

Developing Exercises for Teaching Computing and Information Technology: A Theoretical Framework

Paula Kotzé
University of South Africa,
Pretoria, South Africa

kotzep@unisa.ac.za

Peter Purgathofer
University of Technology
Vienna, Vienna, Austria

purg@igw.tuwien.ac.at

Abstract

Computing and information technology are very much design disciplines, although not always recognized as such. To succeed as a designer in this context requires experience in design skills, which implies appropriate practice. In the context of teaching computing and information technology this implies providing students with appropriate design exercises. This paper discusses a theoretical framework for the development of such exercises, based on the argument that students must experience design to enable them to learn effectively. The framework scopes exercises on a continuum ranging from creativity to real-world use based on two philosophies: elaboration and reduction.

Keywords: Design exercises, design process, elaboration, reduction, creativity, real-world use.

Introduction

Computing and information technology are very much design disciplines, although not always recognized as such. Design disciplines require the mastery of skill. Skills are acquired or mastered by trying things out, i.e. students need to practice design in order to become competent (Baumann, Kotzé, Oestreicher, Bannon, Varey, et al., 2007; Sheppard, 1999).

If one studies current curricula focussing on computing and information technology it is clear that most focus on specific technologies with less emphasis on the crafts of design (methods and practices). Computing and information technology therefore lean towards teaching from the point of view of exploring and implementing technologies rather than innovation, i.e. discovering new avenues of doing something.

One of the toughest skills to acquire is the combination of creative insight and analytical thinking and it usually comes with experience (Sheppard, 1999). But this is not to say that we should not

Material published as part of this publication, either on-line or in print, is copyrighted by the Informing Science Institute. Permission to make digital or paper copy of part or all of these works for personal or classroom use is granted without fee provided that the copies are not made or distributed for profit or commercial advantage AND that copies 1) bear this notice in full and 2) give the full citation on the first page. It is permissible to abstract these works so long as credit is given. To copy in all other cases or to republish or to post on a server or to redistribute to lists requires specific permission and payment of a fee. Contact Publisher@InformingScience.org to request redistribution permission.

equip our students with the necessary skills to be innovative, over and above being able to implement the things they were taught in the classroom. After all, when they go out into the real world they will be expected, at least at some stage, to be innovative and to implement new ways of doing something. Lecturers have come to understand that creative or innovative design cannot be taught in a single course. Design must be experi-

enced and it is an experience that must grow with the student's development. One way of making sure this happen is to use appropriate exercises in 'traditional' classes throughout the curriculum.

The question that arises is how do we include these ideas in an already overfull curriculum. As lecturer, where do you start, how do you define your syllabus and the exercises to equip your students with the practical skills to be successful designers and innovators? In this paper we are not going to explicitly focus on the syllabus. We rather argue that we should rethink the way in which we design the exercises we ask our students to do: we should test their basic skills up to system implementation level, but we should also challenge them to be innovative, intuitive, creative and even artistic. It is these skills that would give them the competitive edge amongst a group with the same technical background.

This paper proposes a framework for design exercises for use in the teaching of computing and information technology aiming at positioning the exercises with regards to both creativity and working implementations (i.e. real-world use). The framework can guide educators to design exercises that will enable students to experience design and innovation. Since the framework is generic, it can be adapted to fit design exercises for any syllabus. The idea for the framework was conceived during a workshop on teaching interaction design, and the first version of the framework, focusing specifically on interaction design, was published in Kotzé and Purgathofer (2007).

The paper starts off by discussing the theoretical foundation for the proposed framework. This is followed by a discussion of the components of the proposed framework and possible examples of its use. The final section concludes.

Theoretical Foundation

Moggridge (2007) argues that to educate intuitive designers we must equip our students with the skills to trust that they will be able to do design (teaching them basics methods and skills), but we also need to teach them how to analyse alternatives and existing designs. In order to be a good designer he defines five design skills that a designer must have:

- To frame and reframe the objectives.
- To create and envision alternatives.
- To select from those alternatives, knowing intuitively how to choose the best approach.
- To visualize and prototype the intended solution.
- To synthesize a solution from the relevant constraints, understanding everything that will make a difference to a result.

The question this paper addresses is how we can design the exercises we give our students in order to master the methods and skills they have been taught to meet these required design skills.

The CONVIVIO Faculty Forum (Baumann et al., 2007) lists a number of properties good design exercises should hold in order to stimulate creativity and support learning outcomes, including:

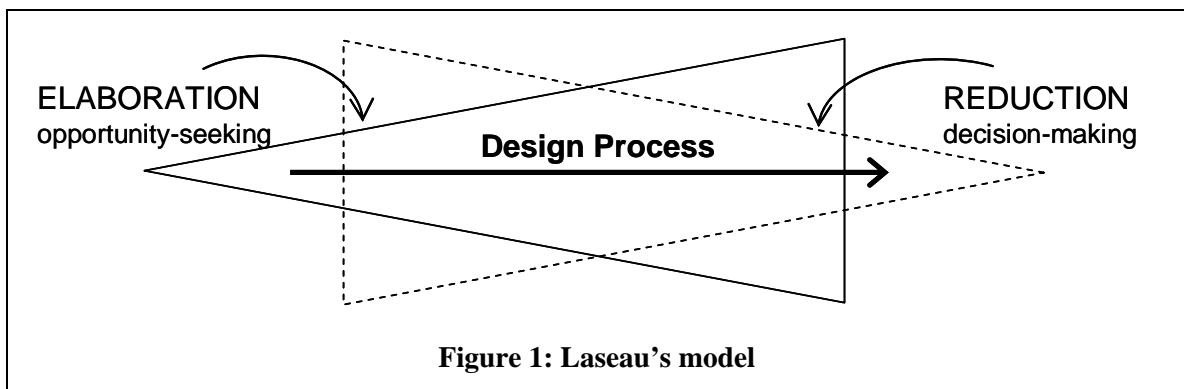
- Be relevant, preferably real, and tangible.
- Keep students motivated and engaged throughout.
- Be of appropriate span to the learning outcome envisaged: Short and sharp for applying skills, methods and techniques or longer for reflection and understanding.
- Aim for continuity if at all possible, i.e. it is preferable to have three days intensive work rather than several short exercises.
- Provide well defined constraints and limitations (it helps creativity). There should be a clearly defined design space, with clear rules of the game.
- Allow for peer-critiquing (which should preferably not be grade relevant) without instructor's intervention (allegedly).

