VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

Towards Evaluation of Object-Oriented Designs

Robert Biddle, James Noble, and Ewan Tempero

Technical Report CS-TR-02/8
April 2002

School of Mathematical and Computing Sciences
Victoria University
PO Box 600, Wellington
New Zealand

Tel: +64 4 463 5341
Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

VICTORIA UNIVERSITY OF WELLINGTON
Te Whare Wananga o te Upoko o te Ika a Maui

School of Mathematical and Computing Sciences
Computer Science

PO Box 600
Wellington
New Zealand

Tel: +64 4 463 5341, Fax: +64 4 463 5045
Email: Tech.Reports@mcs.vuw.ac.nz
http://www.mcs.vuw.ac.nz/research

Towards Evaluation of Object-Oriented Designs

Robert Biddle, James Noble, and Ewan Tempero

Technical Report CS-TR-02/8
April 2002

Abstract

This paper describes our approach to evaluation of object-oriented designs. It is based on work introducing object-orientation to industry groups, and is motivated by our observations about the needs for design evaluation in industry. We argue that design evaluation is an important but undervalued part of software development. We discuss possible techniques for both summative and formative design evaluation, and present our experience with them.

Author Information

Robert Biddle and James Noble are members of the academic staff in the School of Mathematical and Computing Sciences and the School of Information Management at Victoria University of Wellington. Ewan Tempero is a member of the academic staff of the Department of Computer Science at the University of Auckland.

For reviewer use only: ID #

ACIS 2002 PAPER SUBMISSION COVER-PAGE

Paper Title:	**Towards Evaluation of Object-Oriented Designs**		
Author Name(s):	**Robert Biddle, James Noble, Ewan Tempero**		
Contact Author: Email address: Fax: Phone:	**Robert Biddle** **Email: Robert.Biddle@vuw.ac.nz** **Fax: +64 4 463-5045** **Phone: +64 4 463-5833**		
Paper Category (circle one):	Research Paper	Research in Progress	Teaching Case
Prior Publication (delete not applicable):	The paper has NOT been presented or published elsewhere		

AUTHOR DECLARATION

By submitting this paper to ACIS 2002 the author(s) warrant that:

- they are the copyright holders in the paper;
- the paper is accurate in all factual respects;
- the contents have been cleared for publication by any organisation having the capacity to preclude their publication;
- the approval has been secured of the copyright owner of any material which has been previously published.
- they accept full responsibility for statements made both in written papers and in presentations;
- they undertake that at least one of the authors will register for the, will present the paper, and will be available to answer questions and discuss the paper with other delegates;

(Please leave blank, reviewer use only)

Towards Evaluation of Object-Oriented Designs

Robert Biddle[a]
James Noble[a]
Ewan Tempero[b]

[a]Victoria University of Wellington, New Zealand
Email: robert,kjx@mcs.vuw.ac.nz

[b]University of Auckland, New Zealand
Email: ewan@cs.auckland.ac.nz

Abstract

This paper describes our approach to evaluation of object-oriented designs. It is based on work introducing object-orientation to industry groups, and is motivated by our observations about the needs for design evaluation in industry. We argue that design evaluation is an important but undervalued part of software development. We discuss possible techniques for both summative and formative design evaluation, and present our experience with them.

Keywords

FA11 OBJECT-ORIENTED APPROACH, FB04 IS DESIGN, FB0406 IS design issues, FC29 OBJECT-ORIENTED DESIGN

INTRODUCTION

We have been exploring ways to improve techniques for the early stages of object-oriented development. In particular, we have begun to address the issue of object-oriented design *evaluation*. This is not a traditional step in the development process, but our work with industry has indicated there is a growing need for awareness and advice on this topic. In this paper we present the approach we have taken in introducing this topic in industry, outlining the principles we introduce, and the techniques we have explored.

The concept of evaluation is not new in the context of system development. Early in the system development process, the key issue is requirements elicitation and analysis, typically done by working with actual or potential customers; for example, see (Leffingwell, Widrig and Yourdon 1999). In actual system implementation, there are also established techniques for the early evaluation of quality via walkthroughs and inspections; for example, see (Weinberg and Freedman 1990). The ultimate techniques for evaluation late in the process are testing and possibly formal verification, once the system has actually been implemented.

