

# Testing a Requirements Pattern Language through Reverse Engineering

Peter Merrick and Patrick Barrow  
School of Computing Science, University of East Anglia,  
Norwich, Norfolk, U.K. NR4 7TJ

**Abstract.** This paper looks at a case study to reverse engineer an IT system that supports the Health and Safety Executive in making planning recommendations with respect to hazardous installations. It compares a Use Case model created from a requirements specification with a Use Case model derived from an inspection of the built system. The objective is to discover how accurately the final system could be predicted using a series of requirements patterns. Through the application of requirements patterns, it was possible to predict the functionality delivered to a high degree, which suggests this is a useful contribution to the prediction of functionality and thereby to system sizing. This work is part of a bigger project for the Health and Safety Executive investigating improvements to system sizing/effort estimation and its impact on the management of complexity.

## *Introduction*

The purpose of this case study is to contrast a requirements representation produced with the aid of a pattern language with that produced after an inspection of the built application as evidence to suggest the usefulness of the language in predicting the functionality of the finished product. The requirements language is designed to be independent of vertical business domains (e.g. banking, travel, retail) although it is constrained to predicting the shape of transactionally-oriented database driven applications, sometimes characterized as solutions that fit the ‘workpiece’ frame (Jackson 2001). The Domain Theory (Sutcliffe 2002) proposes that all software engineering problems can be described by a small set of fundamental abstractions. Sutcliffe suggests that one of the ways an application may be built is through specification in a high level requirements language leading to program generation. The ability to predict the final shape of an application from its requirements expression would be valuable in expressing system size, as an input to effort estimates, in managing change and reporting progress. If it can be shown that such a language exists it could suggest the existence of other languages, such as those necessary for the representation of real time or embedded applications. For instance, the ‘connection problem frame’ characterizes applications requiring the detection of objects for management within a model of the outside environment (Jackson 2001).

The representation of requirements is often ambiguous. Where natural language is employed in requirements specifications misunderstanding can lead to problems and project failure (Standish 1994; Taylor 2001; staff 2002). Requirements representation using formal methods (Spivey 1989) may have clear meaning to engineers but at the expense of understandability to customers (Sutcliffe 2002; Khazaei and Roast 2003). The informal diagrams of Use Cases, within the Unified Modeling Language (UML), are less ambiguous than natural language, with the added benefit of retaining their understandability to customers. One of the perceived benefits of embracing requirements expression with Use Cases at a consistent level of abstraction is the ability to transform the problem in a manner that suggests a solution.

Use Case modelling is a popular technique for representing requirements, normally employed

in the analysis phase of the software engineering lifecycle. The UML (Fowler and Scott 1999) specifies Use Case modelling as the basis of a development approach where Use Cases act as the input to design, as the basis of verification, validation and testing. Use Cases were first introduced by Jacobson (Jacobson, Jonsson et al. 1992) and have since become part of the UML standard. A Use Case is typically made up of two parts, a graphical element and a textual element. Modeling in this paper is constrained to the graphical element, a component of a complete Use Case which Kulak (Kulak and Guiney 2000) refers to as a 'façade'. The models presented in this paper make use of the aggregation notation rather than either the <<include>> or <<extend>> stereotype to avoid making a decision at a premature stage as to whether the functionality is mandatory or optional. A complete description of Use Case Modeling is beyond the scope of this paper.

This case study was undertaken as part of a wider enquiry into possible improvements to the effort estimation of projects based on functional complexity expressed in Use Case notation. As part of the wider work, it was necessary to calculate an initial measure of efficiency that described the software supplier organization's effort in building an historic project where expended time and system shape were already known. One model was constructed through the application of a requirements pattern language (RPL), in isolation from the other, which was constructed through an inspection of the built system that took the original business requirements document as its input. The system under investigation manages planning advice for developments near hazardous installations, known as PADHI, for the Health and Safety Executive (HSE) in the U.K. After the initial model was made using only patterns (control model), a second modeling iteration of the same application was undertaken (evaluation model) based on an inspection of the application itself. These two models were then compared.

The application inspection, for the purpose of constructing the evaluation model, was undertaken by the authors and the project manager responsible for building the application. The parties had not previously met, although the project manager had been the contact during the research (via email) responsible for the supply of the original requirements specification document. A decision was taken not to seek any clarification of the original specification. This was done to minimize the introduction of bias that might be introduced with the benefit of hindsight.

