# Do We Practise What We Preach in Formulating Our Design and Development Methods?

Paula Kotzé[1] and Karen Renaud[2]

[1] Meraka Institute (CSIR) and School of Computing (UNISA),
P O Box 395, Pretoria, 0001, South Africa
[2] Department of Computing Science, University of Glasgow,
Lilybank Gardens 17, Glasgow, G12 8RZ, United Kingdom
paula.kotze@meraka.org.za, karen@dcs.gla.ac.uk

**Abstract.** It is important, for our credibility as user interface designers and educators, that we practice what we preach. Many system designers and programmers remain sceptical about the need for user-centred design. To win them over, we need to be absolutely clear about what they need to do. We, as a community, propose many different methods to support naïve designers so that they will design and implement user-centred systems. One of the most popular methods is HCI design patterns – captured and formulated by experts for the sole purpose of transferring knowledge to novices. In this paper we investigate the usability of these patterns, using both theoretical and experimental analysis, and conclude that they are *not* usable. Hence, unfortunately, we have to conclude that we don't practice what we preach. We conclude the paper by making some suggestions about how we can address this situation.

**Keywords:** Design patterns, usability, learnability, memorability, efficiency, errors, satisfaction.

## 1 Introduction

In human-computer interaction we advocate that human factors must be considered during the planning, design, development, implementation and evaluation stages of interactive systems. In software engineering there is a growing awareness of human factor issues, although few of the effects of this awareness are evident in actual systems development processes and delivered system interfaces. A myriad of tools, techniques, methods, etc. are being advocated for use by designers and developers to support them in developing systems that cater for the human factor issues in interactive systems. A cursory scan of any of the prominent textbooks used in the teaching of HCI will reveal many of these techniques. Examples of these are lifecycle models, such as the Star lifecycle model by Hartson and Hix [24], the Usability Engineering Lifecycle by Mayhew [39], and the Simple Lifecycle Model of Preece *et al.* [46]. There are also design rules, such as principles to support usability [15, 44], standards [26, 27], guidelines [38, 52], golden rules [51] and heuristics [43], and HCI design patterns [18, 55, 57].

J. Gulliksen et al. (Eds.): EIS 2007, LNCS 4940, pp. 567–585, 2008.

Most of these techniques and tools attempt to address the needs of the *user* of the interactive system. There is also another angle to be considered: that of the *designer* and/or developer of the interactive system. The above-mentioned methods claim to facilitate the design of usable systems, but the question we are asking is whether these methods themselves are usable? This paper focuses on this key question: do the guidelines and principles we promote for facilitating the design of usable products apply to the very methods we advocate for the development of such usable products? Furthermore, do these methods adhere to the usability principles advocated by usability experts such as Nielsen [43]?

Using both a theoretical and experimental analysis, this paper will examine the use, by designers, of one of the most popular methods and one that has received a lot of attention in recent years: *HCI design patterns*. We will analyse design patterns from the perspective of the most widely accepted usability metrics with special attention being paid to the most relevant of these: learnability and memorability.

Section 2 introduces and discusses patterns. Section 3 describes widely accepted usability metrics. Sections 4 to 7 consider patterns from the perspective of each of these metrics in turn. Section 8 wraps up by considering how the usability of patterns can be improved. Section 9 concludes.

## 2 HCI Design Patterns

A design pattern can be defined as 'a piece of literature that describes a design problem and a general solution for the problem in a particular context' [10:2]. Designers have striven towards the elusive goal of reuse for many years now, but it only became widely achievable with the advent of the object-oriented paradigm [20] and the patterns that emerged from repeated use of successful object-orientation. The use of design patterns in HCI was a natural progression from the use of patterns in other domains and was discussed at a number of workshops in the late 1990s (for example at CHI '97, INTERACT '99, and HCI '00) [14]. An influential book by Gamma, Helm, Johnson and Vlissides [20], based on the Alexandrian format, also played a role in promoting the acceptance and use of design patterns in the field of HCI [4].

An object-oriented SE design pattern can be considered a 'solution to a general design problem in the form of a set of interacting classes that have to be customized to create a specific design' [48:225]. The definition of an HCI design pattern has a somewhat different perspective – as a proven solution for a common user interface or usability problem that occurs in a specific context of work [14].

HCI design patterns are assigned to different categories, including task representation, dialogue, navigation, information, status representation, layout, device aspects and physical interaction, user-profile, and overall system architecture [14]. A comprehensive list of HCI design patterns is available from Tidwell's collection [55], Sally Fincher's Pattern Form Gallery [18] and Van Welie's collection [57], amongst others.

Over the last few years, the idea of anti-patterns has gained favour in SE design pattern research [36, 58]. Anti-patterns capture poor or sub-optimal design or software development practices, and many also explain why such practices appear attractive to a novice and why they turn out to be a bad solution [6, 9]. The basic rationale in

publishing anti-patterns is to identify recurring design flaws for the purpose of preventing other people from making the same mistakes. An anti-pattern is therefore a pattern that 'describes a commonly occurring solution to a problem that generates decidedly negative consequences' [6:7].

HCI patterns and pattern languages are characterised by a number of features, that, it is claimed, distinguish them from rules and guidelines [2, 14, 16]:

- They capture design practise and represent knowledge about successful solutions (in the case of patterns) or unsuccessful solutions (in the case of anti-patterns).
- They encapsulate the essential common properties of good design, but do not tell the designer exactly how to do something, but rather *when* to do something and *why*.
- They represent design knowledge at varying levels, encompassing a range of issues from social issues through to widget design.
- They are not neutral but represent values within their rationale, e.g. they can express values about what is humane in interface design.
- As the concept of pattern languages is generative in nature, they can provide support in the development of complete designs.
- Patterns appear to be an effort to introduce an HCI-wide 'lingua franca'. They are, in general, claimed to be intuitive and comprehensible and it is claimed that they can therefore be used as a communication medium between various stakeholders. If this claim is true, then HCI patterns should be *accessible and understandable by end-users*. The end-users, in our context, are the designers of user interfaces.

Having given a brief overview of patterns, we now consider their usability in supporting the design process in the following sections.