- Allow time for reflection by students.
- Focus on knowledge skills within a specific context.
- Separate designs results and implementation.
- Include project and time management as part of the plan.

We propose a framework for exercises to assist the lecturer in designing such exercises. The proposed framework is based on combining the ideas of Moggridge (2007) with three theoretical frameworks: the graphic thinking for architects and designers framework proposed by Laseau (2000), Buxton's (2006) Separator for Sketches and Prototypes and the framework for teaching and learning design proposed by Kotzé, Renaud and Van Biljon (2006). We briefly introduce these three theoretical frameworks in the next two sections.

Laseau's Graphic Thinking for Architects and Designers Framework and Buxton's Separator for Sketches and Prototypes

Laseau (2000) captures a number of core aspects of the design process in a simple model, illustrated in Figure 1. The model consists of two funnels: an expanding funnel (elaboration) and a contracting funnel (reduction). In this model he balances permanent creativity and idea generation, on the one side, with the reduction resulting from decision-making as main forces in design, on the other side. These two ingredients of design benefit from quite different approaches and methods that can be characterized as 'sketching' and 'prototyping', respectively.



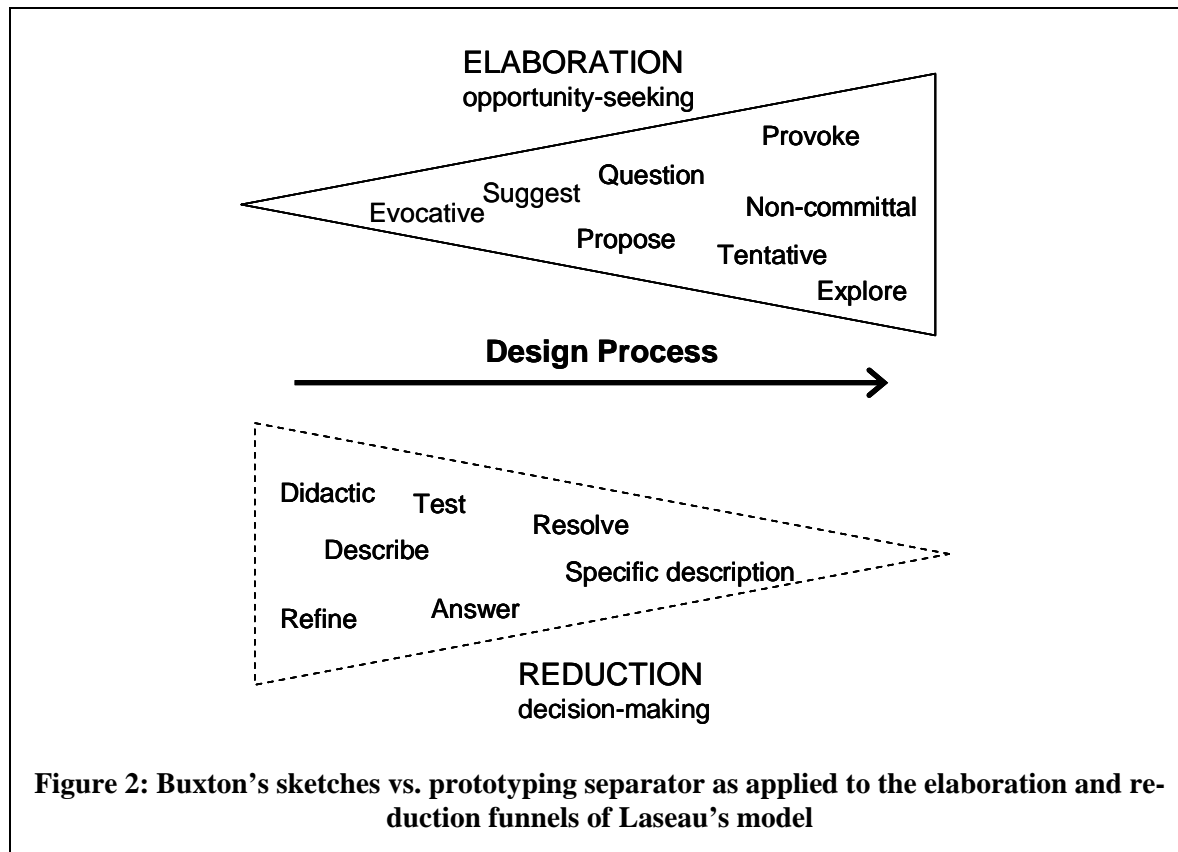
In this context, it is necessary to understand sketching as an activity that is not necessarily tied to pen and paper but encompasses any generative design work that share a number of characteristics with pen-and-paper sketching (for example making a free hand drawing of alternative ways to structure an algorithm). More specifically, sketches (in any medium) are (among others) quick, inexpensive, plentiful and disposable, and have distinct gesture, constrained resolution and ambiguity. It is obvious that these are properties that prototypes normally don't have. Buxton (2006) defines a separator between attributes of sketches versus prototypes using opposite concepts:

- Evocative vs. didactic.
- Suggest vs. describe.
- Explore vs. refine.
- Question vs. answer.
- Propose vs. test.
- Provoke vs. resolve.
- Tentative and non-committal vs. specific description.

