*Int. J. Engng Ed.* Vol. 19, No. 2, pp. 260–271, 2002
Printed in Great Britain.

0949-149X/91 $3.00+0.00


# Teaching Creativity in a Technological Design Context*

KEES VAN OVERVELD, RENÉ AHN, ISABELLE REYMEN and MAXIM IVASHKOV
*Stan Ackermans Institute, Centre for Technological Design, Eindhoven University of Technology, PO Box 513, 5600 MB Eindhoven, The Netherlands. E-mail: k.van.overveld@wxs.nl*

*We want to teach creativity techniques to prospective technological designers in a domain-independent way. To facilitate this, we adopt a format and nomenclature that is close to the terminology used by engineers. Central notions are concepts, attributes and values. A crucial role is played by, what we call, productive attributes: attributes that come with a set of values that can easily be enumerated. In this paper we show how this format supports several creativity techniques and how it allows engineers to explore option spaces in a structured manner. We briefly discuss some practical experiences with our approach.*

## INTRODUCTION: PROBLEM STATEMENT

THE STAN ACKERMANS INSTITUTE (SAI) is a subsidiary of Eindhoven University of Technology [1]. It has been established in order to organise various postgraduate programs in advanced technological design. In order to enrol in these programs, candidates possess a completed masters degree in some technical discipline; after successful completion of an SAI program, they receive the title MTD (Master of Technological Design): a degree that is certified by the Dutch Institution of Engineers (KIVI).

The current SAI programs specialise each in one technological discipline: software design, information and communication technology, mechatronic design, product and process design, and others. The intention of these programs is to educate prospective designers in skills and attitudes that should be complementary to the mere technical, discipline-related topics that are commonly taught in graduate programs in science and technology. These skills and attitudes include communication, process management, and business orientation, as well as skills related to *designing*, like creativity, modelling and decision-making. In 2000, SAI has started efforts to teach design skills in an inter-disciplinary, i.e. *domain-independent* manner. One possible benefit of such an approach is that designers in various disciplines would be capable of speaking the same language. In particular in early design stages, where crucial decisions are often being made that cannot easily be justified from within one technical discipline, designers with various backgrounds have to collaborate. A common vocabulary helps to avoid confusion, delay and inefficiency.

* Accepted 24 May 2002.

## ANALYSIS

We focus on the following problem: 'How to teach creativity techniques in interdisciplinary design for post graduate technological design programs'.

Even when dealing with design problems that offer ample opportunity for an innovative approach, novice designers and students in technological design very often fall back to conventional, and frequently sub-optimal textbook-type solutions. Few students consider using creativity techniques. Even those that do so, rarely use more advanced methods than brainstorming (with the possible exception of TRIZ, [2]). Despite much literature on creativity (see below), technological designers rarely use creativity techniques. This may relate to the fact that creativity techniques are relatively unknown to technological designers. However, a more fundamental problem may be that the existing creativity techniques are not appropriate for use in a technological context due to a mismatch between the playful attitude that belongs to for example, metaphorical thinking or mind mapping, and the more formal style of argumentation that is taught in most forms of scientific education. In this paper we hope to tackle this problem. We believe that our approach has the additional benefit that it encourages precise communication, helps to be explicit about assumptions, and provides additional structure during design assignments. The format that we propose:

- invites explication of the concepts and assumptions used in design processes;
- invites structured searching for alternatives;
- matches with the line of thinking that is familiar to technologists;
- deals with technical and non-technical issues using the same vocabulary;
- uses no other terminology than the most

common terms from logic and undergraduate math;
- interfaces well with existing technological methods and formalisms (e.g. physics, computer science, mathematics).

Interestingly, we find that the use of such a format does not only help us to introduce creativity techniques to engineers, but also helps us to reflect on these techniques, and to improve and extend them in several ways.

## RELATIONS TO EARLIER WORK

Creativity research has been performed all over the world, as shown in [3]. Creativity has also been studied from several viewpoints, like a psychological viewpoint [4, 5] or an educational viewpoint, like [6, 7 or 8]. Also the relation between creativity and designing (engineering) has received attention, like in [9–14]. Our approach differs from the general literature about creativity in the sense that we use notions and a terminology that can be understood by engineers in several design disciplines.

We can position our work as follows. According to Amabile [15] there are three basic ingredients to creativity:

- domain skills
- creative thinking skills
- intrinsic motivation.

In our work, we focus on creative thinking skills. Creative thinking skills according to [16] include seeking novelty and diversity, being independent, being persistent, and having high standards. In this taxonomy, our work regards seeking novelty and diversity. Our work, as in [16, 17], tries to foster creativity in an engineering curriculum. This is however, not easy. According to [18] it can be both daunting and time-consuming. We hope to mitigate some of these problems by stimulating students to unify creative exploration with the pursuit of clarification and structure. We think that this can be accomplished by borrowing ideas from data modelling in object-oriented software design [19]. For brevity's sake, we refer to [20, 21] for further relations between our work and earlier literature.

## PROPOSED APPROACH

In teaching creativity techniques to our students, we try to convince them that these techniques lead to useful results that can be incorporated within a technically oriented design and development culture. To do so, we have adapted these creativity techniques, related them to technical concepts, and translated them into formal and technical terms that are close to an engineering culture. This has the added benefit that the structure of the design problem comes out more clearly, and that the relations between the various techniques are more transparent.

We formulate a **creativity technique** as follows. We consider a domain of interest (for instance: solutions to a design problem; stakeholders; risks; use cases; . . . ). This domain, which we call the **option domain**, is characterised as follows:

- it is large;
- it has no a priori metric (structure);
- it contains an element S ('start') that can be easily found according to standard thought patterns ('textbook methods');
- it contains an element T ('target') that is to be considered as highly relevant, highly advantageous, highly beneficial . . .

Most often, $T \neq S$. Creativity techniques are techniques that increase the chance for getting from S to T.

According to most writers in the field, one of the reasons why it is difficult to get from the start S to the target T, is that our brain tends to follow known pathways, and once S is found, it requires considerable effort to deviate from S. This inspires to randomised techniques such as brainstorming. We agree with this observation, but we think that, in particular in technological context (where often quite some structure exists, albeit not obviously visible) randomness is only part of the solution: we think that exploiting the structure is at least the other half. We believe that the format that we propose allows us to express and exploit this structure.

To help students to explore the option domain, we propose a number of different techniques, which we present to the students within a formal framework that is inspired on developments in object oriented software engineering.