Our observation is that there is a step in between requirements and implementation, and that is design: and there is little advice on evaluation of design. There have been international workshops to consider the issue of object-oriented design evaluation in educational settings (Biddle, Mercer and Wallingford 1999); it turns out the issue is also of great importance in industry. The need for design evaluation is especially strong now because the increased use of outsourcing means that system analysts and system designers are often now in different organisations.

The rest of this paper is organised as follows. In the next section, we outline how we motivate industry groups to understand the nature of design evaluation. In the third section we introduce the technique of heuristic evaluation, and discuss our findings from trialing this technique with industry groups. We then outline some other techniques we also explored, and discuss their significance. Finally, we present out conclusions, and our advice for future work in this area.

MOTIVATION

In the past two years we have had the opportunity to work with a number of industry groups introducing techniques for object-oriented (OO) analysis and design. We worked with about six industry groups over an eighteen month period, and the groups had approximately 15 participants each. Some participants were experienced developers, and others were experienced business analysts with no development experience.

Our initial stress was on the principles or object-orientation and to apply them in the creation of new analysis models and designs. As we worked, however, we increasingly realised the need not only to create designs, but to evaluate ones already created.

One issue arose as we developed an "active learning" approach to teaching OO design (Biddle, Noble and Tempero 2001), because the techniques we used frequently involved working with already created designs. More importantly we recognised that in current industry practice, business analysts are now commonly put in the position of working with developers outside their organisation, and thus evaluation becomes a major part of their role. As we recognised this, we adjusted our presentation, and thus our introduction to OO now features this new emphasis on evaluation, as we discuss below.

Introducing Evaluation

We introduce evaluation into our course as follows. Design evaluation is not an established step in any typical development life-cycle, so we begin with some gravity and fanfare in an effort to promote its importance.

Our introduction begins with a simple gloss and scoping. The basic idea of evaluation is determining value. We are familiar with many versions of this in everyday life, ranging from simple measuring devices to find the size of things, to justice systems to find the truth in arguments. Evaluating a system design may seem as dispassionate as making a simple measurement, but in fact it can become as complicated as a court case.

To highlight the key aspects of an evaluation process, we refer to the image of Themis, the figure in ancient Greek mythology associated with judgment. She is usually depicted with three prominent features: she is blindfolded, she holds a set of weighing scales, and she carries a sword. Together, these features serve as reminders of the ideals in evaluation, and can help us focus on some issues that arise in system design.

The blindfold indicates impartiality, especially important so that the status of what is being evaluated does not determine the result. This signifies that paupers and princes will be treated the same, but also applies to design evaluation. There are all kinds of practical reasons unrelated to design quality that may influence evaluation. Some involve outside factors, such as influence by people or organisations involved with the design. Other factors simply involve commitments that have already been made, including money spent, publicity, and even simply the effort in preparing documentation for the design under evaluation. All these are perfectly natural, but may influence approval of a design without real regard for how good the design really is. This represents a risk, because the quality of the design — good or bad — will have future consequences.

The scales indicate the process of weighing evidence carefully, so that the correct result is obtained. This shows that there is indeed an observable process that determines the result, rather than some mysterious and unfathomable result emerging without explanation. This too is important for design evaluation. The quality of a design should be determined as good or bad, better or worse, according to some understandable method, and based on explained reasons. The quality of a design cannot be often proven in advance, because it involves prediction of the future. Keeping the process open and reasoned, however, allows correction of factual mistakes, incorporation of other relevant factors, and long term process improvement.

The sword indicates the power to make a decision. The significance of the sword can serve as reminder how terrible and final a decision can be! While most system design evaluation decisions are not so devastating, there is still an important principle involved. Making a decision marks an end to a time of uncertainty, and allows forward movement. Indecision itself can be problematic, so this may be very desirable. However, this forward resolve can

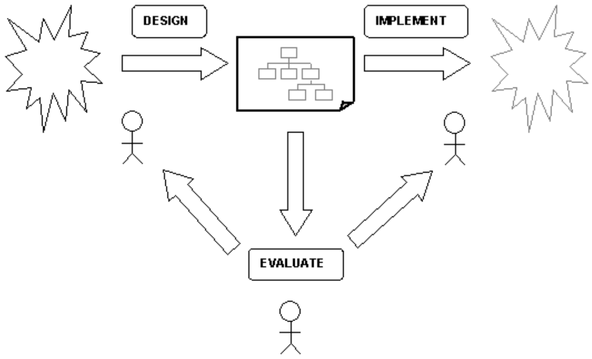

Figure 1: The role of design evaluation within the system development process, highlighting how evaluation can both inform later use of the design (summative evaluation) or iterative improvement of the design (formative evaluation).

only inspire confidence as long as it is based on a good process of evaluation.