Buxton stresses that in practice these concepts are not in opposition, but they open up a continuum between sketching and prototyping. For the sake of teaching, we might very well treat them

as if they were either/or propositions, in order to help students understand the differences and respective merits of the two activities.

If we link this separator to Laseau's model, as illustrated in Figure 2, sketching would be the best approach for the expanding funnel of elaboration and opportunity seeking, while prototyping is the ideal method for the contracting funnel of reduction by decision-making. Put differently, it shows that we will have to teach two opposite sets of skills to computing and information technology students if we wish to fully equip them for the challenges they will face in the 'real-world'.



Becoming good in prototyping, on the one hand, is based on skills that are usually subsumed under the term 'problem solving'. Teaching of problem solving skills often makes up the major part of computing and information technology programs. Exercises that let students practice problem solving characteristically have clear and unambiguous objectives, coherent and defined settings and explicit, traceable evaluation criteria. It is understandable that such exercises are preferred by lecturers since they are much easier to define and assess and lead to less misapprehension and discussion among the students. Such exercises could be characterized as being relatively 'low-maintenance'. With the design problem 'given', students can go ahead and (taken Buxton's separators) answer and resolve these problems by testing and refining specific descriptions of a solution.

For sketching, on the other hand, problem solving is of relatively little interest and can even be seen as detrimental. Schön (1983) coined the term 'problem setting' as a contrast to problem solving. To train the skills needed for problem setting, coherent, unambiguous, defined and clear exercises won't work and would be the exact opposite of what is needed. Students need to work on open and even inconsistent assignments, in undefined and ambiguous contexts. Performance in

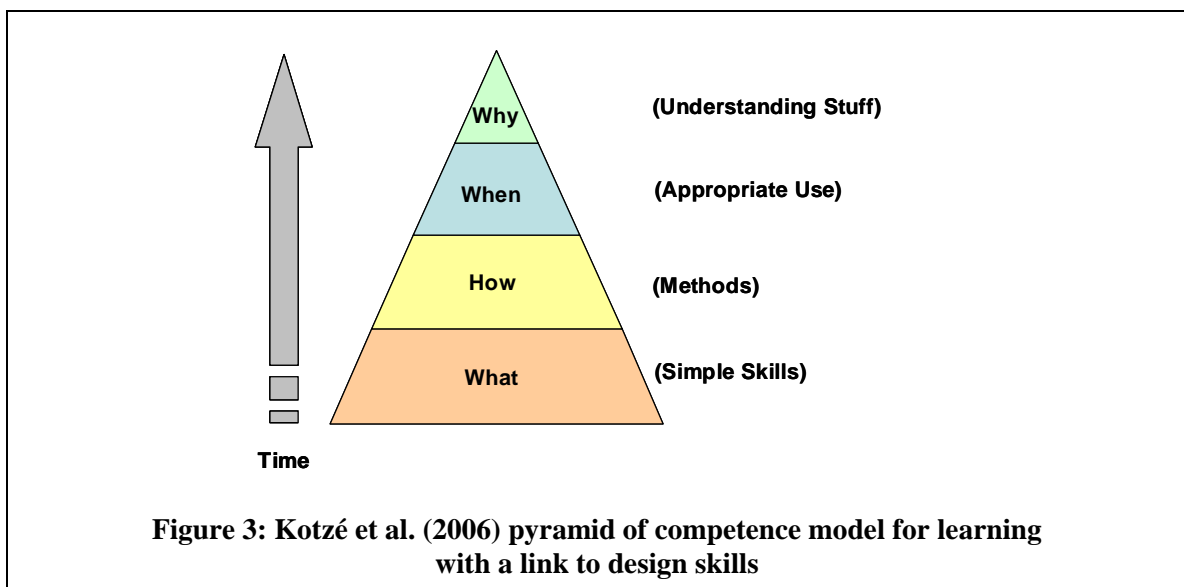
this type of exercise is much harder to evaluate by objective means, and it is often impossible to publish clear evaluation criteria up-front. Still, it is important to see that this is not impossible.

According to Laseau, exploration can be conducted systematically, and he proposes a number of sketching techniques that are all based on the idea that to change an image enables us to get a new look at it, thereby expanding our thinking. These techniques can be used as starting point for the design of exercises, e.g. transforming, structuring and ordering images. These techniques, however, only work if, as Laseau writes, we are ‘comfortable with exploration that is not tightly focused, let the mind wander, and be open to unexpected results’ (Laseau, 2000: p.115). This might be the real challenge in any kind of education, not only teaching computing and information technology, as McKim (1972: p.45) recognizes: ‘For most people, breaking lazy, category-hardened, fear-inducing habits of seeing is an educational task of considerable magnitude’.

Kotzé et al.’s Framework for Teaching and Learning Design

The second part of our theoretical foundation is based in learning theory. People ‘learn’ by repeated exposure to concepts using one of two major types of learning – implicit or explicit (Kotzé, Renaud, & Van Biljon, 2006):

- Implicit or unintended learning (Kirkhart, 2001; Polanyi, 1958; Reber, 1993; Taatgen, 1999) can be seen as a passive process where people simply acquire knowledge of the information by means of exposure to such information, i.e. it is unconscious and always active. Invoking implicit knowledge involves the indirect application of the knowledge without the requirement of knowledge declaration.
- Explicit learning, or intended learning (Kirkhart, 2001; Taatgen, 1999) is characterised by people actively seeking out the structure of any information presented to them, i.e. it is intentional and conscious. For example, explicit learning would be involved if a designer is instructed to acquire some target knowledge and then to explicitly apply and state the knowledge acquired in the design phase.