## 3 Usability

The ISO 9241 Standard [26] defines usability as the effectiveness, efficiency and satisfaction experienced by a user in achieving specified goals in a specific environment. These three aspects are in line with the five attributes that contribute to usability as identified by Nielsen [42]:

1. *Learnability*: Learnability refers to the promptness with which users start performing their tasks with the system. It pertains to the features allowing novice users to understand how to use the system initially and how to attain a maximal level of performance once the system has been mastered [15]. This aspect is directly related to short-term memory and the skill acquisition process.
2. *Memorability*: Memorability refers to how easy it is to remember how to use a system feature, once learned [46] and the effort required to reuse the system feature after not having used it for some time. This aspect is directly related to long-term memory and skill retention. If something is memorable, it can be recalled with little conscious effort.
3. *Efficiency*: Efficiency refers to the level of productivity, i.e. the resources spent in relation to the accuracy and completeness of the goals achieved [26]. Efficiency therefore refers to the ways in which a system supports users in carrying out their

tasks [46]. The kinds of resources we usually measure are time and monetary cost to the user.

4. *Errors*: Users should be able to use the system with accuracy without making undue errors, and, if errors *are* made, they should be able to recover from them and still achieve their goals with minimal disruption.
5. *Satisfaction*: Satisfaction refers to the comfort and acceptability of the user-system interaction process, as well as the effects on other people affected by its use [26]. This is also related to the cognitive load placed on a user by the system – if the cognitive load is high, users will generally feel dissatisfaction.

The following section will consider learnability and memorability issues, since both are related to memory and therefore cannot be separated. For example, a system cannot be memorable unless it is easily mastered – and it needs to exhibit a high level of learnability to support this.

## 4 Learnability and Memorability

### 4.1 How Do Humans Learn?

To judge anything in terms of learnability and memorability, we must first understand how humans learn and remember things, i.e. how we form mental models and how knowledge transfer takes place.

People store what they know in mental models, which are small-scale psychological representations of real, hypothetical, or imaginary situations [12]. The mind constructs mental models as a result of perception, imagination and knowledge, and the comprehension of discourse [28, 29] in order to be able to anticipate events, to reason and to underlie explanation. It is therefore reasonable to assume that we construct mental models to represent HCI patterns (and anti-patterns) in a problem context.

People 'learn' by repeated exposure to concepts using one of two major types of learning: implicit or explicit:

- *Implicit learning*, or unintended learning or tacit (silent) learning [45, 47], can be seen as a passive process where people, when exposed to information, simply acquire knowledge of the information by means of that exposure, i.e. it is unconscious and always active [30, 47, 54]. Invoking implicit knowledge involves the indirect application of the knowledge without the requirement of knowledge declaration [30]. This aspect is thus related to the memorability of a system.
- *Explicit learning*, or intended learning, in contrast, is characterised by people actively seeking out the structure of any information presented to them, i.e. it is intentional and conscious [3, 30, 54]. For example, explicit learning would be involved if a designer is instructed to acquire some target knowledge and then explicitly to apply and state the knowledge acquired in design phase [30]. This aspect is related to the learnability of the system.

An alternative perspective on learning, closer to the process of learning as supported by HCI design patterns, is presented by Gorman [23], who identifies four types of knowledge in technology transfer:

1. *Declarative knowledge* (*what*) 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) [33]. In the context of design patterns this is the process of learning about a design pattern – its name, its rationale, its recommended application.
2. *Procedural knowledge* (*how*) that refers to the skill of knowing how to do something. Procedural knowledge is usually encoded as declarative knowledge first and then translated into procedures (algorithms) [1], 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 [33]. In the context of design patterns this is the process of learning how to use the design pattern.
3. *Judgement knowledge* (*when*) that 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 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. [33]. In the context of design patterns this is the process of learning to recognise situations where the previously learnt pattern should be applied.
4. *Wisdom* (*why*) knowledge refers to meta-cognitive monitoring which may lead to a new course of action. It is related to judgement knowledge referring to the ability to reflect, question, and come up with new courses of action. It involves an element of moral reasoning. [33]. In the context of design patterns this is the process of understanding the rationale of the pattern, and understanding why it comprises a good and effective design.

This model is confirmed by Miller [41] in his 'pyramid of competence'. Miller was concerned with the assessment of medical students. He proposes 4 levels of competence:

1. *Knows* – factual knowledge.
2. *Knows how* – ability to apply the knowledge.
3. *Shows how* – ability to identify situations where knowledge can be applied.
4. *Does* – ability to use the skills in everyday medical practice.

Level 1 aligns well with Gorman's 'what' level. Gorman's 'how' level encompasses level 2 of Miller's pyramid while level 3 accords well with Gorman's 'when' level. Finally Gorman's 'why' level can be thought to be somewhat similar to level 4 – the 'does' level (see Fig. 1). Interestingly, both Miller and Gorman communicate the concept of different kinds of knowledge building onto each other, and the acquisition of the knowledge being acquired in a particular sequence over a period of time.