*Use of concepts*

In our format we use a common term, 'concepts', to refer to both points and regions in the option space. Information about concepts can be expressed through the 'attributes' of a concept. Attributes are functions, that, when applied to an option, yield a certain value. As designers, we add information, when we constrain the value of an attribute for a certain concept.

Symbolically, we write this as:

$$A(C) = V, \text{ e.g. shape(hole)} = \text{square}$$

$$A(C1,C2) = W,$$

$$\text{e.g. distance(pump, valve)} = 12\,\text{cm}$$

Values can also be tuples, sequences, etc. For example:

$$A(C) = \langle V1,V2 \rangle \text{ e.g. location(station)}$$

$$= \langle 51^\circ 3' 12'' \text{ lattitude}, 34^\circ 2' 12'' \text{ longitude} \rangle$$

In our view the attributes are used to represent *relevant* knowledge about a certain concept. In

design situations, we very often have the case that attributes are applied to concepts that denote large regions of the option space, rather than distinct unique elements. As a consequence, the value of these attributes can be underdetermined. For instance, if we are designing a vehicle, we may have an attribute 'energy source' that is underdetermined as long as we haven't made a particular decision:

energy source (vehicle)

$\in$ {electricity, petrol, diesel}.

The value of an attribute may be one of a set of alternatives (as in the above example of energy sources); it may be a procedure, or it may be a concept in its own right. For instance, in the context of designing, say, the ICT-infrastructure of a building, a first design session may result in the conclusion that the most relevant concept is 'IT infrastructure'. A first attribute can be 'contains( )', with 'contains(ICT-infrastructure) = $\langle$users, computers, management$\rangle$'. A next design session may reveal that users, a concept in its own right, come in types, with 'types-of(users) $\in$ {staff, customers}', where 'staff' is a next concept. This concept 'staff', according to the design team, has as relevant attributes 'number', 'training level' and 'desired service'. The types of the values are, respectively, one of a set of known values (namely, the positive integers); an operational procedure (namely, a test to assess the training level); and a next concept in its own right. And so on.

Notice that the stepwise refinement of concepts entails an invitation to be more and more explicit, until no further new concepts occur, and everything is sufficiently operational. This is reminiscent to the way data models are constructed using inheritance, attribute relations and refinement in object orientation [19].

Having introduced the above terminology, we subsequently use this format to acquaint our students with various creativity techniques. The used format allows the students to express the progress that they make during design sessions, and cannot only be used with the more traditional creativity techniques, but also with variations on such techniques. In the next section we illustrate how we can introduce various techniques through examples and exercises using our format.

## CREATIVITY TECHNIQUES

*Use of brainstorming*

Brainstorming, together with brain writing, and other collective idea-generation techniques, are the best known (and for many designers: the *only* known) methods to creatively explore the option domain. A model of brainstorming, in our terminology, is a process of random sampling the domain, scattering it with concepts. Apart from some problems, like the occurrence of clustering and various forms of social censorship, there are two main problems with traditional brainstorming.

If we call the number of explicitly formulated degrees of freedom the 'dimension' of a sample (a *sample* here is a suggested concept), then most samples will be 0-dimensional. Of course, the option domain has many degrees of freedom (although these are not formulated explicitly—in fact, they are often not known in the beginning), and with brainstorming, we are therefore attempting to cover a high-dimensional domain with 0-dimensional samples. This is likely to be inefficient. Here, a *degree of freedom* is an attribute that can take on multiple values. For instance, in a brainstorm about various means for transportation, one can suggest 'a diesel train'. The value 'diesel' is a value for the attribute 'energy source', applied to the concept 'train'. A 'diesel train' is then a 0-dimensional sample, whereas 'a train with some given source of energy, namely diesel, electricity or steam' is a 1-dimensional sample. We give further examples later.

Most authors agree that some form of clustering is recommended afterwards, as in [6], but in general no real techniques are suggested to do this. In fact, clustering amounts to replacing some samples by a single one, which destroys information and reduces the number of samples even further. The option domain is covered even less dense. In terms of covering efficiency, this seems a bad idea.

*Dimension lifting*

A first remedy to the problems with brainstorming, outlined above, is deliberately searching for samples with dimension higher than 0. We give two examples. First a more academic one, of the kind often used in warming-up events for creativity sessions:

> Suppose you are the managing director of a production plant of sugar cubes. You have a large supply of 5 million sugar cubes, but suddenly the market collapses. How can you avoid to go bankrupt?. Most often, suggestions such as 'convert the sugar cubes to confectionery', or 'dissolve the sugar cubes to produce sugar syrup' are soon proposed. At this point, one may observe that these suggestions are different with respect to the aggregation state: confectionery is solid, sugar syrup is liquid. But if the aggregation state is an attribute, its values are easily enumerated: solid, liquid, and . . . gas! So what about gas-like substances, for instance spray-cans with castor sugar or candy-floss. Also, a suggestion might be: 'use sugar cubes as bricks for temporary buildings. This makes use of the physical properties of sugar: it is nearly incompressible. But 'physical' is a value of the attribute 'type of property used for this application'. Other values are 'geometrical', 'chemical', 'biological', 'psychological', 'social', and 'economical' (we simply took items from a list of academic disciplines). So we have now found two (largely) independent attributes, (or parameters or dimensions) that can be used to characterise the solution space: the aggregation state and the type of

property used for the application. Of course, in a next step we can set about to find applications with all combinations: 'can we think of a gaseous sugar product with mainly psychological effect?' and so on.

Our second example is less academic:

> Suppose we are designing a mechanism, and we have to move a certain part from A to B. One suggestion could be to use a hinge; a second suggestion is to use a glider construction. We can unify these 0-dimensional samples by introducing the higher dimensional sample: 'a motion that is generated by an affine transformation'. Indeed, affine transformations are rotations, translations, mirror reflections, scaling, skewing, and the identity. The suggested hinge and glider categorise in the first two classes, respectively. A mirror reflection would inspire to a balanced counterweight construction; a scaling corresponds to pneumatically inflating a balloon with the moving part being stuck to it; skewing would amount to a parallelogram construction (maybe using elastic deformation), and the identity amounts to questioning if the part has to move at all. So we see that dimension lifting here brings four additional suggestions 'for free'. (In fact, a skilled designer occasionally may find himself applying dimension lifting. We don't claim this to be a new method; we merely identify it as a useful pattern, see also [22]). Also, many taxonomies as published in designer's handbooks are in fact examples of dimension lifting—except that the unifying degree of freedom is often not mentioned.