Kotzé et al. (2006) proposes a pyramid of competence model for learning (and consequently teaching) design derived from the models of learning proposed by Gorman (2002) and Miller (1990). Both models are based on the concept of different kinds of knowledge building onto each other. The pyramid of competence model, as illustrated in Figure 3, identifies four types of

knowledge in design or technology knowledge transfer (Anderson, 1993; Lebiere, Wallach, & Taatgen, 1998; Shedroff, 1999):

- What (declarative) knowledge refers to the recall of facts and events. Declarative knowledge is composed of chunks, consisting of a number of slots each of which can hold a value (which can also be another chunk). In the context of computing and information technology this is the process of learning factual knowledge (or simple skills) about design techniques and the basic elements and processes of computing and information technology. The ‘what’ knowledge is therefore just data: it is the raw material we find or create and then use to build the artefacts.
- How (procedural) knowledge refers to the skill of knowing how to do something, the methods (ability) enabling the application of the knowledge or skills. Procedural knowledge is usually encoded as declarative knowledge first and then translated into procedures (algorithms), but can also be learned by feel or intuition. Procedural knowledge therefore consists of productions, which are condition-action pairs specifying the action to be taken if a particular condition is satisfied. In the context of computing and information technology this is the process of learning how to use the design elements (e.g. constructs and algorithms), processes and design techniques. It is about transforming data into information, accomplished by organizing it into a meaningful form, presenting it in meaningful and appropriate ways, and communicating the context around it.
- What (judgement) knowledge involves the ability to recognise when knowledge is applicable to a particular instance, i.e. recognising that a problem is similar to one for which a solution is known and knowing when to apply a particular procedure or solution. Judgement knowledge is therefore structured in a way that facilitates problem solving and creativity, and is usually applied by experts in a particular context. Whereas novices would rely more on declarative and to a lesser extent on general or weak heuristics based on procedural knowledge, experts rely more on judgement knowledge. In the context of computing and information technology this is the process of learning to recognise situations where the previously learnt technique or design element should be applied. It is about appropriate use, i.e. the ability to identify situations where techniques, skills or knowledge, or particular design elements can be applied. It is about conversation, storytelling and integration, and building artefacts so that the patterns and meanings in the contained information can be learned by others.
- Why (wisdom) knowledge refers to meta-cognitive monitoring which may lead to a new cause of action. It is related to judgement knowledge referring to the ability to reflect, question, and come up with new causes of action, and involves an element of moral reasoning. Wisdom is the most vague and intimate level of understanding. It is much more abstract and philosophical than the other levels and less is known about how to create or affect it. The why knowledge cannot be created, as we can with what, how and when knowledge and we cannot share it with others as we can these other levels of knowledge. It is about contemplation, evaluation, interpretation and retrospection. In the context of computing and information technology this is the process of understanding the rationale of the technique or design element, and understanding why it comprises a good and effective design, why a specific skill or method or design element will be appropriate or not, why when applied creatively it would enhance the problem domain, etc.

Interestingly, although we have to keep these four levels of knowledge in mind when teaching computing and information technology, the same four levels should also be kept in mind when designing computing and information technology artefacts. In terms of design skills

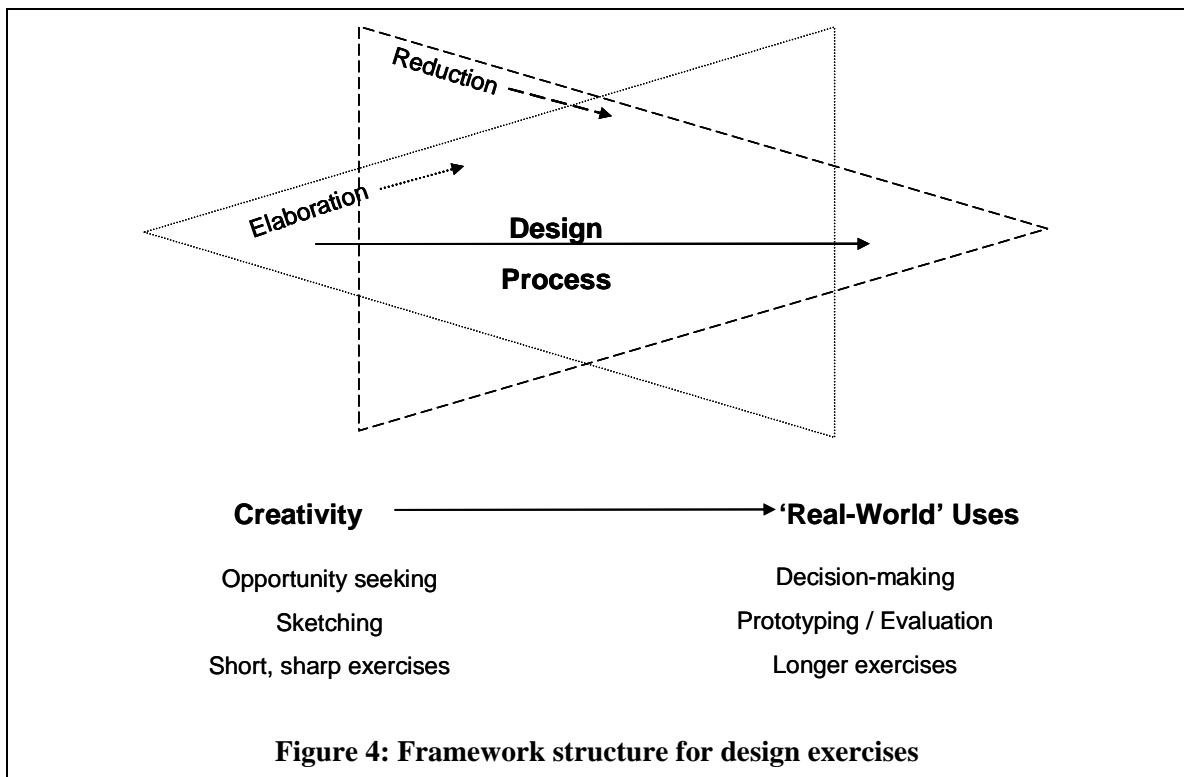
these four levels of learning can be defined as (Shedroff, 1999): knows about, knows how, shows how, and knows why.

