INTERNATIONAL CONFERENCE ON ENGINEERING DESIGN
ICED 03 STOCKHOLM, AUGUST 19-21, 2003

VISUALISING EARLY PRODUCT DESIGN INFORMATION WITH ENHANCED CONCEPT MAPS

Filippo A. Salustri and Jayesh Parmar

Abstract

The authors believe that an appropriate diagramming tool can be of substantial benefit to designers, especially in the early, pre-geometry stages of product development. We find no such tool to exist. We therefore introduce *design schematics* (DS) as such a tool. We outline the general benefits of diagramming and then consider the advantages and disadvantages of some existing diagramming methods. Our analysis of this motivates the development of DS. Several examples demonstrate how DSs can capture important information during early design stages. We are currently developing a computational tool that implements DS and discuss some of the issues we face in this regard. While there is not yet any quantitative data by which DS can be evaluated, there is anecdotal evidence suggesting that the tool has potential to be of benefit to practising designers.

Keywords: visualisation, product model, concept map, non-geometric information

1 Introduction

Designing products is becoming increasingly difficult due to increasing complexity, precision and quality requirements, new materials, environmental and ergonomic issues, etc. Designing is especially difficult in the *early stages* of product development, when geometry has yet to be determined, due to a relative lack of methods to represent and reason about product models. The early stages are those that precede the establishment of product geometry, and are the most important. One may invoke the Pareto Principle to argue that most of the performance of a product is determined by these early design decisions.

Furthermore, engineering as a whole, and design in particular, has become a global endeavour organisationally and geographically. Design information must be made relevant to all aspects of the product development process (PDP) for concurrent engineering and related practices to yield their full benefits. Teams must share information with colleagues, suppliers, and managers who may be physically distant, so it is essential that design information be presented in a clear, concise way. There are cultural issues too: in the "global village", design information written in one natural language can be misinterpreted by recipients not very familiar with it. Lessening the dependence on natural language will aid in the internationalisation of design information and thus benefit culturally diverse design teams.

In order to address these issues, a variety of *graphical* techniques have been developed, such as bond graphs, Integrated Definition Language (IDEF) [1], entity-relationship diagrams, the Modelica language for physical systems modelling [2], and the various kinds of block diagrams. These tools can serve in some aspects of the PDP. However, these tools are best

suited and most readily applied to specifying designs once the overall conceptual and systems design stages have been completed.

This is understandable. In the early design stages, product information is vague and qualitative. Mapping this information into software is very difficult. There are also issues of software usability. While work has been ongoing for many years to develop software that supports qualitative information (e.g. computer-based sketching), such software remains quite expensive and difficult to use compared to a simple sheet of paper and a pencil. The early design stages are heavily influenced by individual experience and preference, and by group dynamics and synergy, which are not captured readily by tools that must by necessity commit to some underlying methodology. Each methodology implies certain perspectives on design and product development that are not necessarily shared by the tools' user community.

The authors believe that a solution lies in the development of simpler tools that rely more on *informal* representations of product and design information. Our hypothesis is that the "right" product-modelling tool for the early design stages will *facilitate* human cognitive abilities rather than *displace* that burden from the human designer to the tool itself. That is, we are looking for a tool that promote clear, structured thinking in modern design environments. We need tools that help the designer think more clearly and about the right things.

A *graphical*, rather than textual, notation can succeed here [3]. It is cliché to say that a picture is worth a thousand words, because it is true. In design meetings, designers typically use diagrams to aid the team's work. These diagrams function as cognitive cues to the designers about important issues, decisions, and actions. Furthermore, appropriately *laid out* diagrams can communicate a holistic perspective of information that is very difficult to extract from purely textual descriptions, thus giving designers a better overview of their work [4].