We now describe dimension lifting in our terminology:

> Suppose that an initial brainstorm has produced concepts C1 and C2. When we consciously want to attempt dimension lifting, we search for an attribute A that applies to both C1 and C2: A(C1) = V1, A(C2) = V2. In the mechanical engineering example, the attribute was 'the affine transformation that corresponds to motions C1 and C2, respectively'; the two values V1 and V2 were 'rotation' and 'translation'.

Now an attribute A, applied to concept C is called *productive*:

- if it gives relevant information about C;
- if it produces only one value for any given C (that is, A(C) is a genuine function of C);
- if the set of values that A can produce are easily enumerated;
- if, in some sense, completeness of the set of values of A can be seen.

It sometimes happens that an attribute in itself is not productive, but that it can be made productive without much effort. For instance, consider an exercise to find new products for a production unit for paper objects:

> Assume that two concepts have been proposed: kites and mail envelops. One attribute that seems to be a candidate for dimension lifting is 'target customers'. Indeed, kites are for children, and mail envelopes are for adults. In our jargon, we write 'target customers (kites) = children' and 'target customers (mail envelops) = adults'. The attribute 'target customers( )', however, is not immediately productive: we cannot be sure that we can enumerate all categories of customers. It has a more complex structure, because it is a concept in its own right. This means that we can interrogate 'target customers( )' by introducing additional attributes, for instance 'age', or 'amount of money to spend', or 'cultural background'. The first two are productive; the latter again is a concept in its own right and may require further attributes in order to become operational or even productive. (The process of introducing further attributes to make the information contents of a concept explicit, given the purpose of the designer, is called *operationalisation*.) The two attributes 'age' and 'amount of money to spend' are productive. Age for instance gives rise to the set of values '0–2 years', '2–5 years', '5–12 years', '12–20 years' and '20 years and over'.

Returning to our initial problem, we can see suggestions for paper products being generated such as:

- 'diapers' (for the 0–2 years category);
- 'toddler's napkins' (for the 2–5 years category);
- 'kites' (one idea that we started with, for the 5–12 years category);
- 'fancy coloured CD self-adhesive labels' (for the 12–20 years category);
- 'mail envelopes' (the other initial idea, for the >20 years category).

A similar exercise for the productive attribute 'amount of money to spend' is left to the reader.

We summarise our dimension lifting method:

- start with a regular brainstorm to get some 0-dimensional samples;
- search for a productive attribute that applies to at least two of these samples;
- if a potentially suitable attribute is not yet productive (because it is a compound concept in its own right), operationalise it with further productive attributes;
- generate the values for the productive attribute;
- use these values to generate new concepts.

This is one reason why productive attributes are interesting: they serve as *generators* for concepts rather than concepts proper. A second, even more relevant reason is, that they can serve to partition the option domain, which is a more powerful way to explore it than sampling. We study partitioning in the following two subsections.

*Hierarchy*

In a metaphorical sense, we can view the option domain P as a high-dimensional manifold with a complicated topological structure. With brainstorming, we sample this manifold with 0-dimensional points; with dimension lifting we have slightly improved our efficiency by sampling it with 1-dimensional 'curves' or even higher dimensional 'surfaces' or 'hyper surfaces'. An attribute then plays the role of a *coordinate*. But we are still sampling, and not covering the option domain. In general, there is no guarantee that we will be able to cover the entire manifold. (Compare this with the observation that also in geometry, we

are in general not able to use one coordinate system for an entire manifold: we typically have to split the manifold in sub-manifolds where every sub-manifold has its own coordinate system. Special care has to be taken to make sure that in the seams, consistent coordinates can be found.) That is because the productive attributes are not likely to apply to *all* concepts. In this and the following technique, we attempt to search productive attributes that are indeed capable to cover the entire manifold, and that therefore can produce a *partition* (instead of a sampling) of P.

We start again with a regular brainstorm, and we obtain a list of concepts, C1, C2, C3 . . . Next we search for a productive attribute A, which applies to *all* these concepts, but which yields different values for some of them.

We can now partition our option space P with respect to the different values of $A(C_i)$, the *distinguishing* attribute. For instance, suppose that the option domain (partition P) contains transportation methods. The initial list of concepts, obtained from brainstorming is, say, 'freight train', 'bicycle', 'walking', 'jumping', 'conveyor belt', and 'taxi'. To create a hierarchy, we need to come up with an attribute (a distinguishing attribute) that is applicable to everything in P, and which serves to split P in two or more sub-partitions.

In this case, a possible distinguishing attribute, that is operational but does not produce one unique value for all concepts in P, could be: 'the type of load( )', with values 'goods' and 'passengers'. (We ignore for now the other two values, 'both goods and passengers', and 'neither goods nor passengers'.) The 'goods'-partition, say P1, contains 'freight train', 'conveyor belt', and the 'passengers'-partition, P2, contains 'bicycle', 'walking', 'jumping', and 'taxi'.

Notice that the partitions P1 and P2 are *represented* or *inhabited* by sets of concepts, namely the concepts that came about in the original brainstorm. The partitions now also *become* (abstract) concepts in their own right. So P1 *is* the abstract concept 'methods to move goods', inhabited by the brainstorm concepts 'freight train' and 'conveyor belt'. Similar P2 *is* the abstract concept 'methods to move passengers', inhabited by 'walking', 'jumping', etc. In general, this method generates *abstract* concepts (corresponding to *regions* in the option space), where every abstract concept has a number of operational attributes with unique values and others with multiple (i.e., underdetermined) values.

We can now iterate this process on one of the resulting partitions, using one of the attributes with multiple values. For instance, we can distinguish between a freight train and a conveyor belt on the basis of the attribute 'mobility of the device'. In this way, we can create partitions P3 and P4 that contain the conveyor belt and the freight train, respectively.

Of course, when we iterate this process, for instance to subdivide P2, we are now allowed to introduce attributes that are applicable in P2 only. An attribute that is only operational for P2, is 'role of passengers during transportation', with values 'active' and 'passive'. This attribute does not produce one and the same value for all concepts that are sub-concepts of P2, but it is operationally defined because P2 only deals with moving passengers. Applying this attribute gives rise to the two subclasses (partitions) in P2, one (P5) being represented by the concepts from our original brainstorm 'bicycle', 'walking', 'jumping', the other one (P6) being represented by 'taxi'. The first three are now representatives of the concept P5: 'methods to move passengers actively'. The 'taxi' is a representative of a new partition, P6, 'methods to move passengers passively'.