Evaluation of Design

After our initial emphatic introduction about evaluation in general, we then move to a pragmatic consideration of what it means to evaluate *design*.

System design is a complex activity, and can involve all kinds of considerations. A simple model, however, is that the design process involves taking information from analysis, and produces a plan that facilitates implementation. This simple model can illustrate the place of design evaluation (see figure 1).

One aspect of design evaluation is that it can take place after the design process is complete, and the role of the evaluation produces is to produce advice for potential implementors. For example, such advice might recommend whether it is worth using a basis for implementation, and perhaps might even recommend a reasonable price for the design. This is close to the ideal of evaluation discussed above, where the evaluation leads to a decision. In education theory, this is known as "summative" evaluation. The idea is that it comes fully after the process, and determines a result. This kind of evaluation is familiar from school as the letter grade following a exam, or from sports as the kind of medal awarded at the Olympics.

Another aspect of design evaluation is that it can take place during the design process, working with a partial design result, or even by observing the design process itself. The role of the evaluation process here is to produce advice for the designers themselves. For example, such advice might comment on the strengths and weaknesses of the design, might suggest changes, and might even ask questions and engage in dialogue. This is still evaluation, but working in a different way to the ideal discussed above. In education theory, this is known as "formative" evaluation. The idea is that it is done in the midst of the design process, and helps improve the eventual design. This kind of evaluation is also familiar from school as the helpful comments made by teachers when students show exercise work, or from sports as the kind of advice given by coaches as the observe athletes in training.

These two views of evaluation, summative and formative, are both important. However, in system design work, it is unfortunately easy and common to focus too much on summative and not enough on formative evaluation. System design is difficult and costly, and more formative evaluation is one way to ensure the best result possible. Formative evaluation is a key step in using the iterative and incremental approach to system development.

Although we have distinguished designers, implementors, and evaluators, it is possible for people to play multiple roles. When the designers are also evaluators the impartiality is compromised, but especially in formative evaluation, simply distinguishing the evaluation activity from the design activity can lend a useful change in perspective. An alternative is to use peers as evaluators, and ultimately it may be worthwhile identifying experts on system design specifically as evaluators. There are also other roles that may be involved in the process, including other stakeholders in the development, or end users of the system.

HOW TO EVALUATE A DESIGN?

Approaches to Evaluation

The heart of a design evaluation process must be some technique for evaluating a design. The sad truth is that there are no really techniques for evaluating object-oriented designs already established. However, there are some related techniques that may be adapted for the purpose. As with much of system development, there is no guarantee of success, but we believe that some adapted techniques should prove very useful.

The workshops on educational evaluation of OO designs (Biddle et al. 1999) identified several evaluation strategies that could play a role in a design evaluation process:

- Metrics, such as class width and hierarchy depth
- Heuristics
- Use-Case Walkthroughs, especially with role-play
- Checks of Pattern and Pattern Language usage
- Modification Tasks where the design must accommodate reasonable change
- Specification Checks to see if the design does match analysis
- Design Critiques, similar to art criticism or reviews

There are two broad approaches that underlie the kinds of techniques that we might consider adapting: inspection and testing. These two approaches have been distinguished in the field of user interface evaluation (Nielsen 1992a, Nielsen and Mack 1994, Nielsen 1992b) and may serve as a model for design evaluation.

Inspection involves assessing the static design, looking for structures and detail elements that have critical implications about the design quality. Both heuristics and metrics fall within this category, and checks for patterns and critiques have similar properties.

Testing involves actually trying out the design. The ultimate testing involves implementation, of course, and that is usually too high a cost to bear to evaluate the design. We can simulate testing, however, by working with prototypes. Prototypes can involve lightweight implementation that merely simulate functionality. Even paper simulations can be helpful, and walkthroughs with object role-play might be regarded as a kind of very lightweight prototype. The main idea of such "testing" is to actually track through the design, as this turns out to be a good way to help think about the design implications. Following the implications of modification tasks would also be a kind of testing.