A Framework for Design Exercises

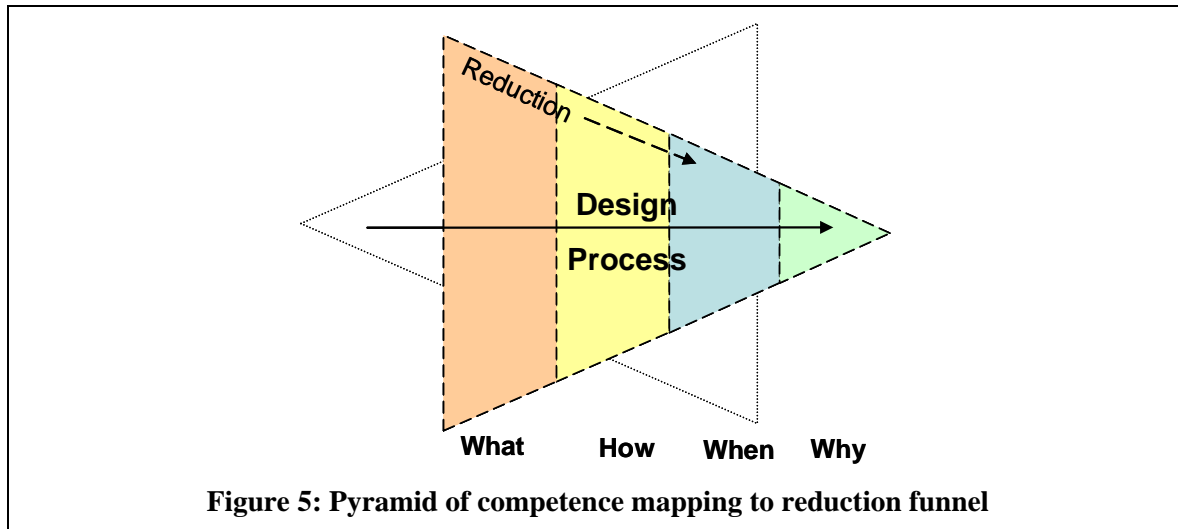
Philosophy of the Proposed Framework

The basic philosophy of our proposed framework for design exercises for computing and information technology is based on integrating and enhancing the ideas from the theoretical frameworks discussed above.

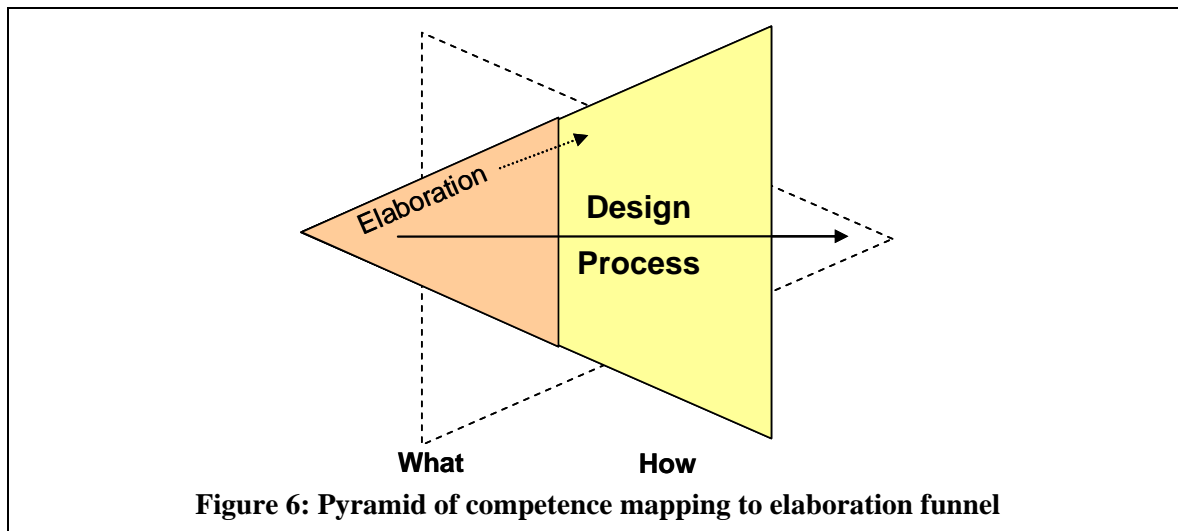
Using the ideas from Laseau (2000), Figure 4 illustrates the basic structure of our framework by suggesting that design is in some ways a continuum ranging from purely creative work to real-world implementations, i.e. sketches to prototype to use the Buxton (2006) terminology. Design exercises can be aimed at any point of this continuum, either starting at a specific point, or ending at a specific point. There is no wrong or right way of ‘designing’ design exercises entrenched in this model. When aiming at the creative side, exercises should be short, sharp and varied, requiring students to work with a variety of computing and information technology elements (methods, algorithms, data structures, application domains, etc.). Aiming at the real-world side, exercises will generally be longer, requiring judgment knowledge, and more exactly specified.