In summary, the authors believe that it is possible to design and implement a simple tool that will be small, usable, and effective for at least some of the upstream stages of design processes by giving up the requirement that all information must be carefully structured. This paper introduces such a tool, called *design schematics* (DS). It is a diagramming method intended to capture product information in early design stages. While computer support for DS would obviously help, we will focus here on the language of DS itself; nonetheless, some comments regarding the nature of a computer-based DS tool are given at the end of the paper.

2 Fundamentals

Various graphical and matrix-based tools are being investigated, including bond graphs, flow charts, IDEF, entity-relationship diagrams, the Modelica language, electrical block diagrams, Unified Modelling Language (UML) [5], design structure matrices, visual programming languages, and concept maps. Though this background study is ongoing, some points of interest are already apparent.

To the authors, the main disadvantage of all of these methods except concept maps is that they provide well-defined structures to specify product information unambiguously and prepare it for various analyses. This is not in itself bad, but in doing so these tools impose too many constraints on the early design stages, which can stifle innovation. The chief hindrance of concept maps, on the other hand, is their lack of structure, which prevents designers from achieving the concentrated focus needed to treat design issues effectively and efficiently.

DS is being developed to lie somewhere between concept maps on the one hand, and the more structured tools on the other. We are studying each tool to identify features that can be

combined with concept maps to guide designers while preserving the flexibility needed in the early stages of designing.

2.1 Requirements Analysis for Early Design Stages

The authors consider the *early design stages* to include strategic planning, requirements analysis and specification, conceptual design, systems design, and configuration and layout.

By the time that the configuration and layout stage is completed, a sense of physical size and shape is usually available, for which conventional CAD solutions may then be employed. Concept sketches may be possible, but sketching by computer is still an immature technology, seldom used in North American engineering industry. Furthermore, sketches are only able to capture some of the kinds of important information that engineers need to guide their thinking. They do not capture information about requirements, systems, etc. Thus, some new representation is needed. This representation must meet the following requirements.

1. Promote thinking and learning about design problems rather than just specifying solutions.
2. Be extremely intuitive and easy to learn.
3. Be flexible enough to represent different kinds of information consistently.
4. Provide structure to disambiguate kinds of information arising in the early design stages.

2.2 Designing as Learning

The first requirement is the main reason we have chosen concept maps as the basis for DS.

The authors subscribe to the notion that learning is the integration of new information into a learning agent's existent *cognitive structure* [6], which can be seen as a richly interconnected network of information upon which cognitive processes operate. The more richly interconnected new information is, the greater the impact of the new information on cognitive processes of a learning agent, and the better the new information has been learned.

We believe that designing – especially in the early stages – is a learning exercise. The acts associated with specifying the initial requirements, concepts, and systems of a product are acts of exploration and discovery, which result in learning. Once a design problem is understood well enough, at least part of the answer (the design) is usually quite apparent. Indeed, it is a common perspective that design problems and their solutions evolve concurrently. In this view, then, effective design tools promote the integration of new information by being as compatible as possible with the network-based model of learning described here.

2.3 Concept maps as learning tools

Concept maps are learning tool meant to target specifically the mode of learning noted above. They are visual representations of the key concepts in a domain and their interrelationships. Concepts are shown as nodes and relations as labelled arcs connecting the nodes. Mathematically, concept maps that take the form of network graphs have been correlated to both deep learning and innovative thinking [7].

Concept maps have been classically used in early childhood education as both learning and assessment tools. Elsewhere, they are often used elsewhere during brainstorming to capture quickly and easily ideas that are not precisely defined or understood. By using such an informal technique, the emphasis remains on the brainstorming itself (i.e. thinking) and not on the mechanistic aspects of specifying its results.

We have experimented extensively with conventional concept maps for representing product information. We have found that concept maps are just too simple to represent the breadth and scope product information. Some areas requiring improvement include the following.

Layout management. We have found that the layout of nodes in a concept map can communicate substantive information, and that the best layout for a particular diagram depends on the kind of information in the map. Examples of this will be shown below.

Hierarchies. It must be possible to collapse one map into a single node within a larger map; this would allow direct representation of hierarchical levels of information.