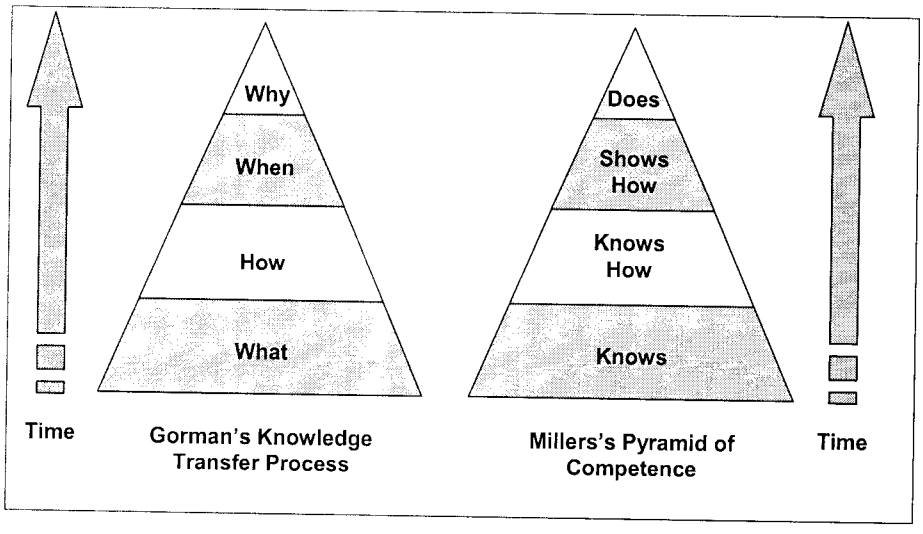


**Fig. 1.** Gorman and Miller's perspectives on knowledge transfer models

The distinction between declarative and procedural knowledge maps roughly onto the distinction between explicit and implicit knowledge since declarative knowledge is generally accessible (and therefore explicit) while procedural knowledge is generally inaccessible (and therefore implicit). It is, however, not uncommon for implicit learning also to require declarative knowledge, although there is no consensus as to the function or the source of the declarative knowledge [30]. The development of judgement knowledge is also implicit, and occurs over a period of time during the process of applying declarative and procedural knowledge to problems or instances, and whilst experience is gained in the use of this knowledge. Wisdom is tacit knowledge and therefore implicit [23]. Wass *et al.* [59] refer to Miller's pyramid of competence and point out the difficulty of assessing whether a student has reached competence in the top-most level of the pyramid. They argue that, even if the student is able to pass exams testing the first two competencies and is observed treating a patient to test the third level ('shows how'), this still does not guarantee competence at the apex of the pyramid. The implication is that the 'does' competence does not follow automatically from the student having mastered the knowledge this builds on. This appears to imply that the 'does' competence is implicitly mastered, unlike the explicitly studied knowledge it builds on. In this context, Fig. 2 gives a graphical representation of the relationship between implicit and explicit learning and the four knowledge types identified by Gorman [23].

Whether or not implicit or explicit learning is involved, one cannot present a concept only briefly and expect it to be encoded and available for retrieval after any significant interval without any further effort. There has to be an effort made in order to encode the information. If, during the encoding process, the new concept is linked to already-encoded knowledge, the retrieval process becomes easier and more likely at a later stage. Repeated exposure to a concept strengthens the encoding and makes retrieval faster and stronger, i.e. memorability is improved.

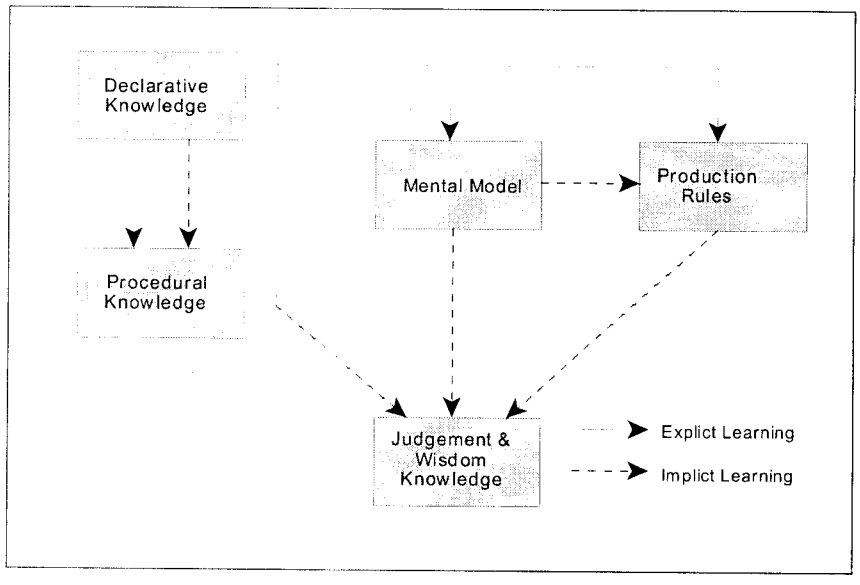


**Fig. 2.** The relationship between different types of learning and knowledge types

Fig. 1 aims graphically to depict the knowledge transfer/acquisition process using Gorman's [23] and Miller's classifications and their relationship with time. It indicates that time is required to form procedural knowledge based on acquired declarative knowledge, and then judgement and wisdom knowledge built on these. We can therefore realistically use the Gorman model, as presented in this figure, to evaluate the learnability and memorability of HCI design patterns and anti-patterns.

### 4.2 Knowledge Encapsulated in HCI Design Patterns

A pattern aims to encompass all the different types of knowledge enumerated by Gorman [23]. The procedural and declarative knowledge types can be taught and learnt but the judgement and wisdom knowledge can only be assimilated over time. It therefore is clear that novice designers master the declarative and, to a small extent, the procedural pattern-related knowledge, but that they do not develop judgement knowledge very quickly. This is probably due to the fact that the only way to develop judgement knowledge is by making use of the declarative and procedural knowledge over a period of time. Gorman [23] explains that judgement knowledge is developed gradually over a long period of time, so it is perfectly understandable that novice designers cannot develop this knowledge simply because they have been given a book of design patterns to read. Judgement knowledge is implicit – and is developed in the process of using explicit knowledge repeatedly, in context.

However, given the fact that patterns *are* being used as a knowledge transfer artefacts, let us consider how a novice designer might assimilate the knowledge captured in the pattern.

A novice designer's receptivity to the pattern creator's envisaged transfer of pattern-encapsulated knowledge will depend absolutely on how well the pattern is formulated and how strongly it is linked to the problem for which the pattern is the solution. The efficacy of the pattern, therefore, does *not* depend on the technical brilliance of the implemented design, but rather on the quality of the mental model the user constructs as a result of the way in which the pattern is structured and presented. This internalised mental model will be matched against future design problems encountered by the novice, and used if the problem matches the potential solution proffered by the model. If the model is sufficiently well captured, there is a better chance of the learner identifying it and using it. Hence, the efficacy of any design pattern's knowledge transfer process depends on how well the issues in the pattern are communicated to the learner *at the first encounter*, which is when the pattern is first understood and internalised, and the mental model constructed [56].

The tricky problem in the formulation of effective patterns therefore lies in ensuring that the formulation satisfies the needs of naïve user interface designers. Experts often omit essential details, simply because they assume knowledge of these facts. The efforts of many researchers in the field of HCI design patterns have been aimed at closing this communication gap [17, 50]. When we consider the use of patterns in HCI knowledge transfer, the closing of this gap becomes essential.