Notice that the most important thing that happens here is that we build a hierarchy from partitions; every partition is *represented* or *inhabited* by a subset of concepts from the initial brainstorm, but in virtue of explicitly identifying the operational and unique attributes, we also promote every partition into a concept in its own right. These (abstract) concepts form an added value: they are a *structural hierarchy* on the option domain.

The added value of such a structure becomes apparent, in cases where some of the partitions turn out to be empty.

This may well indicate that we have overlooked certain possibilities during our initial brainstorm.

The resulting hierarchy is one of increasing abstraction. A concept P is *more abstract* than a concept P1 if every attribute $A_i$ that is operational for P, is also operational for P1 and yields the same or more restricted values for P1, while there is at least one attribute $A_j$ that is operational and unique on P1 which is not unique (= underdetermined) on P. $A_i$ is called the *distinguishing* attribute. The different values that are yielded give rise to specialisations of P, to be called *sub-partitions*. Notice that this hierarchy is comparable to the 'is-a' (i.e. inheritance) hierarchy that is used in object orientation. Schematically, we depict the process as follows:

For the example at hand, the hierarchical partition looks as in Fig. 2. (Notice we have not fully elaborated the structure. In a complete structure, no abstract partitions exist that are inhibited by more than one item. Indeed, there is no need to introduce an additional attribute if no items need to be distinguished.)

In order to ensure that the hierarchical structure is not too much dependent on the coincidental outcome of the initiating brainstorm, we may want to work with two teams—both build the hierarchical structure—and next attempt to classify the concepts of the other team into their own structure. This avoids a bias towards setting up a structure that is too much dedicated to the current set of brainstorm concepts.

If we commit ourselves to the following rules:

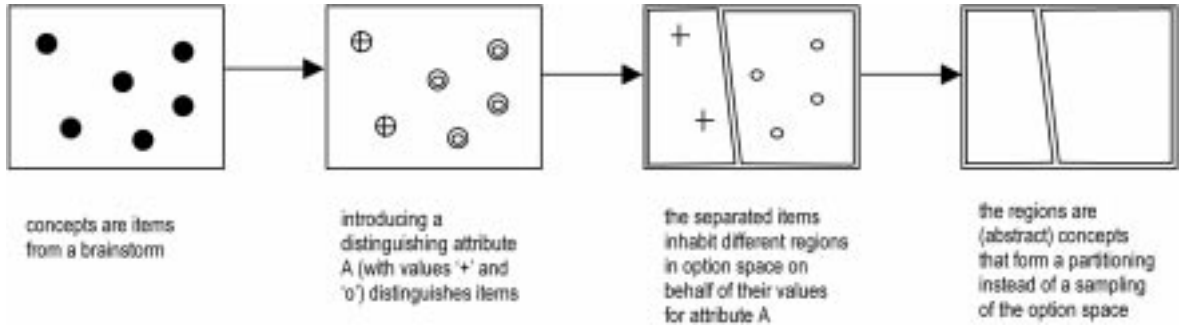


Fig. 1. The process of building a hierarchic partitioning of abstract concepts.

- All attributes are operationally defined for the entire partition of the option domain that they should work upon.
- All attributes should be complete, in the sense that the resulting values span all possibilities for that attribute (if this is not attainable, we can always include an auxiliary value, 'others').
- All attributes should be true functions, in the sense that a concept produces only one value. If this is not attainable, we can usually solve this by simple combinatorial arguing. For instance, an attribute 'usage' with values 'serious usage' and 'fun usage' can return both values when applied to some concepts. A remedy is here to make it a four-valued attribute: 'serious usage', 'fun usage', 'both serious and fun usage', and 'neither serious nor fun usage'.
- Partitions at the lowest level contain precisely one concept.

Then we can be assured of the following:

- the hierarchic structure is a structure on the entire option domain;
- if we decide to reject a partition corresponding to a particular value V1 of a certain attribute A1, all sub-partitions of that partition will be