Desirable Characteristics

While we can talk of the "quality" of a design, this is not a simple concept: there are many desirable characteristics of a design. One important distinction concerns the time-frame under consideration.

There are a set of desirable characteristics of a design that concern the present domain situation. We should know a lot about this situation, as the design should have been created with this situation in mind, and the implementation will definitely be expected to work within it. The primary desirable characteristics are functionality, and usability. These are both complex characteristics in themselves, as for example functionality would include functional coverage and correctness, and usability might include learnability and user error handling.

The best approach to evaluating a design for these kind of current-situation characteristics is testing. Testing is possible because we should know the actual situation in some detail, and are thus able to drive test cases. Testing is also desirable because we really do need to know in detail how well the design copes with the situation.

In fact, all this is so important that our design techniques themselves are driven by this kind of testing. By beginning with use cases, and employing the use cases to direct the design, we are in essence using testing to create the design in the first place. Techniques such as use case and object role-plays use prototyping to enable the testing.

However, it still makes sense to check all this in an evaluation process. At the detail level, designers can sometimes make assumptions that a careful walkthrough can identify. At a larger level, a critical issue is to determine whether all use cases are covered by the design, as sometimes less important use cases can be forgotten by designers. Design evaluation can also be a time to introduce users, and end users can become involved with usability prototypes, and stakeholders can become involved in checking use case coverage.

Our recommendation therefore for evaluation of a design with respect to the current requirements is by forms of lightweight testing using prototypes, walkthroughs, and role-play. These now feature strongly and successfully in our advice and we discuss this in detail elsewhere, considering both techniques for review in the transition from requirements to design (Biddle, Noble and Tempero 2002a), and also in improving the CRC techniques for detail design (Biddle, Noble and Tempero 2002b).

Other desirable characteristics that relate to immediate deployment are those that concern the implementation process. For example, this would include the cost of implementation, and the time required. These are best evaluated by a static analysis of the design, considering what is involved in the implementation of each object or class. This is essentially the idea behind strategies for system estimation, for example (W.Boehm, Horowitz, Madachy, Reifer, Clark, Steece, Brown, Chulani and Abts 2000).

A related concern is whether the design involves reuse where possible. To evaluate this requires knowledge of the domain, knowledge of related systems, and also of the implementation system. It can be worth careful analysis looking for similar classes or objects elsewhere, and consideration whether they can be used or adapted to the current system. Where this is possible, it has the potential for many advantages. For example, this can lead to cheaper and faster implementation because the components already exist, more reliability because they have already been further developed, and easier maintenance because of the wider base. However, care is needed in making decisions about reuse, both in assessment of the quality of components, and in the implications and costs involved in adapting them for use. These general issues are well covered in the software reuse literature: for example see (Tracz 1995), (Jacobson, Griss and Jonsson 1997) or (McClure 1997).

As well as desirable characteristics that relate to immediate deployment, there are many others that relate to future situations. In particular, the future may bring changes in requirements, and a good design will allow the system to be changed to meet those requirements. These desirable characteristics for future possibilities typically stress ability to cope with change, for example: scalability, flexibility, adaptability, flexibility, modifiability.

There are clear difficulties in assessing how well a design will cope with change. One issue is that because the future is unpredictable, there are a large number of possibilities that might occur. Another is that the exact details of the new situation cannot be known. For both of these reasons, neither prototyping nor (direct) testing is a particularly useful technique for evaluation. This is a real challenge for any evaluation process.

Heuristic Evaluation of Design

To address evaluation of a design to cope with change, we have been suggesting approaches based on inspection: in particular, on use of heuristics. This has been exploratory work that, in the absence of an established process, we hoped might help in this important aspect of design evaluation.

Heuristics are guidelines, or rules of thumb. These come typically from experienced experts, and provide an easy way of highlighting design elements that are reasonable, or suggesting design structures are problematic. Many heuristics have been identified by experts, addressing many aspects of design. However, heuristics cannot be blindly applied, and in some cases there are heuristics than seem to offer conflicting advice.