Fig. 3 contrasts the knowledge transfer model (as illustrated in Fig. 1) with the general presentation structure of HCI design patterns [55, 57]. We used the Tidwell HCI Pattern Definition format [55] as example format, but other HCI design pattern formats have a similar structure.

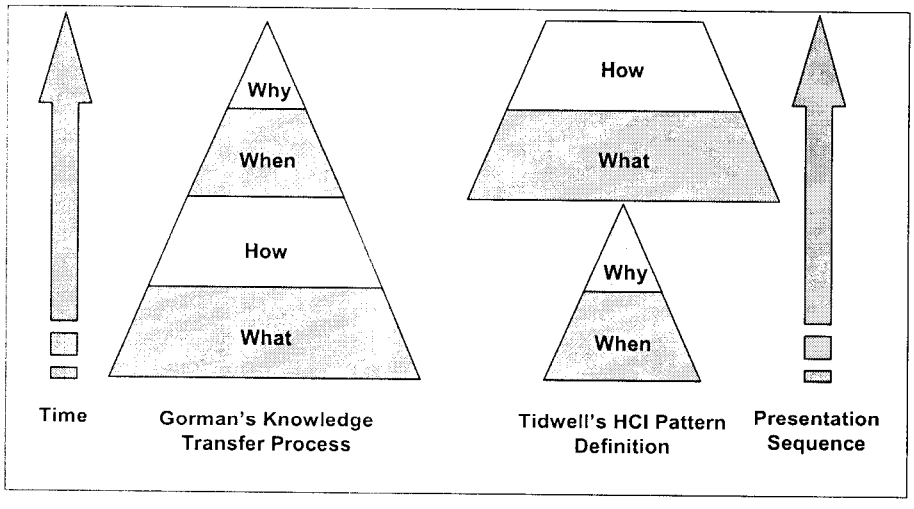


**Fig. 3.** The pattern presentation sequence vs. the knowledge transfer process

When we study Fig. 3 closely, we uncover what may be the primary reason for the difficulties many naïve designers have with comprehending and using patterns. The order in which information is presented in patterns, and the assumptions of embedded knowledge linked to this imposed order, simply do not align with the knowledge transfer process, which needs to occur in a specific sequence. Patterns typically

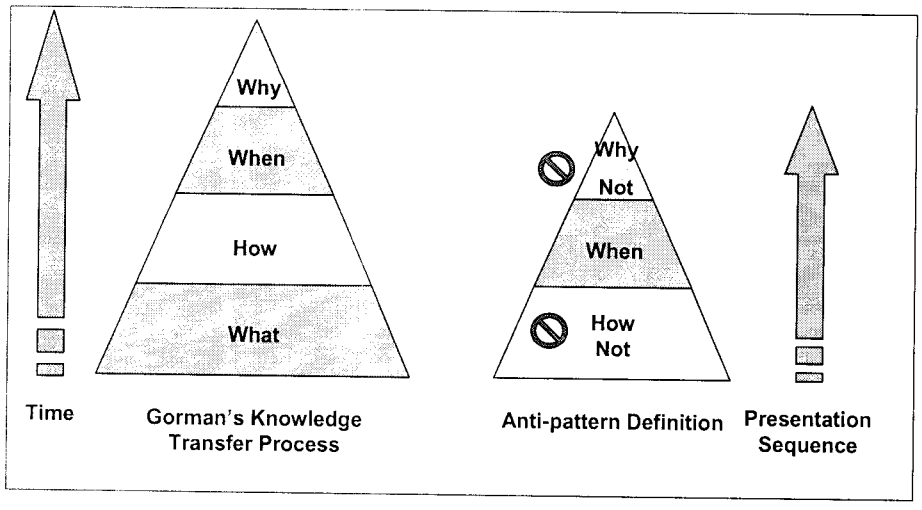


**Fig. 4.** The anti-pattern presentation sequence vs. the knowledge transfer process

introduce first the 'when' and the 'why' and this assumes prior mastery of the 'what' and the 'how'. Patterns appear, at first glance, to accord well with human information processing processes because they include information related to all the mental model knowledge representation processes. However, their knowledge presentation structure does not align correctly with the accepted knowledge acquisition process and this could impair their efficacy.

This problem is even more severe when anti-patterns are contemplated, since the cognitive processing of anti-patterns has to deal with negation. An anti-pattern theoretically shows how to do the 'opposite' of the required solution or 'how not to do it' (not necessarily the opposite of any proper solution).

The negation schema involved with anti-patterns is the schema-plus-tag model [32]. The schema-plus-tag model states that the core supposition of a premise is processed as a cognitive unit, which is then marked with a negative tag [8, 40]. The critical issues are the argument that the core can be disassociated from the negation tag at a later stage (and as result the individual might remember the opposite of the intended meaning), and that the consideration of the core supposition activates associations congruent with the core, but incongruent with the intended meaning of the negation as a whole. The negation of a premise is therefore kept as a 'mental footnote' in the designer's mind, whereas the solution itself is kept as a mental model. These tags sometimes fail to activate and can lead to systematic errors and illusions. For example, if you tell a designer: 'don't use red print on a green button', the designer has to think about the green button with red print on it before storing it with the footnote reminding him/her of the folly of this course of action. According to the schema-plus-tag model we tend to internalise what we focus on, so when the designer thinks of colour schemes for a button s/he may well use a green button with red print because the mental trace to that concept has survived but the footnote has failed to activate.

Fig. 3 demonstrated the inherent defects related to the commonly used design pattern structure. Fig. 4 compares the anti-pattern structures to Gorman's knowledge transfer process. There are two things to be noted about this comparison:

1. Two 'not' tags are used – 'how not' and 'why not'. This invokes the use of the schema-plus-tag negation model, and either or both tags could thus easily go missing.
2. The anti-pattern assumes prior knowledge of the 'what', which, in a novice, cannot be assumed (the 'what' knowledge is not explained or referenced in an anti-pattern presentation).

The effects of anti-patterns on novice designers, therefore, could be confusing, at least, and detrimental, at worst.

### 4.3 Learnability and Memorability of HCI Design Patterns and Anti-patterns

From the arguments above it seems as if design patterns will indeed exhibit problems when assessed for learnability and memorability. But is this indeed the case?