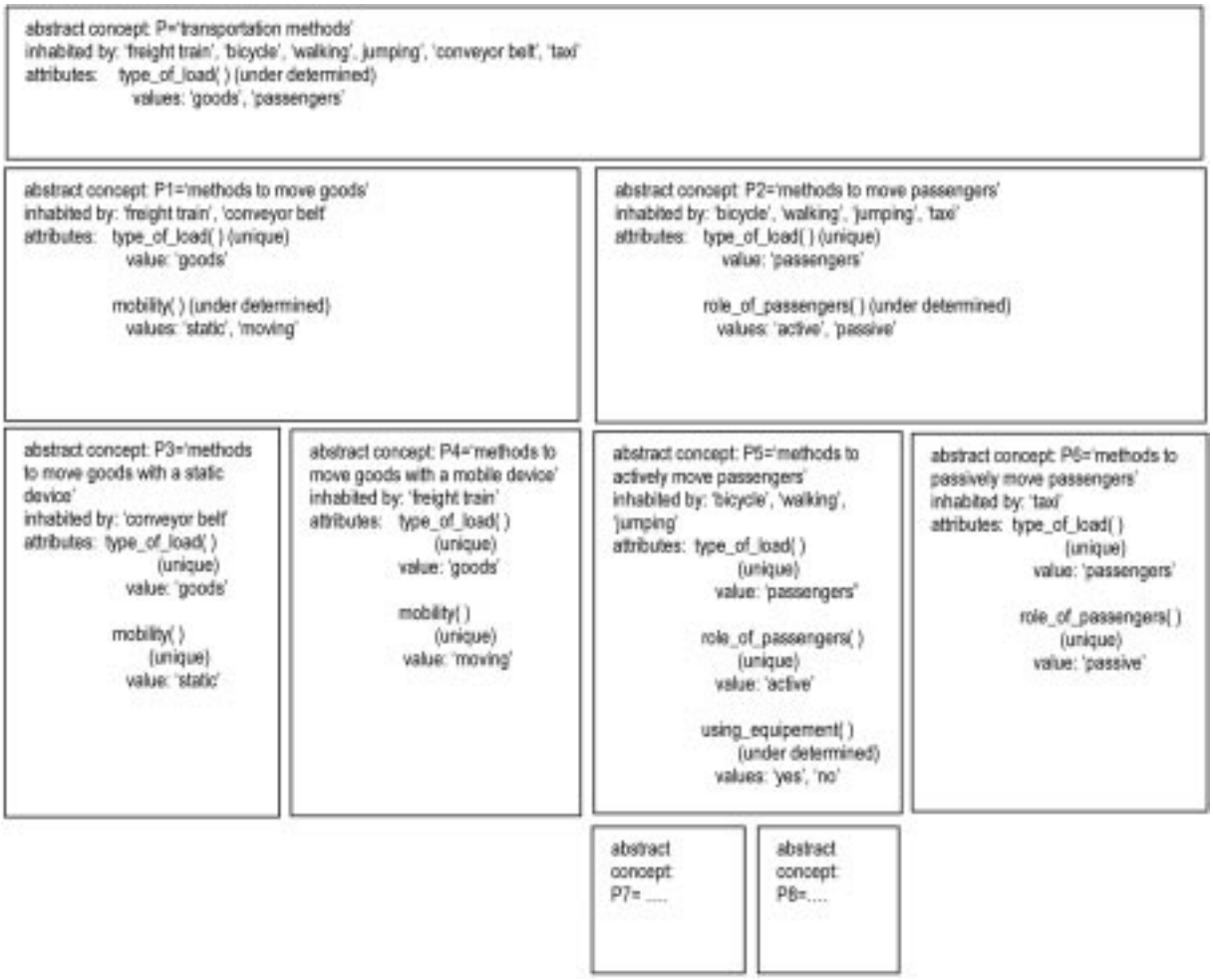


Fig. 2. An example of a hierarchical partition of an option space.

rejected as well (because A1 is defined operationally for all concepts in that partition, and it returns V1 on all these concepts);
- the hierarchic structure captures all nuances that were present in the original brainstorm output, as opposed to the often recommended clustering technique, that essentially *eliminates* these nuances.

It is, of course, possible to build several hierarchies on the same option domain: ultimately, the purpose of exploring the option domain will determine the adequateness of a particular hierarchy.

*Orthogonal structures*

For a hierarchic partition, it is sufficient to have one attribute that is operational (and preferably productive) for the entire option domain. All subsequent attributes only have to work for smaller and smaller sub partitions. Suppose, however, that we can find several operational (and productive) attributes that work for the entire option domain. Then we can build an *orthogonal structure*. Finding productive attributes that are all mutually orthogonal and that are operational on the *entire* option domain is obviously a more demanding task than just finding operational attributes for the subsequently decreasing partitions in the hierarchy. On the other hand, it has serious benefits (for simplicity, we only consider binary attributes in the sequel):

- With N binary attributes, we can distinguish $2^N$ partitions. Conversely, if we fully want to distinguish M concepts (the results from our brainstorm) with an orthogonal structure, we ideally need only round(log(M) + 1) orthogonal attributes.
- In most cases, the number of concepts that is found in an initiating brainstorm is not an integer power of 2. For instance, for M = 17, we need at least 5 orthogonal, binary attributes. But then we can distinguish 32 partitions. So at least 32 − 17 = 15 partitions are empty: these partitions refer to segments of the option space we didn't think of in the brainstorm. Often it pays to investigate these cells: they have been overlooked, but they could contain promising, new ideas.

We typically want to achieve the advantages of fully orthogonal structures, but it may be too difficult to actually find sufficiently orthogonal attributes. (Figure 3 refers.) Two hybrid compromises between hierarchical structures and orthogonal structures exist:

- if only one productive attribute can be found that works for the entire set of initial concepts, we start with building a hierarchy. It may be possible, however, that one or more of the partitions of this hierarchy admits an orthogonal structure.
- if more than one productive attributes can be found, but not enough to assure that all partitions are occupied with only one concept, the crowded partitions (= partitions inhabited by more than one concept) can be endowed with hierarchical structures.

*Dialectics*

The structuring devices that we saw thus far (hierarchies, with orthogonal structures as a special case) require an initial collection of concepts. This is not always convenient. Many innovation projects start with the question: 'What variations could be conceived of C?' where C is an existing concept. We already saw the use of productive attributes, and we recommended searching for productive attributes; here we will use productive attributes again to create a *dialectic structure*. The term dialectics goes back to Hegel and Fichte (see [23]). It refers to the three-fold relation between notions: thesis, anti-thesis and synthesis. We use it in the following context:

> Let C1 be an initial concept, which plays the role of the Hegelian thesis. Let A be a productive attribute (say, a binary attribute for simplicity). Let A(C1) = V1. The concept C2, with A(C2) = V2, obviously plays the role of the anti-thesis with respect

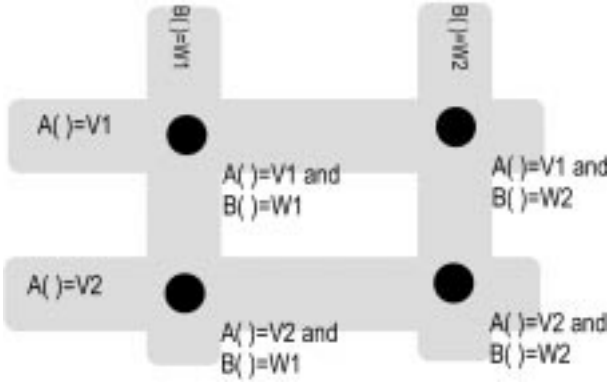


Fig. 3. Schematic depiction of simple orthogonal structure. This is equivalent to two hierarchical structures, one with the attribute A( ) as the distinguishing attribute for the root (= top level) node and B( ) in the child notes, and one with B( ) in the root node and A ( ) in the child nodes. In general, every orthogonal structure with n attributes can be seen as hierarchies in n! ways: one hierarchy corresponds to every permutation of the attributes.

to attribute A. Further, a second attribute A′ with A′(C1) = V′1 can be used to introduce a second anti-thesis, say C′2 with A′(C′2) = V′2. So for a given thesis, we allow multiple anti-theses. Also, multiple anti-theses arise if an attribute can yield more than 2 different values.