Note that use of heuristics is also involved in design techniques themselves. For example, both the CRC technique (Beck and Cunningham 1989) and the technique known as "responsibility driven design" (Wirfs-Brock and Wilkerson 1989, Wirfs-Brock, Wilkerson and Wiener 1990) involves stress on "responsibility". This really is a kind of heuristic, suggesting that each object should have a reasonable set of responsibilities that it is helpful to fulfill, and that it is capable of fulfilling. Related heuristics are suggested in the technique by the stress on distributing responsibility between objects in a design. We discuss these issues in more detail elsewhere (Biddle et al. 2002a).

We have two reasons for suggesting that heuristics can play a specific and significant role in the evaluation of object-oriented designs. The first reason is that there are many heuristics documented in advice on "good" object-oriented design, including advice on design to cope with future change. Many of these heuristics have been collected, categorised, and discussed by Riel (Riel 1996). We hoped that such already existing collections of heuristics might be harnessed in developing an evaluation process. The second reason is that there is already a process for using heuristics in the evaluation of user interface designs that has been well established and explored by Nielsen (Nielsen 1992a, Nielsen and Mack 1994). Our idea was to take sets of heuristics for OO design as documented by Riel, and use them in place of user interface heuristics in the process described by Nielsen.

Nielsen's "heuristic evaluation" process for user interface evaluation does not and cannot deliver perfect results. However, it has been found to be an advance on purely ad-hoc and introspective techniques, and it also has the advantage of being relatively quick and inexpensive. Our familiarity with this process gave us some confidence that it might have similar advantages when applied to OO design evaluation.

Put simply, the heuristic evaluation process is to inspect the design with a set of heuristics specifically in mind. The set of heuristics should be a manageable number, and relevant to the desirable characteristic for which the design is being evaluated. This process is carried out by several inspectors who work separately in parallel to determine their findings about whether the design is in accordance with the heuristics, or whether there are problems. The inspectors then meet to discuss their findings and then together determine an evaluation of the design identifying problems.

While introducing design and the concepts of evaluation in our work with industry, we have also explored this technique for evaluation. Because the general structure of a design can determine how well the design can cope with change, we have been especially interested in evaluating the *topology* of an OO design. For the heuristics, we worked with a subset adapted from those identified by Riel as relating to system topology:

- Distribute system intelligence.
- Do not create god classes in your system.
- Beware of too many accessor methods.
- Beware of too much non-communicating behaviour.
- Model the real world whenever possible.
- Eliminate classes that are outside the system.

We sometimes used a longer list, consisting of other related heuristics adapted from those identified by Riel:

- A class should not be dependent on its users.
- A class should capture one and only one key abstraction.
- Keep related data and behaviour in one place.
- A class must know what it contains, but it should never know who contains it.
- Inheritance should be used only to model a specialization hierarchy.
- Subclasses must have knowledge of their superclass, but superclasses should not know anything about their subclasses.
- Factor the commonality of data, behaviour, and/or interface as high as possible in the inheritance hierarchy.
- When given a choice in an object-oriented design between a containment relationship and an association relationship, choose the containment relationship.

We first presented the heuristics and discussed how they related to design quality. We then arranged for people to work in groups doing heuristic evaluation of a number of small designs, sometimes working with designs presented by us, sometimes by themselves, and sometimes by other groups. The groups worked through the evaluations, and then we met to discuss together findings about the design.

The results of this exploration were mixed. On the positive side, most people understood at least some of the heuristics, saw how they related to design quality, and gained some confidence in identifying them in an actual design. Moreover, the process seemed to add value, because in parallel evaluation different people did identify different issues. These issues could then be discussed and prioritised in a larger group setting and led to a better coverage of issues, as well as serving as a knowledge sharing exercise.

On the negative side, many people did not really engage with all of heuristics well enough, either because they did not understand them, or did not understand their relevance. More familiarity with the heuristics was clearly necessary. It was striking that many people engaged with the heuristic about avoiding “god” classes. We pay special attention to this in presentation because while wanting to use the terminology used by Riel, we are careful explain the term is not meant in any religious way. Whatever the reason, people found it easy to evaluate with that heuristic in mind; “model the real world” worked similarly.

We speculate that, at least for learners, heuristics should be very blunt and engaging, rather than the more dispassionate consideration of OO relationships more typical of the second list above. On the other hand, people with more confidence in understanding those relationships found the more dispassionate language fairly straightforward to apply.