Hypergraph support. Generally, concept maps for design will be hypergraphs (e.g. supporting one-to-many links between nodes). This is rarely provided in existing systems.

Extended link labelling. Conventional concept maps allow labels only on nodes and midway on links (i.e. named links). However, we have found that also allowing labels on both the heads and tails of links substantially increases the flexibility and expressive richness.

Filtering. All-inclusive maps can quickly become too cluttered to be understood easily. Since designers tend to focus on specific aspects of designs sequentially, it must be possible to temporarily filter out unneeded information. The capacity to filter out various portions of a map would help maintain the integrity of the map as a whole while presenting only the minimum required information.

Links between links. The relations represented by links can themselves be thought of as concepts; that is, a whole concept map could be developed to explain a relation like *welded to*. Such maps would be obviously pertinent to manufacturing engineers and engineering analysts. However, no existing concept mapping system that we have studied supports this.

Boolean and cardinality constraints. Conventional concept maps do not support boolean relationships between nodes or links, nor do they support cardinality (e.g. that an assembly requires *four* of one kind of part). Clearly, finding a way to augment concept maps to support these features would be advantageous.

3 Design schematics

This section gives an introduction to design schematics through a set of examples. A more complete specification of the language of DS is in preparation; a draft will be available online at http://deed.ryerson.ca/~fil/R/t/ds/. The DSs included in this paper were drawn "by-hand" using SmartDraw. It will become clear that computer support for DS is important. The authors are working on such a tool, but our main interest here is to discuss the diagrammatic language of DS and to indicate its representational strength for early product modelling.

As a sample problem, we use the design of a powered screwdriver (taken from [8]).

To begin, we consider the description of a *usage scenario* (US), which allows a designer to think about how a product might be used. Information to build a US can come from focus groups, consumer reports, marketing information, designers, etc. USs help designers focus on customer needs, how a product will be used, and issues that arise as a result (e.g. safety). A DS for the powered screwdriver might be as shown in Figure 1, which shows three USs. The arcs in USs show the order of actions taken by users. The terminal node at the right end of this kind of DS has an implied feedback link connecting to the initial (leftmost) node. The number *1* at the branch in the *Usage* US indicates that only one of the branches can be fol-

lowed. The links are unlabelled because there is only one kind of relationship in a US; the links indicate the order of steps in a product's use. "PSD" stands for "powered screwdriver".

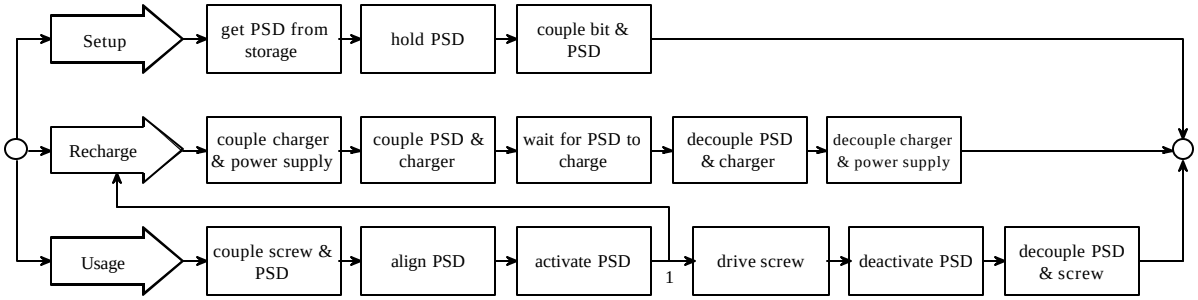

Figure 1. A DS with three usage scenarios for a powered screwdriver.

Designers can then add nodes indicating product characteristics (PCs – things a product must *be*), functional requirements (FRs – things a product must *do*), and constraints that follow from reasoning about the US's design implications. Figure 2 shows the *Setup* usage scenario, with some of these other entities. The branches in Figure 2 are not labelled because the default meaning of a branch is that all the entities at the heads of those links must be attained for the entity at the root of the link to be attained. Blue nodes are PCs, green nodes are FRs, and yellow nodes are constraints.