A more abstract concept C3 for which A(C3) is defined, but has an underdetermined value, is the synthesis. C3 is one level more abstract than C1 and C2. We recommend putting effort in explicitly formulating this more abstract concept, because this will probably reveal some new attributes of C1 (and C2)—which may be productive attributes. We give an example:

Let C1 be a pencil. One attribute is 'deposited material( )', with value 'deposited material(pencil) = carbon'. An alternative value is 'ink', and it takes little effort to find that C2 might be a pen. Now the synthesis should encompass both pen and pencil. An attempt to explicitly define this abstract synthesis could read: 'a hand-held device to put visible material onto a planar surface with the purpose of creating linear shapes (letters or drawings)'. Indeed, we suddenly have a host of values that immediately relate to new (thus far 'hidden' attributes, that already existed in the starting concept 'pencil'):

- 'hand held' as value of the attribute 'connection to a user( )' (as opposed to e.g. 'foot-held', for writing devices for usage by people who miss the usage of their hands)
- 'put' as value of the attribute 'direction of material flow( )' (as opposed to e.g. 'take-from', for writing devices that operate by cutting out letter silhouettes from paper, as in early day stencils)
- 'visible' as value of the attribute 'visual quality of produced traces( )' (as opposed to e.g. 'invisible' for writing devices with invisible ink as used in espionage)
- 'material' as value of the attribute 'nature of produced traces( )' (as opposed to e.g. 'electromagnetic energy' for laser pens that scorch traces into heat-sensitive material)
- 'planar' as value of the attribute 'surface quality of the carrier for the traced lines( )' (as opposed to 'rough' for drawing devices that are designed to draw onto non-smooth surfaces)
- 'surface' as value of the attribute 'topologic genus of the carrier for the traced lines( )' (as opposed to 'volume' or '1-dimensional variety' for writing devices that, respectively, work by squirting free-standing, plastic serpentines into free air to create 3-D constructs, or writing devices designed for labelling electric wires as e.g. in telephone cables)
- 'purpose' as value for the attribute 'reason for being( )' (as opposed to 'useless' for writing devices that serve as artistic objects, decoration material in shop windows, or props in theatre)
- 'creating' as value for the attribute 'direction of communication( )' (as opposed to 'sensing' for a hand-held scanner or an electronic widget that is able to sense signatures etc., or 'erasing' for a pencil-shaped eraser)
- 'linear' as value for the attribute 'quality of the created traces( )' (as opposed to 'fractal' for devices with build-in modulators to create artistic effects, or 'broad' for paint brushes).

Of course, this is only a selection of the millions of a priori hidden productive attributes of a pencil. In order to see dialectics at work, we select the first attribute ('connection to a user') and follow the

Table 1. Table of theses and anti-theses of the pencil-example

| Thesis | Anti-thesis | Distinguishing attribute | value 1 | value 2 | Synthesis |
|---|---|---|---|---|---|
| pencil | pen | deposited material | carbon | ink | hand-held device for [. . .] |
| hand-held device for [. . .] | foot-held device for [. . .] | body extremity that holds the pen | hand | foot | writing device that is directly controlled by body movements for [. . .] |
| device that is directly controlled [. . .] | device that is indirectly controlled [. . .] (e.g., mechanical typewriter) | way of control | direct | indirect (i.e., via mechanical means) | writing device that is controlled by body movements |
| writing device that is controlled by body movements | writing device that is controlled by other autonomous agent | the controlling agent | human | other autonomous agent (e.g., a computer printer) | writing device = device that produces written text |
| device that produces written text | device that reads written text | direction of communication | output | input | device that conveys written text |
| device that conveys written text | device that conveys visual info, other than written text | type of visual information that is conveyed | text | non-text | device that conveys information |
| device that conveys information | device that stores information | type of manipulation with the associated information | conveying | storing | device that manipulates information |
| device that manipulates information | device that manipulates matter | what is manipulated | information | matter | device |
| device | non-device | ? | ? | ? | everything |

abstraction in upward direction in the subsequent rows of Table 1.

This table shows some aspects of using dialectics in design practice:

- Notice that in this table, at every subsequent row the new thesis equals the previous synthesis. This is conforming Hegel's formulation of dialectics. In every new row, one new (thus far 'hidden') attribute is introduced, and one previous attribute looses its distinguishing value.
- It is very tempting to wander off into abstraction. Soon, however, one arrives at a concept labelled 'everything', i.e., no attribute has a distinct value anymore. The lower most few rows of the table, however, won't be relevant in any practical context. The art (and the challenge!) of using dialectics is to use small steps of abstraction, and to keep a constant eye on the *purpose*. In the current case we didn't mention this purpose before hand; a possible purpose could be the extension of a product portfolio of a pencil manufacturer. Obviously, it then depends on the existing expertise, the production facilities, the market situation, etc., which abstractions are meaningful and which are not.

At any time, we have the choice to proceed in three directions:

- into increasing abstraction (as with the subsequent rows in the above table; in the diagram below, these are the upwards arrows in Fig. 4);
- explore at the same level of abstraction (i.e., consider as much as possible hidden attributes of the current formulation of the concept, as we did in the bullet list above the table. In Fig. 4, these are the sideways arrows); or
- to decrease the abstraction by considering the current concept as the root node of a hierarchy (in Fig. 4, they would correspond to recursive hierarchical partitioning according to the method under Hierarchy).
- We observe that a hierarchic structure is a special case of a dialectic structure. (Indeed, a hierarchical partitioning is strictly top-down, whereas a dialectic structure allows expansion in all directions. Further, notice that a hierarchical structure doesn't necessarily requires a brainstorm as bootstrapping device. After some experience has been gained by hierarchically ordering the results from a brainstorm, one can start to suggest hierarchically ordered attributes, *instead of* proposing concepts in a brainstorm session, right away).

We schematically depict part of the result of a dialectic session in Fig. 4. Compare this with the hierarchical partitions as discussed above. Notice that here nodes in the network refer to individual concepts, similar to the abstract concepts that were associated to the partitions under Hierarchy. Again we have a hierarchical (abstraction) relation between 'synthesis'-nodes and the underlying 'thesis'- and 'anti-thesis nodes'.

*Summary of the structured creativity methods*

The method of dimension lifting is mainly a didactic introduction to the usage of productive