As a learning exercise and a vehicle for highlighting evaluation, we regard our exploration of heuristic evaluation a success. As an evaluation process, we feel more work is necessary. We believe that the next steps should be a careful construction of sets of heuristics related to particular desirable characteristics of design and also to related to the background of people who are to do the inspections. With those developed, validation exercises of the process should then be undertaken.

OTHER TECHNIQUES

Metrics

Another inspection technique we discuss is the use of design metrics. Metrics are quantitative measures taken from a design, typically counting certain kinds of features. For example, a simple metric relating to system size is the number of classes, and a simple metric for the complexity of a class is the number of messages to which it responds. Like heuristics, metrics typically come from the experience of experts, and there is no definitive set of standard metrics. We introduce the well-known set of OO design heuristics identified and studied by Chidamber and Kermerer (Chidamber and Kemerer 1991).

We have not conducted any exploration of the use of metrics as an evaluation process. We do point out, however, that metrics can be especially useful if used to compare like with like (Henderson-Sellers 1995): for example this can be used to quickly highlight aspects of a design that are different and potentially interesting to evaluate in detail. Unlike heuristics, metrics themselves do not offer any advice, but do provide a dispassionate input that often relates to advice embodied in related heuristics. An observation made by many people, both from industry and from our students at the university, is that the underlying heuristics really need to be understood before use of the metrics is sensible practice.

Potential Change Cases

In an object oriented system, there is an important relationship between the system design and its ability to adapt to future change. The system design consists of a set of objects and collaborations between them. The objects are encapsulated, so the collaborations only involve the object interfaces, rather than the object implementations. This means that object implementations can be changed without disturbing the overall design, and that new objects can be introduced as long as they conform to exisiting object interfaces. A future change will be easy to accommodate if it can be done in these ways. This means that to make a design future-proof, objects should be identified to best allow change to be accommodated by changing object implementation, or by introducing new objects with the same interface as exisiting objects. Most importantly, this means the objects in the design should align with changeable concepts in the domain. This principle underlies many heuristics.

Another simple approach to evaluating future adaptability is to consider specific possible changes that might arise: the potential “change cases”. Change cases were introduced by Ecklund at al. (Ecklund, Delcambre and Freiling 1996) to describe actual changes to a system through modification. In evaluation, we are just imagining what changes *may* be required. This involves predicting the future, and so can never be precise. However, knowledge of

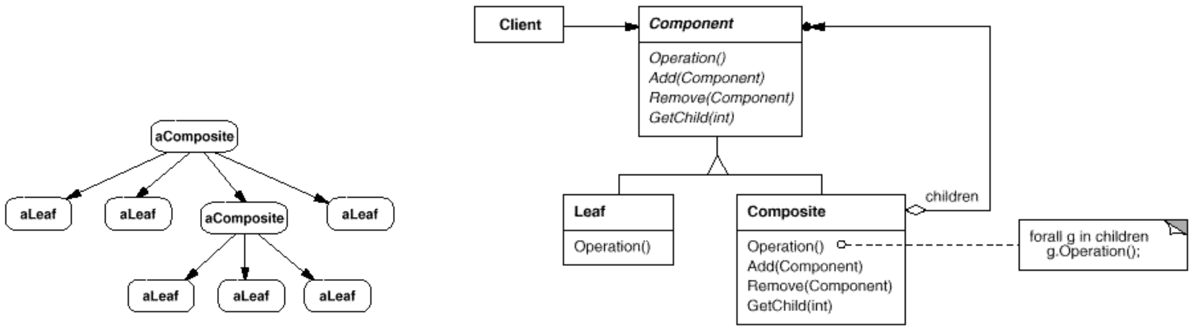

Figure 2: At the left is shown the kind of common structural situation that involves recursive collections; at the right is shown the composite pattern that best models these collections. (From Gamma et al.)

the domain can often lead to reasonable ideas about what is likely to happen in the future. Such "potential change cases" can be identified by focusing on the domain, much in the same way as use cases. After initial identification, change cases should then be prioritised, considering both which are most likely, and which lead to more serious changes.

We have conducted exercises in doing this. The technique is to identify change cases, then to consider each case, and determine how easily the change could be accommodated in the existing design. The approach to seems to be quite accessible. Most people who understand a domain can easily imagine how requirements may change in the future. It is more difficult to see how the change would affect a design, but we have been able to suggest some possibilities to look for.