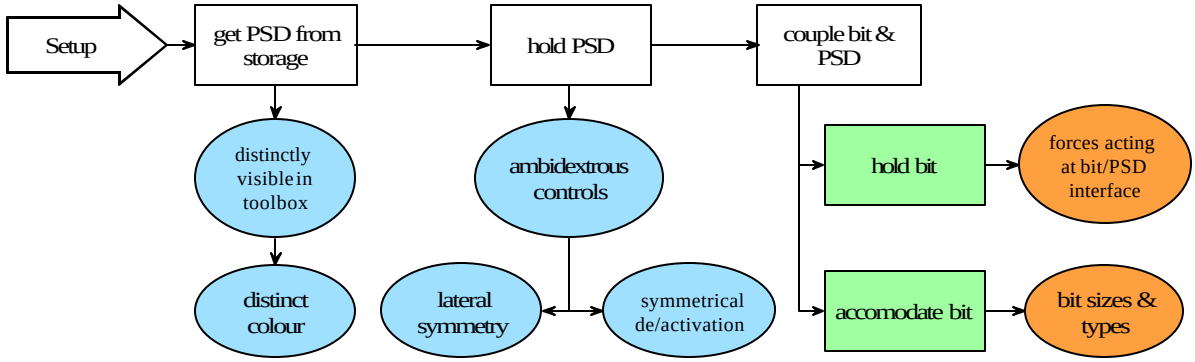

Figure 2. A US augmented with PCs, FRs and constraints.

One can extract all the PCs, FRs, and constraints from a US and, by rearranging them, form a *product design schematic* (PDS) – a DS of the basic characteristics and functions expected of a suitable design. Our PDS representation combines aspects of Pugh's [9] and Dym's [10] work. The details of our representation are not important here and will be presented in future paper. Our goal is to show how a PDS diagram can represent important design information. A *partial* PDS of the screwdriver is shown in Figure 3, for the *Usage* US in Figure 1. The links are unlabelled because the node colours imply the relationships. Bold nodes are of high importance, nodes without borders are of low importance, and the others are of middling importance. The left-most segment of PCs defines some of the basic characteristics of any product. The next segment shows some PCs and FRs specific to the problem. The number of segments varies with the complexity of the problem. General PCs are linked to more specific PCs, which are linked to the FRs that define how a product achieves them. FRs in turn link to constraints that limit a product's performance with respect to the FR. PDSs that form hierarchies are consistent with the Independence Axiom of Suh's Axiomatic Design [11]. However, such uncoupled designs are rare. The highlighted links (in red) in Figure 3 indicate

interactions between PCs and FRs, which appear as *cross-links* in a PDS and which indicate graphically violations of the Independence Axiom. Suh's design parameters have not yet been incorporated into our PDSs; this is a point of future work.