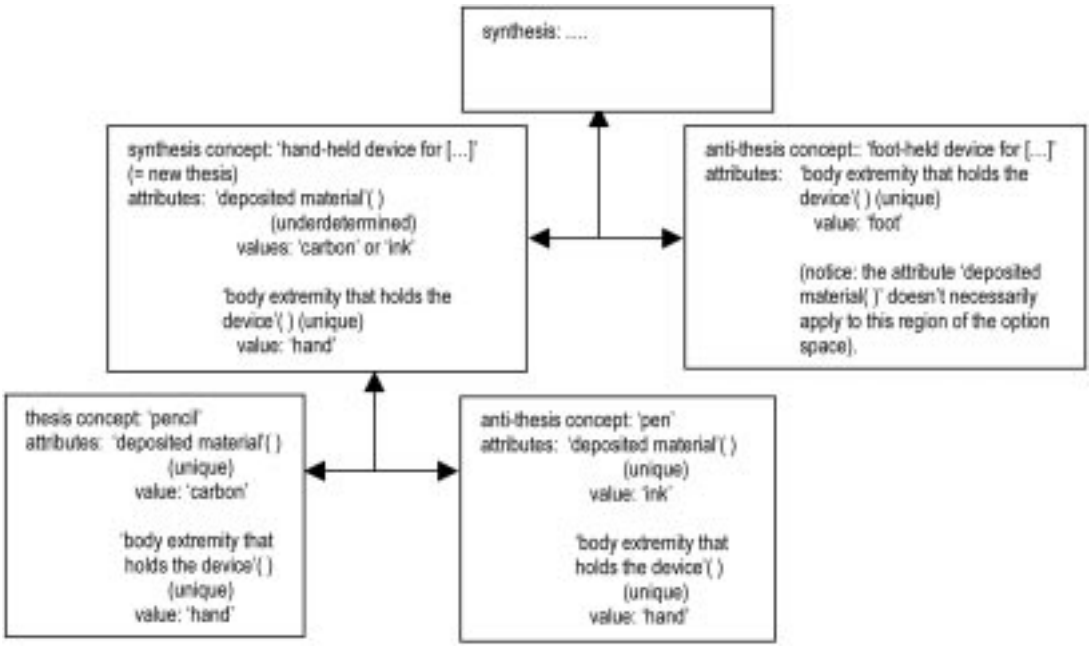


Fig. 4. schematic view of dialectics as a means to (partially) cover the option domain with a network of attributes and concepts. Concepts on one layer (say, level = n) are at the same abstraction level. One layer higher, say n + 1, means one abstraction level higher (= one more degree of freedom). Attributes that are introduced in layer n have an underdetermined value in that layer; they have unique values in all layers ⟨ n, and they do not apply in layers ⟩n. For simplicity, we only give one anti-thesis per thesis; in general, however, there will be multiple attributes, and for any attribute, multiple values that all generate anti-theses. Also, expansion can occur in top-down direction.

Table 2. summary of the structured creativity methods

| | Orthogonal structures | Hierarchical structures | Dialectic structures |
|---|---|---|---|
| is a special kind of . . . | hierarchical structures | dialectic structures | — |
| partitioning? | yes | yes | no |
| difficulty . . . | difficult to find sufficient operational attributes on the entire option domain | not so difficult as orthogonal, because subsequent attributes have to be operational on increasingly smaller sets of brainstorm items | easy: at any node, there are three directions to proceed (more abstract, less abstract, or at same abstraction level with other attributes) |
| needs for a start . . . | Brainstorm | brainstorm | can start from single concept |
| number of attributes needed | O(log N) for N concepts | O(N) for N concepts | O(N) for N concepts |
| generates empty cells (with possible innovative ideas thus far overlooked) | yes | not necessarily | in general not |
| way to proceed | for every new proposed orthogonal attribute, top down | strictly top down | in arbitrary directions. One typically starts with bottom up. Pitfall: one may wander off into impractical abstraction |

attributes. Productive attributes form the kernel of our approach. They are exploited in hierarchical, orthogonal and dialectic exploration. In order to facilitate the choice between the various techniques in a given circumstance, we summarise the merits and disadvantages of hierarchical methods, orthogonal methods and dialectics in Table 2.

## SOME PRACTICAL EXPERIENCES

The authors have been involved in setting up the format for teaching creativity in technological design contexts and in applying it in courses about interdisciplinary design methodology for SAI. Although the courses are still in their infancy, and statistically reliable results are still lacking, their may be some merit in early dissemination of the underlying ideas: first, to solicit comments, criticism and additional ideas from the design education community; second, other teachers in the field may want to experiment with similar ideas so that a larger number of prospective designers is exposed to these ideas, which could accelerate the process of getting insight in their usefulness in the practice of technological design.

The populations in Table 3 have been exposed to the above techniques:

Since the courses are still under development, we have not yet performed in-depth quantitative investigations including, among other things, control-group setups. A more stringent test of our approach would be measuring the use of other than only brainstorming techniques in design tasks performed by students that learned our methods. A test could also measure the creativity of designers applying our techniques, using, for example, the model suggested in [24]. The estimated benefits are only based on individual comments. 'Very high' means that more than an estimated 90% of the students in a discipline claim to believe that structured creativity techniques helps them to consistently gain better understanding of the problem area and generate ideas. 'High' means that is the case for more than 80%; 'Average' means that about 2/3 of the students

Table 3. Experiences with creativity teaching

| Discipline | Nr subjects | Prevailing backgrounds | Setting (C: classroom exercises; D: design assignment; P: professional design project) | Estimated benefit, as based on informal interviews |
|---|---|---|---|---|
| Software engineering | O(100) subjects, 5 courses | Computer science | C,D | Very high |
| Chemical process design | O(25) subjects, 2 course | Chemistry, physics, mech. Engineering, earth science, | D | Average |
| Mechatronic design | O(4) subjects, 1 course | Computer science, mech. Engineering | C, D | (High, but very small group) |
| Interaction design | O(40) subjects, 3 courses | Computer science, cognitive psychology | C,D | High |
| Others (in-house trainings in industrial settings, mixed populations) | O(130) subjects, 7 courses | Various technical disciplines | C,P | Very high |

have this opinion. Notice also that software engineering students have the best score. An explanation could be that the idea of productive attributes is most familiar to software engineers.

Further, our evaluation seems to indicate that, whereas for initial classroom exercises dialectics is often considered to be abstract and difficult, it is the most favoured one in design assignments. Apparently, once some experience has been gained, it becomes useful—in particular because it can start with only one initial concept. This was found consistently for software engineers and interaction designers. The popularity of the 'Hegelian' approach may be related to the fact that it is very natural: it works by introducing modifications to existing systems. The same mechanism can be recognised in evolution, software lifecycles etc.

## DISCUSSION AND FUTURE WORK

Using concepts, attributes, and values, together with some structuring devices such as hierarchies, orthogonality and dialectics, in teaching various design-related creativity techniques to engineers seems to be able to bridge the gap between technological and more artistic forms of creative thinking. This may be due to the fact that the use of concepts, attributes and values is actually common practice in much of technological thinking; it is only the strong emphasis on explication and manipulation of the attributes and concepts that may be new. The principles that inspired to our notions (most noteworthy, object orientation and data modelling) have proven to be extremely versatile in a vast variety of problem areas. There are, however, some inherent risks in over-enthusiastic adoption of these concepts:

- It is an illusion that panaceas exist: there are undoubtedly design situations for which our approach is not at all a natural one, and a designer will, in these cases, be more flexible and versatile if he is familiar with a variety of techniques. It even has some didactic benefits to teach some techniques that are not immediately compatible with each other in order to stimulate the designer's own critical and common sense-style of thinking.
- Our methodology *looks* mathematical, but in the form presented here, it lacks a rigorous mathematical foundation. This could lead to overly optimistic expectations or even to false claims regarding the alleged quality of a design process that is based on it.