In researching the practice of teaching in the negative we did a number of experiments with the teaching of patterns and anti-patterns and observed how students learn based on the mode of teaching. The results of these experiments are described in detail in Kotzé, Renaud and Van Biljon [32], but we will highlight our findings here to support our argument that the learnability and memorability of design problems may be suspect.

- The work of a third year group of software engineering students at the University of Glasgow was observed and serves to illustrate the pattern knowledge transfer process. Students were randomly allocated to groups of five to do a project during their third year. The project entailed the design and implementation of a project management system. Students were taught basic software engineering and HCI design patterns and given examples of their use in a graphical user interface. Although a group project, the students were required to write an individual report about what they learnt during the project, including the role of patterns. Only one of the students reported making use of the full complement of patterns (they could use 5 in the exercise). But what is more interesting is that the student's team members did not report using the same 5 patterns. Even though students had two lectures on patterns, and the lecture notes were also freely available on the module website, only 28% of the students appear to have made use of patterns in their group project. It is possible that students made use of patterns and then did not report it, but this is unlikely because it was an explicitly mentioned topic. The only conclusion we can draw from this is that students had the theoretical knowledge but had difficulty applying it. The discussion on different knowledge levels above offers some explanation for this phenomenon – students master declarative knowledge and, to a lesser extent, procedural knowledge, but they do not develop judgement knowledge.
- Two experiments were conducted on teaching patterns and anti-patterns with third-year Computing Science students at the University of Glasgow: an intra-group study and an inter-group study. The intra-group study found that students had difficulty in applying guidelines stated in the negative, in contrast with guidelines stated in the positive, which resulted in fewer errors. The inter-group study had two groups of students receiving group tutorials separately, either being taught using positive HCI design pattern-like information or anti-pattern like information. Table 1 depicts, as

percentages, the difference between the average scores of the students in the patterns group and those in the anti-patterns group for each of the assessed components. It is clear from the results in this table alone that the students in the patterns group performed significantly better in all of the assessed concepts than did the students in the anti-patterns group. But what is also clear is the extremely low performance even in the group that were taught with patterns, i.e. positively.

**Table 1.** Comparing the marks (as percentages) of students in the anti-pattern group and the pattern group per component

| | Use of Colour | Instructions given | Button Design | Error Reporting |
|---|---|---|---|---|
| Anti-Patterns | 39 | 41 | 22 | 46 |
| Patterns | 47 | 54 | 37 | 59 |

The findings of these experiments can be criticised for not focusing on the usability issues directly, and therefore we conducted a survey with another group of 17 third-year Computing Science students at the University of Glasgow focussing specifically on their experiences with patterns. This survey was done within two weeks of their receiving a number of lectures on patterns. When asked '*how easy it was to understand design patterns when first taught*', 12 of the 17 found it to be difficult, while only 1 thought it was easy. More than half of the students did not understand the rationale behind specific patterns. When asked '*how easy is it to remember patterns that were taught after a week or two*', the overwhelming response was that it 'was hard' (only 2 though it was relatively easy). They also had problems in remembering the patterns they were taught the year before. They forgot either the rationale behind the patterns they were taught or the design method it represented, or both.

Evidence from these experiments, and from the theoretical foundations, therefore show that HCI design patterns and anti-patterns could be deficient with respect to learnability and memorability. This leads us to the inescapable conclusion that HCI design patterns and anti-patterns *do not* meet the first two of Nielsen's [43] usability attributes.

In the next three sections we will briefly look at the other three attributes of usability, namely efficiency, errors and satisfaction and consider the extent to which HCI design patterns adhere to these attributes.

## 5 Efficiency

Efficiency refers to the level of productivity, i.e. the resources spent in relation to the accuracy and completeness of the goals achieved [26]. Efficiency also refers to the ways in which a system supports users in carrying out their tasks.

For a pattern language to be efficient in generating solutions it should be generative, allowing users to develop new solutions, and provide a taxonomy enabling the user to easily locate relevant core patterns, to find related or proximal patterns, and to evaluate the problem from different standpoints [19].

The organization of pattern languages in HCI is particularly problematic because of the wide range of different levels that have to be addressed by HCI design patterns, from the broader social context in which an interactive system is used, to the low-level details of interaction itself [14].

Efficiency is therefore related to the completeness of the pattern languages. This is particularly problematic in HCI design patterns, as no coherent pattern language exists. There are a lot of competing voices and individual (and often repeated) efforts [14]. This is often as a result of the demands on researchers to publish and own work. Although pattern language development needs to be a community effort, the competitive pressures within the wider research context can mediate against such a cooperative approach [2].

Unless a collaborative process can be developed in future whereby participants can select and develop the patterns towards a coherent pattern language, HCI design patterns will continue to fail to meet the efficiency usability attribute.

Our experiences [32] suggest that poor knowledge transfer by means of the use of patterns can be attributed directly to the fact that students do not develop judgement knowledge in the short period of time allowed for teaching a concept. Furthermore, we also argued that anti-patterns confused students and did more harm than good.

During our survey amongst the third-year Computing Science students we asked them '*how difficult it is to match design problems to the patterns you were taught when you are designing software now?*' Only 4 of the 17 students found it relatively easy – the other 13 found it very hard. When they were asked whether they '*get frustrated when they have to try to find a pattern to match a problem*', 12 of the 17 expressed dissatisfaction and frustration with matching patterns to problems.

In terms of efficiency and efficacy in knowledge transfer and use, therefore, patterns have yet to prove their worth.

## 6 Errors

When we introduced the concept of patterns in section 2, we referred to two types of patterns, namely *patterns* and *anti-patterns*. There is, however, a third type of pattern, called an *amelioration pattern*. An amelioration anti-pattern tells the reader how to go from a bad solution to a good solution. It defines a migration path (or refactoring) from a negative to a positive solution. It tells you why the bad solution appeared viable in the first place, why it turned out to be bad in conjunction with the desired new outcome or behaviour, and what positive patterns are applicable instead [6]. Amelioration anti-patterns are only required because people fail to locate the correct pattern and then apply the wrong pattern, or, if they do manage to match the correct pattern to the problem, they apply it incorrectly.