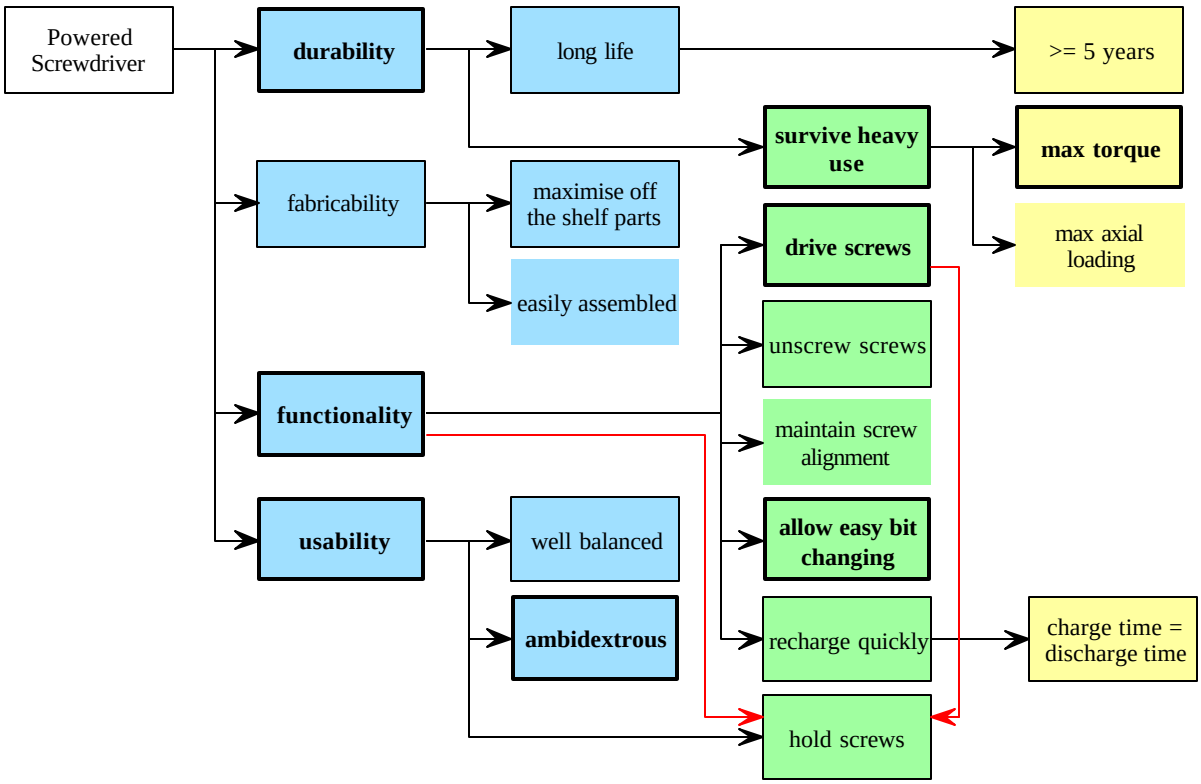

Figure 3. A partial PDS for the powered screwdriver.

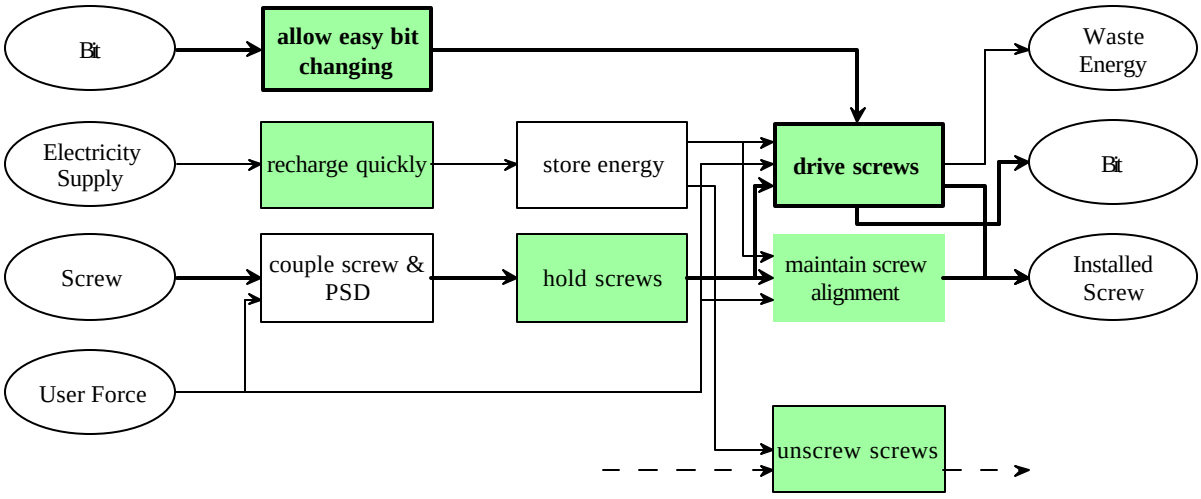

Figure 4. A partial FS for the powered screwdriver.