Observing the above criticisms, we see that a lot of work still should be done. The most important tasks are:

- Tool support: since we stress explication, the amount of information that will be administrated is quite large. The designer should not be bothered by doing this by hand.
- Probably more important: we should collect a number of more or less realistic design assignments that show our ideas at work. Prospective designers are helped much more by one convincing and inspiring example than with twenty sophisticated, but incomprehensible guidelines.

## REFERENCES


1. Information about the SAI: http://www.sai.tue.nl
2. J. Teminko, A. Zusman and B. Zlotin, *Systematic Innovation*, Ideation International Inc. Southfield, Mich. (1998).
3. M. Raina (ed.), *Creativity Research: International Perspective*, National Council of Educational Research and Training (1980).
4. J. Glover, R. Ronning and C. Reynolds (eds.), *Handbook of Creativity*, Plenum, London (1989).
5. J. L. Adams, *Conceptual Blockbusting*, Norton, New York (1980).
6. E. de Bono, *Lateral Thinking*, Penguin Books, London (1970).
7. E. de Bono, *Serious Creativity: Using the Power of Lateral Thinking to Create new Ideas*, HarperCollins, London (1993).
8. H. Lytton, *Creativity and Education*, Routledge and Kegan Paul, London, UK (1971).
9. R. Baily, Disciplined Creativity for Engineers, Ann Arbor Science, Ann Arbor, USA (1978).
10. H. Christiaans, *Creativity in Design: The Role of Domain Knowledge in Designing*, Ph.D thesis, Lemma, Delft, The Netherlands (1992).
11. S. Dasgupta, *Creativity in Invention and Design: Computational and Cognitive explorations of technological originality*, Cambridge University Press, Cambridge (1994).
12. J. Gero, and M. Maher (eds.), *Modeling Creativity and Knowledge-based Creative Design*, Lawrence Erlbaum, Hillsdale (1993).
13. S. Isaksen, K. Dorval and D. Treffinger, *Creative Approaches to Problem Solving*, Kendal & Hunt, Dubuque (1994).
14. T. Rickards, *Creativity and Problem Solving at Work*, Gower, Aldershot (1990).
15. T. M. Amabile, *The Social Psychology of Creativity*, Springer-Verlag, New York (1983).
16. L. G. Richards, *Stimulating Creativity: Teaching Engineers to be Innovators*, 28th Frontiers in Education Conference, 3: 1034–1039 (1998).
17. S. Ghosh, *Architecting a Course for Engineering Design, Critical Inquiry, and Creativity*, 30th Frontiers in Education Conference, 1: F1C/24 (2000).
18. K. L. Kitto, Using TRIZ, *Parametric Modeling, FEA Simulation, and Rapid Prototyping to Foster Creative Design*, 30th Frontiers in Education Conference, 2: S2E/14–S2E/18 (2000).

19. B. Mayer, *Object-oriented Software Construction*, ISE Inc./Prentice-Hall, Santa Barbara, Cal. (1997).
20. I. M. M. J. Reymen, Improving Design Processes through Structured Reflection: A Domain-independent Approach, Ph.D. thesis, Eindhoven University of Technology, Eindhoven, The Netherlands (2001).
21. M. Ivashkov and K. van Overveld, An Operational Model for Design Processes, *Proc. Int. Conf. Engineering Design*, Glasgow, UK, 2: 139–146 (2001).
22. Z. Kogan, *Essentials in Problem Solving*, Arco, New York (1956).
23. W. van Dooren, Dialectiek (in: Grote Winkler Prins Encyclopedie. Elsevier, Amsterdam, NL (1973)).
24. C. Redelinghuys, A model for the measurement of creativity: relating expertise, quality and creative effort, *Int. J. Eng. Educ.*, **13**(1), 1997.

**Kees van Overveld** (1957) obtained a M.Sc. (1981) and Ph.D. (1985) in physics at the Eindhoven University of Technology. Also in 1985, he joined the computing science department of the faculty of Mathematics and Computer Science of EUT as a university lecturer; since 1990 as associate professor. From 1989 to 1998 he was head of the Computer Graphics group. From November 1996 to June 1998 he was also employed as a Senior Researcher at Philips Research; he continues working for Philips Research as a consultant. In May 2000 he joined the Stan Ackermans Institute (SAI) to teach and research design methodology.

**Rene Ahn** (1956) graduated at the Eindhoven University of Technology, in 1982. From 1982 until 1988 he worked as a researcher at Philips Research Laboratories, mainly in the area of computer aided systems design. In 1988 he joined the language and computer science group at Tilburg University. Here he was involved in research on semantics and knowledge representation, as well as the design and development of intelligent user interfaces. In 1998, he joined the Centre for User-System interaction, at Eindhoven University of Technology. At present, he is employed as an assistant professor within the User-Centred Engineering group of the faculty of Technology Management.

**Isabelle Reymen** (1973) graduated in June 1996 as Civil Engineer Architect at the Faculty of Applied Sciences of the Katholieke Universiteit Leuven, Belgium. In September 1996, she started a Ph.D. at the Stan Ackermans Institute (SAI), Centre for Technological Design at the Technische Universiteit Eindhoven, The Netherlands. She received her Ph.D. degree, in April 2001, with a thesis entitled 'Improving Design Processes through Structured Reflection, A Domain-independent Approach'. Since February 2001, she is working at the SAI as a post-doc. She performs design research related to reflection and teams and is involved in design education for the SAI programs.

**Maxim Ivashkov** (1975) graduated from the physics department of the Belo Russian State University in Minsk in 1997. He spent 1.5 years working for the branch of Invention Machine Corporation in Minsk. There he became acquainted with TRIZ and other strategies of solving technical problems. In 1999, he joined the Technical University of Eindhoven in the Netherlands to begin the program of Mathematics for the Industry. Since May 2000 as a Ph.D. student, Maxim is doing research focused on understanding of existing design processes and improving them with respect to the structure, operationality and traceability.