If we superimpose the Kotzé et al. (2006) pyramid of competence on the reduction funnel of the framework illustrated in Figure 4, we found that it has a natural fit, as illustrated in Figure 5. It follows the traditional way we teach in the fields of computing and engineering. It also follows the arguments of basic learning theory, on which the Kotzé et al. pyramid of competence is based.



Creativity is a judgement in a different way, involving a product, process, and a situation. The same aspects are often present in the reduction funnel, although frequently used in opposite ways of a range or being pre-specified. When we are creative we take particular thoughts, ideas and elements and connect them together in a fresh way so as to produce an experience which is novel, interesting and possibly of value. The elaboration funnel therefore differs from the reduction funnel in the way it links to the pyramid of competence, as illustrated in Figure 6. As it is focused on opportunity seeking, it does not necessarily reach the ‘why’ or even the ‘when’ stages, although they are not completely excluded (in fact the reduction funnel will take over to reach these). The primary focus of the elaboration funnel is using skills and methods to manipulate the skills (‘what’) in a new way (‘how’) to create something that is not strictly pre-specified or described (and not necessarily usable or useful). This does not match the way in which we traditionally teach computing and information technology, and therefore require from us to rethink the way we approach our teaching.



Linking the Framework to Knowledge Transformers

In order to be usable we need to link the basic philosophy of our proposed framework with more concrete guidelines to assist in adapting our teaching and the exercises we set for our students. We suggest using complementary knowledge transformers to accomplish this objective.

Sim and Duffy (2002) propose a set of knowledge transformers that could assist in making the link between learning and creativity on the design continuum. They distinguish between seven complementary knowledge transformers:

- Abstraction vs. detailing.
- Association vs. disassociation.
- Derivations (reformulation) vs. randomization.
- Explanation vs. discovery.
- Group rationalisation (clustering) vs. decomposition.
- Generalisation vs. specialisation.
- Similarity comparison vs. dissimilarity comparison.

In general the reduction funnel would aim at the knowledge transformer mentioned first, while the elaboration funnel would aim at the knowledge transformer mentioned second.

Laseau (2000) describes three approaches to creative exploration, which can also be seen as knowledge transformers. The approaches aim to re-centre visual thinking and ‘unlearn’ particular viewpoints with the aim of finding ‘unexpected viewpoints’. They are:

- Open-ended images that suggest a number of different perceptions or interpretations.
- Transformation of images.
- Structuring or ordering of images.

Table 1 integrates these two sets of knowledge transformers to propose ten knowledge transformers that can be used in the teaching of computing and information technology and which would support our proposed framework. The reduction funnel of our proposed framework would aim at the knowledge transformer mentioned first, while the elaboration funnel would aim at the knowledge transformer mentioned second.

| KNOWLEDGE TRANSFORMERS | HOW IS KNOWLEDGE TRANSFORMED |
|---------------------------------|--|
| Abstraction / Detailing | Abstraction generates a new, less detailed version through the use of abstract concepts and operators. Detailing generates new knowledge with more detail. |
| Association / Disassociation | Association determines a dependency between entities based on some logical, causal or statistical relationships. Disassociation asserts lack of dependency. |
| Derivations / Randomization | Derivations derive some knowledge from another piece of knowledge (based on some dependency between them). Randomizations transforms one knowledge segment into another by making random changes. |
| Explanation / Discovery | Explanation derives additional knowledge based on existing domain knowledge. Discovery derives new knowledge without existing underlying domain knowledge. |
| Clustering/ Decomposition | Clustering involves the grouping of past designs according to their similarities when considering certain perspective and criteria. Decomposition removes the groupings. |
| Generalization / Specialization | Generalization creates a description that characterises the entire concept based on a conjunction of all the specialisations of that concept. Specialisation increases the specificity of the description. |

| | |
|---|--|
| Similarity comparison / Dissimilarity comparison | Similarity comparison derives new knowledge about a design on the basis of similarity of the design and similar past designs. Dissimilarity comparison derives new knowledge on the basis of lack of similarity between two or more designs. |
| Constrained design spaces / Open-ended design spaces | Open-ended design spaces allow for ambiguity, collage and multi-valency of ‘sketches’ and invite new interpretations and ideas. Constrained design spaces remove this ambiguity by limiting interpretations. |
| Preservation of designs / Transformation of designs | Transformation is invoked by open-ended design, and changes perspective or perception, making the familiar seem strange. Preservation maintains the meaning and structure of designs. |
| Real-world context / Structuring or ordering of designs for non-specified context | A non-specific context creates an artificial context within which new responses can be made. Specifying a real-world context makes the environment exact. |