One may then develop a *function schematic* (FS) of the problem, which is a DS that captures the result of function decomposition. It is similar to the function diagrams of Wood et al [8], and is used to expand the FRs into a function model of the product. A fragment of a FS for the powered screwdriver is shown in Figure 4.

To save space, only some functions from the PDS are shown in Figure 4. Our intention is to demonstrate the DS language and not to provide a complete account of the screwdriver problem. In actual cases it will be important to indicate where and how a DS is *incomplete.* One way is shown in Figure 4: the dashed arrows on the function *unscrew screws* shows that there is more to this FS than is shown. The green nodes are *identical copies* of their counterparts in the PDS. Also, the sides of function nodes have specific purposes. Borrowing from IDEF, inputs into the left side are resources consumed by the function and inputs into the top of a node indicate non-consumed resources (e.g. the bit of the screwdriver).

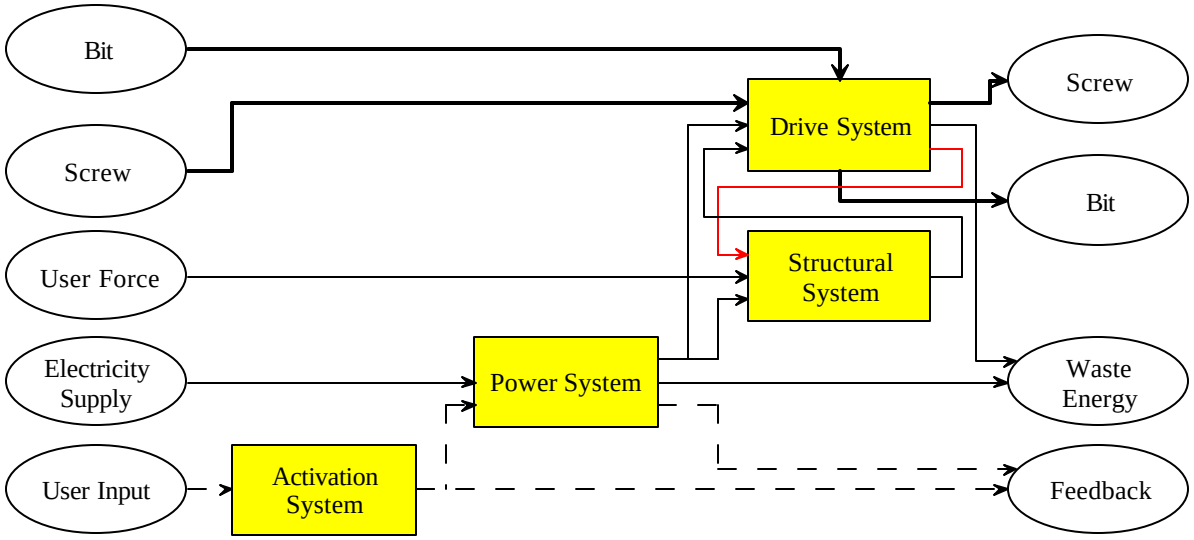

Figure 5. A possible PAS for the powered screwdriver based on the FS in Figure 4.

Finally, one can use the FS to develop a product architecture schematic (PAS). Based somewhat on the diagrams by the same name in [12], our PASs are more structured. Again, the construction rules of the PAS are not as important here as the notion that they represent important design information that can be represented diagrammatically within the Design Schematic framework. A PAS for the powered screwdriver is shown in Figure 5. Note the feedback between the *structural* and *drive* systems (highlighted in red for clarity) is clearly evident by a simple visual inspection of the PAS. This is due to the layout of the diagram, which is designed to highlight such patterns of interaction in the diagram.