One easy kind of change involves only behaviour or memory of one class or object. Also easy is a new object that is an alternative to one already in the design, and can lead to an inheritance hierarchy. More difficult is a change to behaviour or memory that spans many objects or classes. This can often suggest there should be another object or class, so that change can be isolated there.

It can also be useful to consider future situations that are based less entirely on the domain, and that do involve the existing system design itself. In particular, it can be useful to consider future possibilities involving reusability. To evaluate reusability, the idea is to consider whether objects in the current system might possibly be reused in other possible systems. Where there is a possibility, it can be worth considering whether the design might be changed to improve the likelihood of reuse. This can involve making objects more general, for example by adding functionality, or by allowing customisation. This can itself be expensive, and care is needed in determining whether the potential for reuse is worth the risk of costly further development.

Our exploration of change case evaluation was brief but encouraging. We particularly like the way it begins in a way that is accessible to anyone who understands the domain, and so for example is open to business analysts and non-technical stakeholders. The later analysis of how a design copes with change cases is less accessible, but at least the key questions will have been asked, and implications can then be called for and assessed. We believe this technique is worthy of more study.

Patterns

Another topic that forms part of our recent introduction to OO is that of *design patterns*. Design patterns were introduced in 1994 (Gamma, Helm, Johnson and Vlissides 1994) and are becoming well known by OO programmers. There have been efforts to introduce design patterns to a broader audience in information system modeling (O'Callaghan 1999, Taylor 2002), but we fear they are still not enough well known in the analysis and modeling context.

Design patterns document particular OO design structures that solve particular modeling problems within certain contexts. Some of these problems are very common, yet good solutions are sometimes surprisingly difficult to discover by oneself. Design patterns document these good solutions. We therefore feel that OO design patterns also suggest an inspection approach to design evaluation.

We have explored this by introducing some simple patterns and explaining the idea. We then offer a design modeling task with a well known design pattern solution. The task we usually set involves a collection which consists of several items of different kinds, one of which involves another collection, and so on recursively. This is a common model, and occurs frequently in domains such as document structure and organisational hierarchies.

For example, we ask people to model a package tour booking that may include air fares, accommodations, and possibly sub-packages. For even people with some familiarity with OO, getting this model correct is difficult and time-consuming. We present and explain the design pattern that provides a solution, the *composite* pattern, as shown in figure 2. People then learn quickly, and our experience is that they are able to quickly apply this solution, and to see when and how it should be applied.

We have explored the mechanisms at work here in detail described elsewhere (Noble, Biddle and Tempero 2002, Noble and Biddle 2002), and suggest that design patterns are *signs* and play a semiotic role in modeling. Our experience with industry has focused on teaching the value of design patterns in modeling. However, we feel the results show that there is clear potential for a link to evaluation. We suggest that a design evaluation technique could be based on inspection of a domain, and checks to see if the well known designs patterns were used as appropriate.

CONCLUSIONS

In this paper we have addressed the topic of evaluation of object-oriented designs. Our findings are based on our work introducing object-oriented analysis and design to industry groups. We have found that there is a need for design evaluation because actual designs now form the interface between business analysts and system developers, yet there are few techniques and processes available.

We have discussed in this paper how we have motivated the importance of design evaluation, and how we have identified and explained the issues that arise in design evaluation: formative and summative evaluation, and approaches involving both inspection and testing. One critical idea is that the quality of a design must always mean quality for some purpose. To really evaluate quality, one should always keep the purpose in mind.

We have also outlined our exploration of several techniques, including heuristic evaluation, change case analysis, and design pattern inspection. We found all to be successful as techniques for teaching and learning, and we believe that with further study and development these could become successful evaluation processes. We also feel the principles of design evaluation we have identified, together with experience of evaluation practices in related areas of information systems development, can lead to significant improvements in system development management.

References

Beck, K. and Cunningham, W.: 1989, A laboratory for teaching object-oriented thinking, *Proc. of OOPSLA-89: ACM Conference on Object-Oriented Programming Systems Languages and Applications*, pp. 1–6.

Biddle, R., Mercer, R. and Wallingford, E.: 1999, Report on the oopsla workshop on evaluating OO design, *Addendum to the Proceedings of OOPSLA'98*.