The mere existence of amelioration patterns hints at problems with the usability of HCI design patterns. Recovering from a problem should not require the designer to look up a solution from yet another set of HCI design patterns. On the positive side, if an amelioration pattern exists for a specific problem or incorrectly applied solution, it will provide the designer with a 'way out' when things go badly wrong or when the designer does not know how to correct an obvious mistake. At present there are, unfortunately, only a small number of amelioration HCI design patterns in existence.

In terms of errors, once again HCI design patterns do *not* prove to be the silver bullet of design – confirming Fred Brooks' [5] prediction that design, being inherently complex and difficult, will never be eased by one particular innovation or tool.

## 7 Satisfaction

Cognitive load is high when designers are working on a project within limited time constraints, and this has been proved to be counter-productive for the interpretation of false or negated information [21, 22], or detailed information requiring the designers to choose between various option (e.g. choosing the correct HCI design pattern for a specific interaction design).

For seasoned designers who have developed judgement and wisdom knowledge this should not be a problem, but for novice designers who are still attaining and developing such knowledge, it might lead to a high degree of dissatisfaction if they cannot easily identify a suitable design pattern. Furthermore it is likely that they simply will not understand how to use it or why it should be used.

Although all but 3 of the students in our survey saw the point of learning patterns, the majority of them (12 of the 17) found patterns to be obscure.

Dearden and Finlay [14] argue that one of the most obvious weaknesses of HCI designs patterns is the lack of substantive evidence as to the benefit of using them in actual design practice. Considerable attention has focused on generating patterns and developing various individual pattern languages, rather than on their use in practice. Significant effort is now required to examine the use of these languages in actual design (e.g. via empirical and observational studies) and in education to demonstrate what, if any, benefits might be gained from a patterns-led approach. We argue that satisfaction levels will stay low until these benefits have been proven.

## 8 Improving Pattern Usability

From the arguments above we have to conclude that HCI design patterns do not meet any of the basic usability principles or attributes. Our investigations have also convinced us that patterns are neither efficient nor efficacious in transferring expert HCI design knowledge to naïve designers.

Should we give up on patterns altogether? Not at all! We should simply be more realistic and circumspect about their use.

We can compare the process of learning how to design systems with language acquisition, albeit on a very superficial level. People learn a new language starting by mimicking particular words. Only once they have accumulated a fair number of commonly used words, and built up a bare framework of the language, and used it for some time, can they start to understand more intricate formalisms such as sentences, tenses and grammar.

Perhaps we can learn a lot from the way schools have changed how they teach in the last 40 odd years. Crystal [13] provides some interesting insights into the changing modes of language instruction. Before the 1960s children were taught grammar – given sentences to analyse in terms of grammatical constructs. Those of us

who experienced this approach often remember it with a sense of repugnance. Grammar was reduced to a set of rules but the meaning and richness of the language was never experienced or understood. Between the 1960s and the mid 1990s children were taught no grammar at all. This too was found to be unsatisfactory because one needs an understanding of grammar to understand the immense creative power of language. A comprehensive study of grammar also helps us to master second and third languages. Consequently, in the late 1990s the approach changed once more, to reintroduce grammar into the curriculum. Only now, a different, more effective paradigm was applied – discovery-based learning. Grammar was no longer merely prescriptive, but was introduced to help students to understand meanings and effects of different constructs in communicating and language. The paradigm was: discovery first, definitions of terms last.

The fact is that we learn in a stepwise fashion, learning rudimentary skills (declarative and procedural) first, then we learn by doing and by watching others more skilled than ourselves (moving towards judgement and implicit procedural skills) and then, only once we have mastered the basics and used them over a period of time, can we be said to have the basic skills to start looking at formalisations such as patterns (once we have the judgement knowledge.)

Someone learning to design interfaces will learn information about basic widgets, and accumulate an understanding of basic HCI principles in a discovery-based way. Only once they are fully conversant with the basic building blocks of the interface can they start thinking about formalisms such as using basic concepts in conjunction with each other to create more complicated artefacts that are, nevertheless, usable. Only once they have spent some time engaged in this process will they be ready for the pattern formalisms and for understanding patterns which bring all the different concepts together in a structured way.

Since we've argued that patterns are contra-indicated for naïve designers, what should we do to direct them and prevent them from making errors? *We should provide them with rules and guidelines, which are easily understood and applied.* We should provide them with a mentor – a seasoned designer to guide their discovery process.

This is not an arbitrary recommendation. There is empirical evidence that guidelines may be easier to use and more effective than patterns [11, 60]. There is little evidence that interfaces produced by using HCI design patterns are better than interfaces designed using guidelines [11]. Koukouletsos, Babak and Dearden [31] also found that patterns, being longer in text and more difficult to assimilate, are harder for novice designers to comprehend. Novice designers need to undertake an extra mental process when contemplating the use of a pattern. Patterns need to be analysed and well understood to be efficacious. Guidelines do not suffer from these problems. The University of California studies in the early 1990's on teaching with or without patterns also confirm this [7, 25, 34, 35, 37, 49]. The group's overall finding was that patterns need rich connections to examples and multiple links to context of use if they were to be effective in teaching. If patterns are too narrow or inflexible, novices have difficulty abstracting from them and would rarely use them. It is generally accepted that the way in which expert programmers work has a great deal more to do with large 'libraries' (patterns) they have built up over time of stereotypical solutions to

problems, as well as strategies for coordinating and composing them, than the mere syntax and semantics of language constructs [53]. If novice students are to mature into expert programmers, they should be taught explicitly about building up these libraries and developing strategies for activating them.

We therefore argue that HCI design patterns should be recorded by experienced designers but should not be inflicted on naïve designers – rather they should be available for use by seasoned designers, those who have attained a particular proficiency in the language of design – much as colloquialisms are understood only by people who have attained a high level of proficiency in a particular language. In the same way, patterns can only really be comprehended and correctly applied by people who have attained a high level of proficiency in the language of design.

## 9 Conclusion

It is clear that HCI design patterns are basically unusable by their currently targeted audience, since they do not exhibit the basic characteristics of usability, as defined by Nielsen [42].

If HCI design patterns were to be representative of the design and development methods promoted for the design of interactive systems then the answer to our question '*do we practise what we preach in formulating our design and development methods*' should be in the negative: and the obvious conclusion should be that we unfortunately do *not* practice what we preach.