The task of creating the evaluation Use Cases began with an informal demonstration of the application undertaken by the project manager. This was the first time the authors had seen the application. During the demonstration notes were kept. When the demonstration was complete, the control model was shown to the project manager. To discover the Use Cases that were not predicted by the RPL, the authors asked the project manager to again demonstrate aspects of the system's functionality that had been noted but which did not appear in the control model.

After the demonstration the evaluation Use Case model was constructed and sent to the project manager for verification. He stated in his report to the sponsor of this research that "there is a great deal of merit in adopting the approach [suggested]. I am sure that if we had made use of such diagrams during the requirements phase ... we would have stood a much better chance of spotting issues [that later arose] between the business and [the supplier]".

The models are compared in this paper to test the efficacy of the pattern language in creating models that accurately predict the form of the final delivered system. It is argued that the degree to which the final model corresponds to the initial model will provide evidence as to the usefulness of the pattern language. The principal question under investigation in this paper is whether an accurate Use Case representation can be constructed from a loosely defined customer

requirements statement. The choice of application on which to perform the reverse modelling exercise was chosen for the authors on the basis that it represented a project the customer considered to have gone well. To this extent the authors were not free to choose the examples that might best fit the research, but rather must fit the research to the demands of the industrial collaborator.

As far as the authors are aware the technique of generating Use Case models from requirements patterns is novel. The remainder of this paper is divided into a description of the project being investigated followed by an overview and application of the pattern language, including all resulting diagrams produced before and after application inspection. Results follow, along with a section on lessons learned, threats to validity and suggestions for further work.

## *A Synopsis of the Customer's Requirements*

Planning authorities in the U.K. have a statutory duty to consult on certain developments that fall within a specified proximity to hazardous installations and pipelines. It was intended that the system would support the Land Division of HSE in the processing of land use planning applications. Every application will ultimately receive either an 'allow' or a 'refuse' decision. Rarely, an application results in a 'consult' decision where the application is considered by a specialist risk assessor. Some consultations are not relevant and are either returned or are forwarded to other departments who have specific responsibility. Every application results in some kind of letter being sent in response to a consultation, even if it is to inform the applicant that an application is not necessary or that it is being handled by a different department within the HSE.

Applications are made where a development is within the consultation distance of hazardous installations which are defined as being pipelines, quarries, explosive and nuclear sites. The basis upon which decisions are made is a set of rules that have been codified through many years of experience. A consultation's success will be based on a combination of the type of development being proposed (industrial, residential etc.), the characteristics of the hazard type and the proximity of the proposed development to the identified hazard.

Each consultation is treated as a new consultation; there is no requirement to refer to past applications except for the purpose of reporting. Some applications cannot proceed because there is insufficient information for an assessment to be made. In this situation the applicant is informed by letter including details of the information that is missing. If the application is submitted at a later date with the additional required information it is treated as a new application.

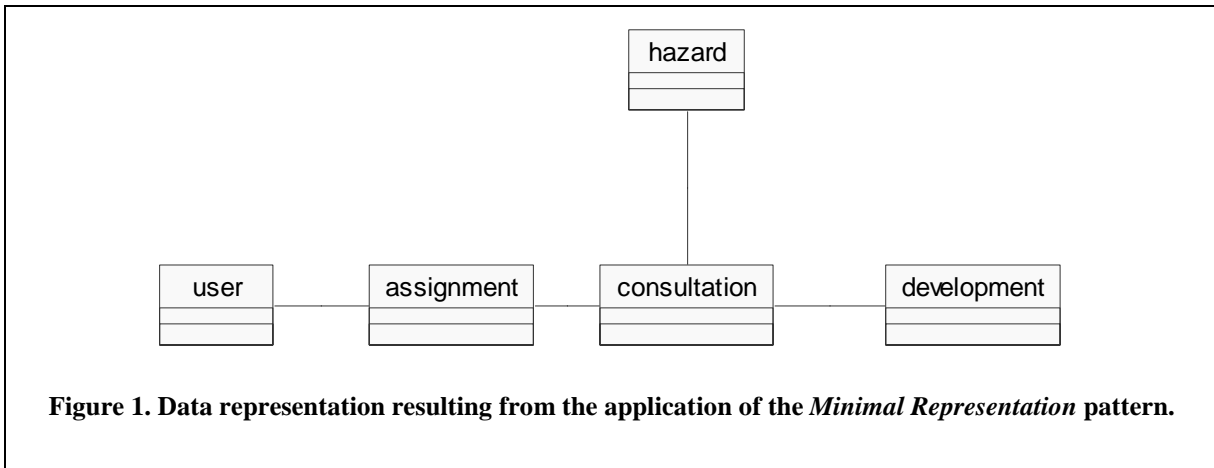
An application will record the planning authority, the authority's reference, the address of the proposed development, a description of the development, the date the application is received, and will be assigned a unique identifier by the HSE.