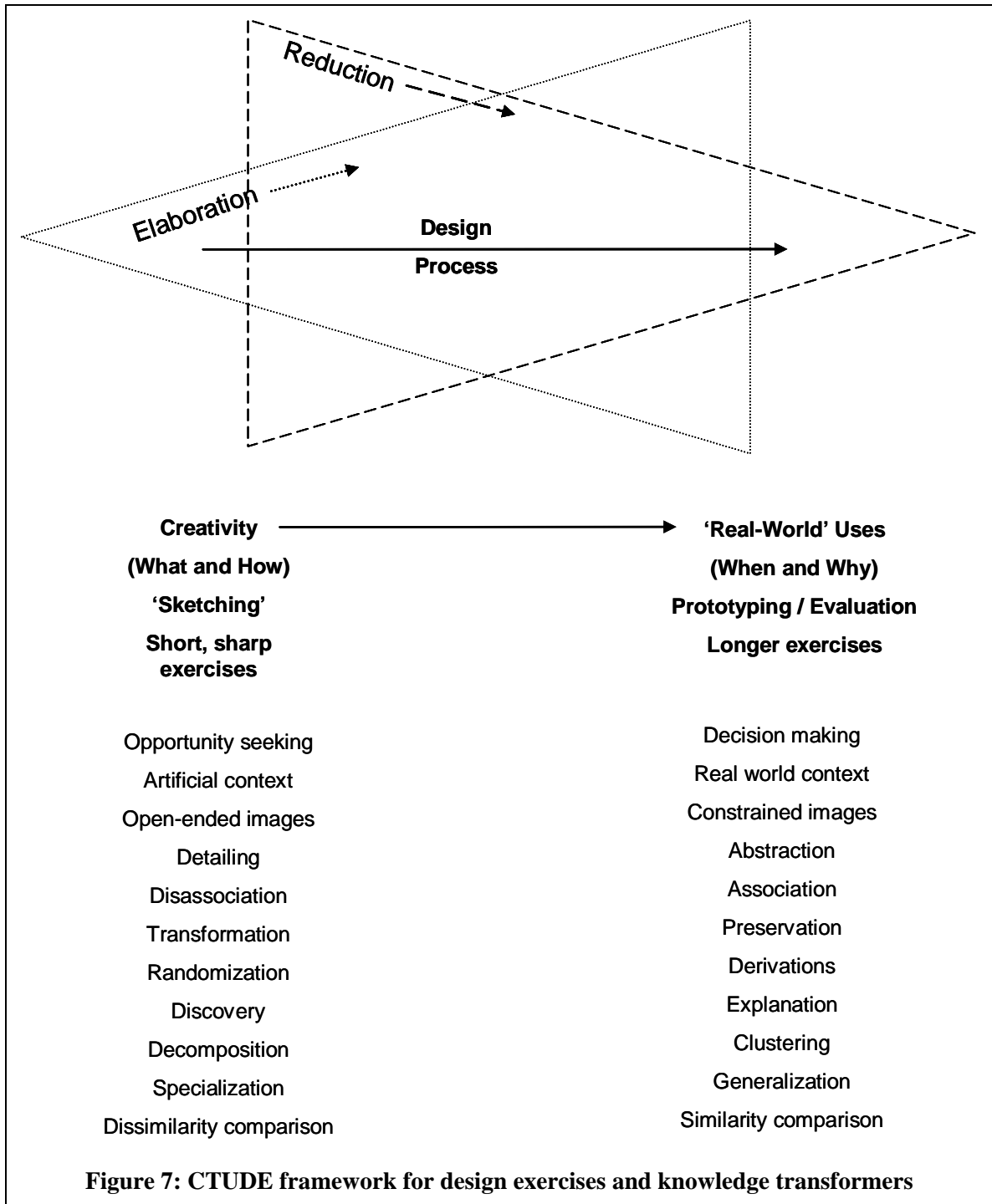
Proposed Framework

Figure 6 presents CTUDE (Creativity to Real-world Use Design Exercises), the proposed framework for design exercises aimed at the teaching of computing and information technology. CTUDE integrates Laseau’s model, Buxton’s Separator for Sketches and Prototypes, Kotzé et al. framework for teaching and learning, and both Sim and Duffy and Laseau’s sets of knowledge transformers.

To use CTUDE the lecturer has to decide where to position the exercises on the continuum and on which funnel, and then plan the experiences to fit this purpose using/suggesting appropriate knowledge transformers. If the purpose of the design exercise is experiencing creativity, one will aim for the left hand side of the continuum and equip the students with the tools and techniques matching the knowledge transformers suitable for this side of the continuum. If the aim is for students to design a small-scale real-world application, one would design the experience around tools, techniques and knowledge transformers at the opposite side of the continuum.

To illustrate the use of the framework, consider the following three examples:

- Students are required to design and implement a completely ‘wacky’ and ‘useless’ design (*opportunity seeking, artificial context*) from scratch and a prototype implementation for it, for example a timepiece that does not look like a watch and work on the principle of using the decimal system to indicate time.
- Students are given an example of a project solution of a previous group of students – say for example the project required the students to design a prototype system to design a web application that would provide ambient information to a user, with the implementation allowing for scalability over a variety of devices (PC, mobile device, etc.). Instead of starting off with a completely new solution from the beginning, the students are required to analyse the existing prototype by *decomposing* it, trying to *discover* the reasoning behind the design, and then to come up with at least three *dissimilar alternative* solutions to the same problem. The students must then analyse and compare the three solutions to find the ‘best’ solution. As a final step they can be asked to compare their solution to the previous year’s one and to explain why they think it is a better (or worse) solution. Alternatively, the students can be asked to adapt the prototype for a new delivery platform.
- On the other side of the spectrum students can be given two or more solutions to a problem and asked to identify the *similarities* and to *abstract* a standard for such problems from their findings.



Discussion

Before we conclude, we have to return to what Moggridge said. Will CTUDE be able to address the skills that he states a good designer should have? Will CTUDE assist the lecturer to design courses exercises, tutorials, practicals, assessments, etc. to give students the opportunity to practise design and gain some of the required design skills? We can't say for certain as this would require a long-term project following our students' future careers. But we argue that it will with-

out doubt go a long way if we use CTUDE to guide us in designing our courseware, including a variety of activities from both funnels, and in the process allowing our students to get some practical experience in variety of techniques and design methods.