Unfortunately this does not apply to HCI design patterns only – the same might be said about design patterns in general, as was exhibited in the University of California studies. As educators and mentors, we should consider these findings carefully and we should be more careful about recommending a technique that we, as experts, find helpful, in the mistaken belief that it will be equally helpful to novices.

## References

1. Anderson, J.R.: Rules of the Mind. Lawrence Erlbaum Associates, Hillsdale (1993)
2. Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., Gross, B., Lehder, D., Marmolin, H., Moore, B., Potts, C., Skousen, G., Thomas, J.: Putting it all together: Towards a pattern language for interaction. SIGCHI Bulletin 30(1), 17–33 (1998)
3. Berry, D.C.: How Implicit is Implicit Learning? Oxford University Press, Oxford (1997)
4. Borchers, J.A.: Teaching HCI Design Patterns: Experience from Two University Courses. In: Patterns in Practice: A Workshop for UI Designers (at CHI 2002 International Conference on Human Factors of Computing Systems). City (2002)
5. Brooks, F.P.: The Mythical Man Month and Other Essays on Software Engineering. Addison Wesley, Reading (1995)
6. Brown, W.J., Malveau, R.C., McCormick III, H.W., Mowbray, T.J.: AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis. John Wiley & Sons, Inc., New York (1998)
7. Clancy, M.J., Linn, M.C.: Patterns and pedagogy. ACM SIGCSE Bulletin (3/99), 37–42 (1999)

8. Clark, H.H., Chase, W.G.: On the process of comparing sentences against pictures. Cognitive Psychology 3, 472–517 (1972)
9. Cockburn, A., Baruz, A., Engelund, A., Hanes, P.B., Brown, C., Siska, C., Olson, D., Xexeo, G., Lowe, I., Chapman, J., Coplien, J.O., Holloway, J., Brown, K., Eichin, M., Phillips, R., Jeffries, R., Gordon, S., McCormick III, H.W.: Antipattern (2005) [cited 2005 2005-12-12], http://c2.com/cgi/wiki?AntiPattern
10. Coplien, J.O.: Software Patterns. SIGS Books & Multimedia, New York (1996)
11. Cowley, N.L.O., Wesson, J.L.: An experiment to measure the usefulness of patterns in the interaction design process. In: Costabile, M.F., Paternó, F. (eds.) INTERACT 2005. LNCS, vol. 3585, pp. 1142–1145. Springer, Heidelberg (2005)
12. Craik, K.: The Nature of Explanation. Cambridge University Press, Cambridge (1943)
13. Crystal, D.: How Language Works. Penguin, London (2005)
14. Dearden, A., Finlay, J.: Pattern Languages in HCI: A critical review. Human-Computer Interaction 21(1), 49–102 (2006)
15. Dix, A., Finlay, J., Abowd, G.D., Beale, R.: Human-computer Interaction. Pearson Education Limited, Harlow (2004)
16. Dix, A., Finlay, J., Abowd, G.D., Beale, R.: Human-Computer Interaction, 3rd edn. Pearson Education Ltd., Harlow (2004)
17. Faridul, I.: Investigating XML as Language for HCI Patterns Representation. Concordia University, City (2003)
18. Fincher, S.: The Pattern Gallery (2000) [cited 2005-12-12], http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html
19. Fincher, S., Windsor, P.: Why patterns are not enough: some suggestions concerning an organising principle for patterns of UI design. In: CHI 2000 Workshop on Pattern Languages for Interaction Design: Building Momentum (2000)
20. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading (1995)
21. Gilbert, D.T., Krull, D.S., Malone, P.S.: Unbelieving the unbelievable: some problems in the rejection of false information. Journal of Personality and Social Psychology 59, 601–613 (1990)
22. Gilbert, D.T., Tafarodi, R.W., Malone, P.S.: You cannot believe everything you read. Journal of Personality and Social Psychology 65, 221–233 (1993)
23. Gorman, M.E.: Types of knowledge and their roles in technology transfer. Journal of Technology Transfer 27(3), 219–231 (2002)
24. Hartson, H.R., Hix, D.: Toward empirically derived methodologies and tools for human-computer interface development. International Journal of Man-Machine Studies 31, 477–494 (1989)
25. Hoadley, C.M., Linn, M.C., Mann, L.M., Clancy, M.J. (eds.): When, why, and how do novice programmers reuse code? In: Gray, W.D., Boehm-Davis, D. (eds.) Empirical Studies of Programmers, vol. 6. Ablex, Norwood (1996)
26. International Organization for Standardization: ISO9241: Ergonomic requirements for office work with visual display terminals (VDTs) (1997) [cited 2006-12-01], http://www.iso.org/iso/en/iso9000-14000/index.html
27. International Organization for Standardization: ISO14915: Software ergonomics for multimedia user interfaces (2002) [cited 2006-12-01], http://www.iso.org/iso/en/iso9000-14000/index.html
28. Johnson-Laird, P.N.: Mental Models. In: Posner, M.J. (ed.) Foundations of Cognitive Science, pp. 469–499. MIT Press, Cambridge (1989)