## *Applying the Requirements Pattern Language*

The pattern language being tested in this case study is comprised of the following patterns: **Minimal Representation**, **The Usual Suspects**, **NewSearchModify (NSM)**, **Informed Management**, **Role Based Assignment**, and **Coherent Whole**. The purpose of this pattern language is to rapidly produce a Use Case model from a statement of requirements. Use Cases created through the application of the pattern language are constrained to represent an abstraction of the problem rather than a statement of a proposed solution (Kulak and Guiney 2000;

Cockburn 2001; Merrick and Barrow 2003). It is also necessary for the accurate application of the language that the complexities of Use Case abstraction are understood. Use Cases can be represented according to different axis of both detail and goal (Kulak and Guiney 2000; Cockburn 2001). The constituent patterns of the RPL are not presented here in their entirety due to restrictions of space however an overview is included to allow the reader to understand the purpose of each component pattern.

An important prerequisite for success in the application of the RPL is that the original specification should include a description that includes the nouns that will form the basis of an analysis that will lead to the identification of entities as described by Abbott (Abbott 1983). The first pattern is **Minimal Representation** which requires the construction of a logical minimal data model sufficient to support the business. The process of applying the pattern language relies on the expertise of the practitioner in being able to construct such a minimal data model. To accomplish this task, noun analysis was applied, with the intent of identifying candidate nouns as entities (potential database tables). Care was taken to select only nouns that would represent entities as opposed to nouns that represented attributes of entities (Connolly, Strachan et al. 1995).



**Figure 1. Data representation resulting from the application of the *Minimal Representation* pattern.**

Once the Minimal Representation pattern has resulted in the generation of a set of candidate data entities, the NSM pattern can be applied. This pattern generates three candidate Use Cases for each data entity; New <<entity>>, Search <<entity>> and Modify <<entity>>. Other authors have suggested that the identification of functionality based on data entities is likely to produce a valid candidate set (Armour and Miller 2001; Jackson 2001). This kind of functionality is commonly termed Create, Read, Update, Delete (CRUD).

<b>Predicted entity</b>	<b>New from NSM</b>	<b>Search from NSM</b>	<b>Modify from NSM</b>
<b>consultation</b>	New consultation	Search consultation	Modify consultation
<b>development</b>	New development	Search development	Modify development
<b>assignment</b>	New assignment	Search assignment	Modify assignment
<b>user</b>	New user	Search user	Modify user
<b>hazard</b>	New hazard	Search hazard	Modify hazard

**Table 1. Application of the NSM pattern over the identified candidate tables generated from the prior application of the *Minimal Representation* pattern.**

The result of this pattern creates a candidate Use Case set; it is not necessary to include all Use Cases identified by NSM in the resulting models. The decision to include a Use Case is a matter of judgement that is informed by considerations such as the dynamic nature of a particular

entity (how often it changes) and the requirement to provide a user interface for the entity's management. For example, *hazards* do not change often and therefore their management was not included.

Once a candidate data model had been constructed, and the *NSM* pattern applied, the **Usual Suspects** can be employed to allow for quick identification of *actors* (Jacobson, Jonsson et al. 1992; Fowler and Scott 1999) that have a high likelihood of occurring in a transactionally-driven software application.