4 Computerisation of Design Schematics

Clearly, computer support for the development and manipulation of DSs is important. In this section, the authors will discuss how we are approaching the development of such software. There are three fundamental purposes for the computerisation of DSs.

Computerisation improves usability. The DSs presented in this paper were laid out "by hand" with conventional diagramming software. The task of laying out the nodes and links was onerous because the layout itself communicates important design information. Implementing a computer tool to lay out DSs automatically would solve this problem.

Computerisation improves information integrity. Nodes and links will be represented independent of the style of DS, so that they can appear consistently in multiple DSs and thus communicate information more effectively to designers.

Computerisation increases the potential for design automation. Since a DS carries a certain amount of semantic content, that content can be examined and manipulated by artificial intelligence engines to conduct various design analyses.

The basis of a computerised DS (cDS) system is straightforward and well understood: there are many graph layout engines and software packages available both commercially and for research purposes. The distinction for cDS systems is in the "intelligence" needed to support design-oriented semantic content, particularly for layout purposes.

To address this, the authors are developing an *extensible intelligent graph layout engine*. Within this engine, different *modes* will be available as run-time modules. A mode will have a layout algorithm and various standards for node and link appearances. Each kind of DS will have its own mode. Modes will be able to filter nodes and links so as to present only information pertinent to that mode. It will be possible to implement new modes and simply load them in the engine as "plug-ins".

For example, a *usage scenario* mode will implement a layout algorithm that will ensure (a) the user action nodes will appear organised in the style of Figure 2, and that other nodes will be arranged as well as possible without altering the action node layout. It will provide users with a defined set of nodes and links that can be used in that mode. The user will then be able to switch between USs and, say, PDSs while maintaining integrity of the underlying structure of nodes and links. Changes to one DS will transfer into others as defined by the specific filters and layout algorithms for the corresponding modes.

5 Initial Assessment of Design Schematics

5.1 Comparison to other methods

The authors are unaware of other tools that provide a measure of "intelligence" to graph construction, layout, and manipulation for engineering design applications; we continue to search the literature for other efforts nonetheless. With regards to existing forms of diagrammatic information representation in general, we may make the following comments.

DSs are more structured than general tools like concept maps. However, DSs are also less structured than systems like Modelica and bond graphs. These other languages also do not treat the very early stages of product development, such as requirements specification, whereas DSs are especially intended for them. DSs also share certain characteristics with some other diagrammatic methods, such as IDEF and flowcharts, but integrate them into a generalised framework to support a broader range of applications. This promotes the uniform use of DSs throughout PDPs, which should help improve them.

UML is a diagramming system that has been successful in software engineering. UML has little support for requirements analysis, but it does support other early design stages. However, the authors find the abstract symbols used to represent information in UML confusing in conventional engineering. We do not discount the future development of a uniform system of "glyphs" for non-software engineering environments (similar to the symbols used in electrical circuit diagrams). However, we find short textual descriptions, as annotations on nodes and links in DSs, are amply sufficient in every circumstance we have considered so far. We believe the issues of a symbol system for design and the development of a diagrammatic tool like DS are independent of each other, and will treat them as such.

Another tool related to DS is the *design structure matrix* (DSM) [13]. The morphological similarities between graphs and matrices bears closer scrutiny than can be afforded here; a detailed comparison of DS and DSM methods will be the subject of a future paper. Still, we believe DSs hold four advantages over DSMs.

1. DSs promote learning through direct depiction of graphical networks that reflect the cognitive structures that appear to exist in human minds.
2. More information can be associated directly with the relations between entities. In DSMs, such information is indicated by the contents of matrix cells; usually, this is a simple mark indicating a relation exists, or a single numeric value. In DSs, information about relations can be much more extensive.
3. DSMs tend to be relatively sparse matrices. In many cases, more than half of the cells in a DSM are empty and the space they occupy on the page or computer screen is wasted. DSs use space on the page or screen more efficiently. This makes Design Schematics visually more dense containers of design information than DSMs.
4. The layout of a DS can communicate a more meaningful *overall* structure of the design than a DSM (which is limited by the nature of its matrix form). Also, different layouts of the same DS can communicate different kinds of information; there is no equivalent capacity in a corresponding DSM.