To support our argument, we briefly list examples of the knowledge transformers required for each of Moggridge's required skills linking it to CTUDE:

- To frame and reframe the objectives: Abstraction and detailing, associations, explanation, generalization, etc.
- To create and envision alternatives: Detailing, disassociation, randomization, discovery, decomposition, specialization, dissimilarity comparison, transformation of designs, etc.
- To select from those alternatives, knowing intuitively how to choose the best approach: Association, derivations, explanation, clustering, similarity comparison, etc.
- To visualize and prototype the intended solution: Abstraction, explanation, clustering, generalization, etc.
- To synthesize a solution from the relevant constraints, understanding everything that will make a difference to a result: Preservation of designs, constrained design spaces, real-world context, etc.

Conclusion

It is our duty to equip our students with more than just the technical skills required to function in industry after completion of their studies. They also need design skills and skills to analyse designs. They need to be able to think about designs without thinking about coding – they need to be able to design with a pencil in hand instead of sitting behind a computer. We argue that the use of appropriately founded design exercises could go a long way in developing these skills.

The main contribution of this paper is CTUDE, a novel framework for design exercises with its foundations in graphical thinking, learning theory and knowledge transformers. This framework can be used as starting point contextualising the exercises we give our students to do – if we include a variety of exercises from both sides of the design continuum, we can equip our students with a wide variety of skills required to operate effectively in a work environment.

Apart from providing a structure for positioning design exercises for computing and information technology, the proposed framework also exposes a number of bare spots in computing and information technology education. While some of the aspects covered by the model (especially the 'prototyping' or engineering facets) are typically represented quite well, sketching and especially the creative 'how', are mostly missing from these programmes.

Acknowledgements

Our sincere thanks go to all the participants of the CONVIVIO faculty Forum of December 2006 who sparked the idea for our design framework.

References

- Anderson, J. R. (1993). *Rules of the mind*. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Baumann, K., Kotzé, P., Oestreicher, L., Bannon, L., Varey, A., Van Greunen, D., et al. (2007). EISH - Exercises in studying HCI. In P. Alexandra Silva, A. Dix & J. Jorge (Eds.), *Creativity3: Experiencing to educate and design - Proceedings of HCI Educators 2007* (pp. 134 – 137). Aveiro: Designeed.
- Buxton, B. (2006). What sketches (and prototypes) are and are not. *CHI'06 Workshop*. Montreal, Canada.
- Gorman, M. E. (2002). Types of knowledge and their roles in technology transfer. *Journal of Technology Transfer*, 27(3), 219 - 231.

- Kirkhart, M. W. (2001). The nature of declarative and nondeclarative knowledge for implicit and explicit learning. *The Journal of General Psychology*, 128(4), 447 - 461.
- Kotzé, P., & Purgathofer, P. (2007). A framework for design exercises – From creativity to real world use. In P. A. Silva, A. Dix & J. Jorge (Eds.), *HCI Educators 2007 - Creativity3: Experiencing to Educate and Design* (pp. 59 - 66). Aveiro: Designeed.
- Kotzé, P., Renaud, K., & Van Biljon, J. (2006). Don't do this. Pitfalls in using anti-patterns in teaching human-computer interaction principles. *Computer & Education*, doi:10.1016/j.compedu.2006.10.003.
- Laseau, P. (2000). *Graphic thinking for architects & designers*. Wiley.
- Lebiere, C., Wallach, D., & Taatgen, N. (1998). Implicit and explicit learning in ACT-R. In F. Ritter & R. Young (Eds.), *Proceedings of the Second Conference on Cognitive Modelling (ECCM 98)* (pp. 183 - 189).
- McKim, R. H. (1972). *Experiences in visual thinking*. Monterey, CA: Brookes/Cole.
- Miller, G. E. (1990). The assessment of clinical skills/competence/performance. *Acad Med*, 65, 563 - 567.
- Moggridge, B. (2007). *Designing interactions*. MIT.
- Polanyi, M. (1958). *Personal knowledge - Towards a post-critical philosophy*. London: Routledge and Kegan Paul.
- Reber, A. (1993). *Implicit learning and tacit knowledge*. Oxford: Oxford University Press.
- Schön, D. (1983). *The reflective practitioner*. Cambridge, MA: MIT Press.
- Shedroff, N. (1999). Information interaction design: A unified field theory of design. In R. Jacobson (Ed.), *Information design* (pp. 267 - 292): MIT Press.
- Sheppard, S. D. (1999 November). Design as cornerstone and capstone. *Mechanical Engineering Design*.
- Sim, K. S., & Duffy, H. B. (2002). Knowledge transformers - A link between learning and creativity. *Learning and Creativity Workshop - 2002* Retrieved 2006-12-22, from http://www.cad.strath.ac.uk/AID02_workshop/Workshop_webpage.html
- Taatgen, N. A. (1999). Learning without limits: From problem solving towards a unified theory of learning. Retrieved 2005-06-05, from www.ub.rug.nl/eldoc/dis/ppsw/n.a.taatgen/

Biography



Paula Kotzé is Professor in the School of Computing and Director of the Centre for Software Engineering at the University of South Africa. She holds a PhD from the University of York (UK), an MSc and a Honours degree in Computer Science and undergraduate qualifications in Computer Science, Industrial Psychology and Education from the University of North West (SA). Her research and teaching specialization field is human-computer interaction (HCI) and she has a keen interest in teaching methods for both HCI and computing in general. She serves in several international portfolios including as Vice-President At Large and Vice-President of Education for ACM SIGCHI, and Chairperson of IFIP WG 13.1 (HCI Education). She is an elected member of the European Academy of Science. She is the author of over 70 journal articles, peer-reviewed conference papers and edited books.



Peter Purgathofer is Associate Professor at the Design and Assessment of Technology Institute, Faculty for Informatics at the Vienna University of Technology. He works in interface design, interaction design and game design, and design theory.