Biddle, R., Noble, J. and Tempero, E.: 2001, Techniques for active learning of Object-Oriented Development, *ACM Oopsla Educators Symposium*.

Biddle, R., Noble, J. and Tempero, E.: 2002a, Essential use cases and responsibility in object-oriented development, *in* M. Oudshoorn (ed.), *Proceedings of the Australasian Computer Science Conference (ACSC2002)*, Melbourne, Australia.

Biddle, R., Noble, J. and Tempero, E.: 2002b, Reflections on CRC cards for OO design, *in* J. Noble and J. Potter (eds), *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS-Pacific)*, Sydney, Australia.

Chidamber, S. R. and Kemerer, C. F.: 1991, Towards a metrics suite for object oriented design, *Proceedings of ACM Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA91)*.

Ecklund, E. F., Delcambre, L. M. L. and Freiling, M. J.: 1996, Change cases: use cases that identify future requirements, *Proc. of OOPSLA-96: ACM Conference on Object-Oriented Programming Systems Languages and Applications*.

Gamma, E., Helm, R., Johnson, R. E. and Vlissides, J.: 1994, *Design Patterns*, Addison-Wesley.

Henderson-Sellers, B.: 1995, *Object-Oriented Metrics: Measures of Complexity*, Prentice Hall.

Jacobson, I., Griss, M. and Jonsson, P.: 1997, *Software Reuse: Architecture, Process and Organization for Business Success*, Addison-Wesley.

Leffingwell, D., Widrig, D. and Yourdon, E.: 1999, *Managing Software Requirements: A Unified Approach*, Addison-Wesley.

McClure, C.: 1997, *Software Reuse Techniques : Adding Reuse to the System Development Process*, Prentice Hall.

Nielsen, J.: 1992a, Finding usability problems through heuristic evaluation, *Proceedings ACM CHI'92 Conference (Monterey, CA, May 3-7), 373-380*.

Nielsen, J.: 1992b, *Usability Engineering*, Academic Press, New York.

Nielsen, J. and Mack, R. L. (eds): 1994, *Usability Inspection Methods*, John Wiley & Sons.

Noble, J. and Biddle, R.: 2002, Patterns as signs, *in* B. Magnusson (ed.), *Proceedings of the European Conference on Object-Oriented Programming(ECOOP 2002)*, Springer, Malaga, Spain.

Noble, J., Biddle, R. and Tempero, E.: 2002, Metaphor and metonomy in object-oriented design patterns, *in* M. Oudshoorn (ed.), *Proceedings of the Australasian Computer Science Conference (ACSC2002)*, Australian Computer Society, Conferences in Research and Practice in Information Technology, Melbourne, Australia.

O'Callaghan, A. J.: 1999, Migrating large scale legacy systems to component-based and object technology: The evolution of a pattern language, *Communications of the Assocation for Information Systems* **2**(3).

Riel, A.: 1996, *Object-Oriented Design Heuristics*, Addison Wesley.

Taylor, P. R.: 2002, Patterns as software design canon, *in* D. Cecez-Kecmanovic, B. Lo and G. Pervan (eds), *Proceedings of the Australasian Conference on Information Systems (ACIS2001)*, Coff's Harbor, NSW, Australia.

Tracz, W.: 1995, *Confessions of a Used Program Salesman: Institutionalizing Software Reuse*, Addison-Wesley.

W.Boehm, B., Horowitz, E., Madachy, R., Reifer, D., Clark, B. K., Steece, B., Brown, A. W., Chulani, S. and Abts, C.: 2000, *Software Cost Estimation with Cocomo II*, Prentice Hall.

Weinberg, G. M. and Freedman, D. P.: 1990, *Handbook of Walkthroughs, Inspections, and Technical Reviews*, third edn, Dorset House Publishing.

Wirfs-Brock, R. and Wilkerson, B.: 1989, Object-oriented design: A responsibility-driven approach, *in* N. Meyrowitz (ed.), *Proc. of OOPSLA-89: ACM Conference on Object-Oriented Programming Systems Languages and Applications*, pp. 71–75.

Wirfs-Brock, R., Wilkerson, B. and Wiener, L.: 1990, *Designing Object Oriented Software*, Prentice Hall.

COPYRIGHT