acknowledge the support of the Natural Sciences and Engineering Research Council of Canada, through the Discovery Grant Program.

References

Dym, C.L., and Little, P., "Engineering design: a project-based approach", Wiley & Sons, 2000.

Novak J.D., "Learning, creating, and using knowledge: concept maps as facilitative tools in schools and corporations", Lawrence Erlbaum Associates, New Jersey, 1998.

Pugh, S., "Total design: integrated methods for successful product engineering", Addison-Wesley, 1991.

Salustri, F.A., "An artifact-centered framework for modeling engineering design", Proc. ICED-95, pp 74-79, 1995.

Salustri, F.A., and Parmar, J., "Visualising early product design information with enhanced concept maps", Proc. ICED-03, pp XX, 2003.

Steward, D.V., "Planning and managing the design of systems", Proc. Portland Intl Conf on Management of Engineering and Technology, 1991.

Suh, N.P., "The principles of design", Oxford University Press, 1990.

Filippo A. Salustri, Ph.D., P.Eng., Assistant Professor
Ryerson University, Department of Mechanical and Industrial Engineering
350 Victoria Street, Toronto, Ontario, M5B 2K3, Canada
tel: +1-416-979-5000 x7749, fax: +1-416-979-5265
E-mail: salustri@ryerson.ca