5.2 Experiences in Usage

DS is a tool that is under development and it is, to the best of the authors' knowledge, the first tool of its kind. Therefore, there is no quantitative data available yet for its evaluation. Nonetheless, the authors have used structured diagrams built with existing diagramming software in various teaching and research settings. In a senior undergraduate mechanical system design course taught by Salustri, students use a PDS diagram to capture and reason about design problems. Students uniformly found the exercise "heavy" because they had had no previous exposure to concept maps. However, they also agreed that their appreciation for the complexity, breadth, and depth of a design problem was significantly improved. Students said they "knew the problem cold," and subsequently reported experiencing far fewer problems as they designed than they did in other assignments where diagrams were not used.

In research settings, the authors have found that DSs help clarify thinking, document activities, and collaborate in teams. We have used DSs and DS-like diagrams to study automotive drivetrains, small appliances and tools, and small housing structures, as well as to manage research projects. (These experiences will be the subject of a future paper.)

Though such demonstrations are unscientific, they do provide some anecdotal evidence to suggest that DSs can be beneficial in design environments.

6 Conclusions

The authors have introduced a new tool to assist designers, *design schematics*, which is based on concept maps, augmented with certain features from other diagramming systems. The focus is on the nature of the representation and not on the admittedly significant computational aspects. Several examples are given to indicate how DSs can capture meaningfully information during the early stages of product design, when (a) little or no information about product geometry is available and (b) decisions have the greatest impact on product performance. Since DS is still under development, there is no quantitative data by which to evaluate

the tool. However, some anecdotal evidence gathered by the authors suggests that the tool does have potential to be of benefit to practising designers.

Acknowledgements

The author acknowledges the support of the National Sciences and Engineering Research Council of Canada for funding the work reported herein.

References

[1] – "Integration Definition for Function Modeling", NIST Defence Technical Information Center, 12/21/93, 1993.

[2] Elmqvist H., Mattson S.E., and Otter M., "Object-Oriented and Hybrid Modeling in Modelica", J. Europeen des Systemes Automatises, Vol, 35, 2001, pp. 1-10.

[3] Kulpa, Z. "Diagrammatic representation and reasoning", Machine Graphics Vol. 3, 1994, pp. 77-103.

[4] Marshall M.S., Herman I., and Melançon G., "An object-oriented design for graph visualisation", Software Practice and Experience, Vol. 31, 2001, pp.739-756.

[5] Booch G., "The Unified Modelling Language User Guide", Addison-Wesley, 1999.

[6] Novak J.D. and Gowin R., "Learning how to learn", Cambridge University Press, Cambridge, 1984.

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

[8] Stone R.B., Wood K.L., and Crawford R.H., "A heuristic method for identifying modules for product architectures", Design Studies, Vol. 21, 2000, pp.5-31.

[9] Pugh S., "Total design", Addison-Wesley, London, 1991.

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

[11] Suh N.P., "Principles of Design", Oxford University Press, 1990.

[12] Ulrich K.T. and Eppinger S.D., "Product design and development", McGraw-Hill, New York, 1995.

[13] Steward D.V., "The design structure system: a method for managing the design of complex systems", IEEE Transactions on Engineering Management, Vol. 28, 1981, pp.71-74.

Corresponding author:
Filippo A. Salustri, Ph.D., P.Eng.
Department of Mechanical, Aerospace, & Industrial Engineering
Ryerson University
350 Victoria Street
Toronto, ON
M5B 2K3 Canada
Tel: 416-979-5000 ext 7749
Fax: 416-979-5265
Email: salustri@ryerson.ca
URL: http://deed.ryerson.ca/~fil/