29. Johnson-Laird, P.N., Girotto, V., Legrenzi, P.: Mental Models: A Gentle Guide for Outsiders (1998), http://www.si.umich.edu/ICOS/gentleintro.html
30. Kirkhart, M.W.: The nature of declarative and nondeclarative knowledge for implicit and explicit learning. The Journal of General Psychology 128(4), 447–461 (2001)
31. Kotzé, P., Renaud, K., Koukouletsos, K., Khazaei, B., Dearden, A.: Patterns, anti-patterns and guidelines: Effective aids to teaching HCI principles? In: Hvannberg, E.T., Read, J.C., Bannon, L., Kotzé, P., Wong, W. (eds.) Inventivity: Teaching theory, design and innovation in HCI - Proceedings of HCIEd2006-1 (First Joint BCS / IFIP WG 13.1 / ICS /EU CONVIVIO HCI Educators Workshop, pp. 115–120. University of Limerick, Limerick (2006)
32. Kotzé, P., Renaud, K., Van Biljon, J.: Don't do this - Pitfalls in using anti-patterns in teaching human-computer interaction principles. Computer & Education (2006), doi:10.1016/j.compedu.2006.10.003
33. Lebiere, C., Wallach, D., Taatgen, N.: Implicit and explicit learning in ACT-R. In: Ritter, F., Young, R. (eds.) Proceedings of the Second Conference on Cogntive Modelling (ECCM 1998), pp. 183–189 (1998)
34. Linn, M.C.: How can hypermedia tools help teach programming? Learning and Instruction 2, 119–139 (1992)
35. Linn, M.C., Clancy, M.J.: The case for case studies of programming problems. Communications of the ACM 35(3), 121–132 (1992)
36. Mahernoff, M.J., Johnston, L.J.: Principles for a usability-oriented pattern language. In: Proceedings of the Australasian Computer Human Interaction Conference, Adelaide, pp. 132–139 (1998)
37. Mann, L.M.: The Implications of Functional and Structural Knowledge Representations for Novice Programmers. In: Graduate Group in Science and Mathematics Education. University of California, City (1991)
38. Mayhew, D.J.: Principles and Guidelines in Software and User Interface Design. Prentice Hall, Englewood Cliffs (1992)
39. Mayhew, D.J.: The Usability Engineering Lifecycle. Morgan Kaufmann, San Francisco (1999)
40. Mayo, R., Schul, Y., Burnstein, E.: I am not guilty vs I am innocent: Successful negation may depend on the schema used for its encoding. Journal of Experimental Psychology 40, 433–449 (2003)
41. Miller, G.E.: The assessment of clinical skills/competence/performance. Acad. Med. 65, 563–567 (1990)
42. Nielsen, J.: Usability Engineering. Academic Press, Boston (1993)
43. Nielsen, J.: Heuristic evaluation. In: Nielsen, J., Mack, R.L. (eds.) Usability Inspection Methods. John Wiley & Sons, New York (1994)
44. Norman, D.: The Design of Everyday Things. MIT Press, London (1998)
45. Polanyi, M.: Personal Knowledge - Towards a Post-Critical Philosophy. Routledge and Kegan Paul, London (1958)
46. Preece, J., Rogers, Y., Sharp, H.: Interaction Design: Beyond Human-computer Interaction. John Wiley & Sons, Inc., New York (2002)
47. Reber, A.: Implicit learning and tacit knowledge. Oxford University Press, Oxford (1993)
48. Schach, S.R.: Object-oriented and Classical Software Engineering, 6th edn. McGraw Hill Higher Education, New York (2005)
49. Schank, P.K., Linn, M.C., Clancy, M.J.: Supporting Pascal programming with an on-line template library and case studies. International Journal of Man-machine Studies 38, 1031–1048 (1993)

50. Seffah, A.: Learning the ropes: human-centered design skills and patterns for software engineers education. Interactions 10(5), 36–45 (2003)
51. Shneiderman, B.: Designing the User Interface. Addison-Wesley, New York (1998)
52. Smith, S.L., Mosier, J.N.: Guidelines for designing user interface software. Mitre Corporation Report, MTR-9420 (1984)
53. Soloway, E.: Learning to program = learning to construct mechanisms and explanations. Communications of the ACM 29(9), 850–858 (1986)
54. Taatgen, N.A.: Learning without limits: from problem solving towards a unified theory of learning (1999) [cited 2005-06-05], http://www.ub.rug.nl/eldoc/dis/ppsw/n.a.taatgen/
55. Tidwell, J.: Designing Interfaces: Patterns for Effective Interaction Design (2005) [cited 2006-07-01], http://designinginterfaces.com/
56. Van Biljon, J., Kotzé, P., Renaud, K., McGee, M., Seffah, A.: The use of anti-patterns in human computer interaction: wise or ill-advised? In: Marsden, G., Kotzé, P., Adesina-Ojo, A. (eds.) Fulfilling the promise of ICT, SAICSIT (ACM Conference Proceedings Series), Pretoria, pp. 176–185 (2004)
57. Van Welie, M.: Patterns in Interaction Design (2006) [cited 2006-07-01], http://www.welie.com/
58. Van Welie, M., Van Der Veer, G.C.: Pattern languages in interaction design: structure and organization. In: Rauterberg, M., Menozzi, M., Wesson, J. (eds.) Human-computer interaction, INTERACT- 2003, pp. 527–543. IOS Press, Amsterdam (2003)
59. Wass, V., Van Der Vleuten, C., Shatzer, J., Jones, R.: Assessment of Clinical Competence. The Lancet 357, 945–949 (2001)
60. Wesson, J., Cowley, N.L.O.: Designing with patterns: Possibilities and pitfalls. In: 2nd Workshop on Software and Usability Cross-Pollination: The Role of Usability Patterns (2003) [cited 2005-12-23], http://wwwswt.informatik.uni-rostock.de/deutsch/Interact/05WessonCowley.pdf

## Questions

***Michael Harrison and Janet Wesson:***
*Question: About the Experiment: How do you measure the quality of the resulting product?*

Answer: We checked the product for obvious mistakes such as violation of guidelines, mismatch of colours, etc...

***Michael Harrison:***
*Question: How did you train the students in the use of patterns?*

Answer: Students were introduced to patterns, as is general practice,during lectures. To ensure that they understood how to apply and use the patterns, students were instructed to use them in a design exercise.

***Laurence Nigay:***
*Question: Were the inspected patterns specific to HCI?*

Answer: Yes, but the discovered flaws and limitations may also be applicable to patterns in other domains (e.g. software design).

*Question: Why is it a problem to use patterns from different languages?*

Answer: Different collections may contain patterns for the same problem, but with different (even contradicting) solutions. This can be explained by the fact that the solution stated in the pattern is bound to the overall context of the language.

***Kirstin Kohler:***
*Question: What makes you think that the problem is the way patterns are written and not the way you teach them to students?*

Answer: The teaching method appears to be representative of the methods used by most Universities to teach patterns. Those practitioners who do not attend classes usually attempt to learn patterns from a textbook. Transferring knowledge by means of patterns is the real issue, which is the core of what we are saying. People can learn a pattern as a kind of recipe to be followed, but matching that pattern to a problem is something which cannot be taught - it only develops with experience.