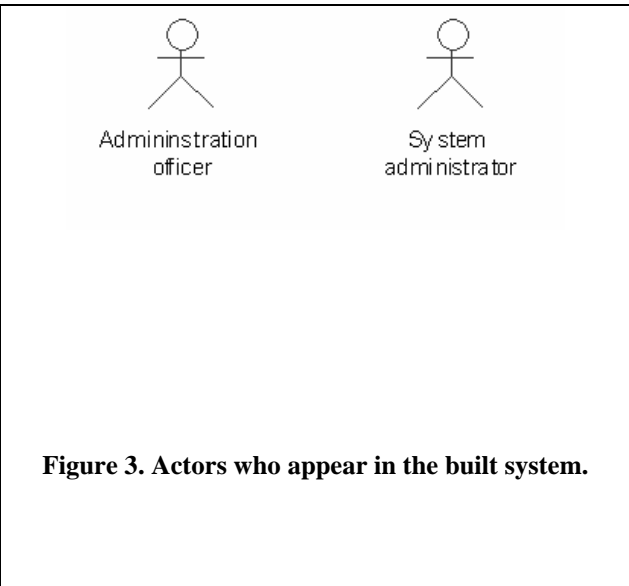
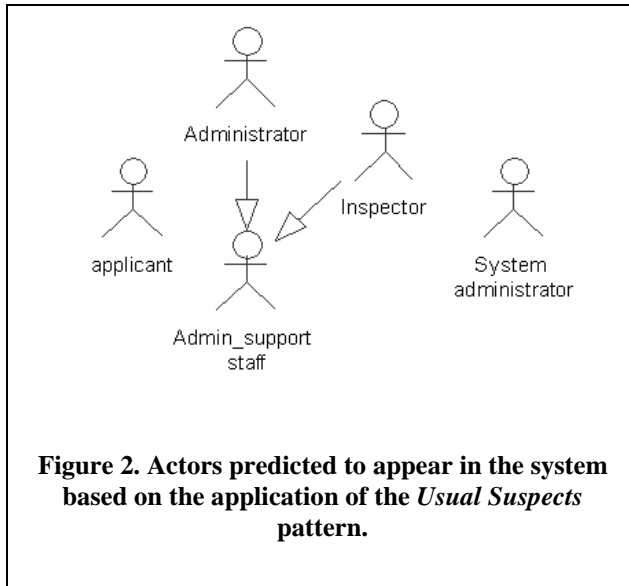


Figure 2 specifies many more actors than exist in the actual system. The two actors that do appear in Figure 3 were predicted by the pattern. The inclusion of so many actors in Figure 2 could be construed as over-specification.

After the application of the *Usual Suspects*, it is appropriate to apply the **Informed Management** pattern (illustrated in Figure 11 and 12) which generates reporting Use Cases often thought of as providing the functionality associated with a management information system (MIS). The pattern **Role Based Assignment** provides a way of grouping Use Cases and representing them in a manner that is logically coherent from the perspective of presentation. The final pattern, **Coherent Whole** provides a check to ensure that the model generated represents a logical whole whose sum provides recognisable value to its cast of users.

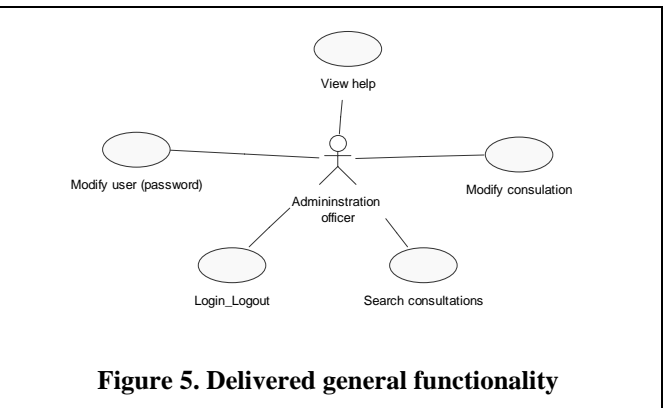
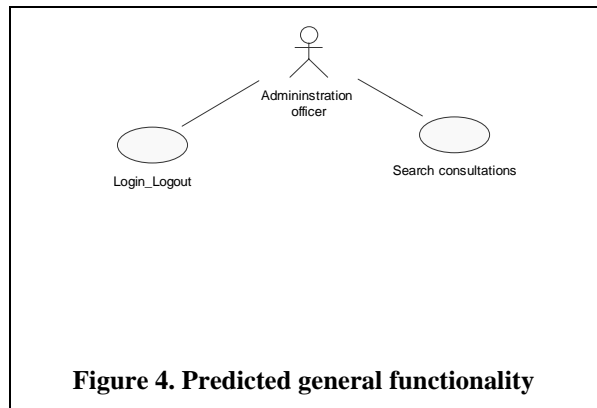
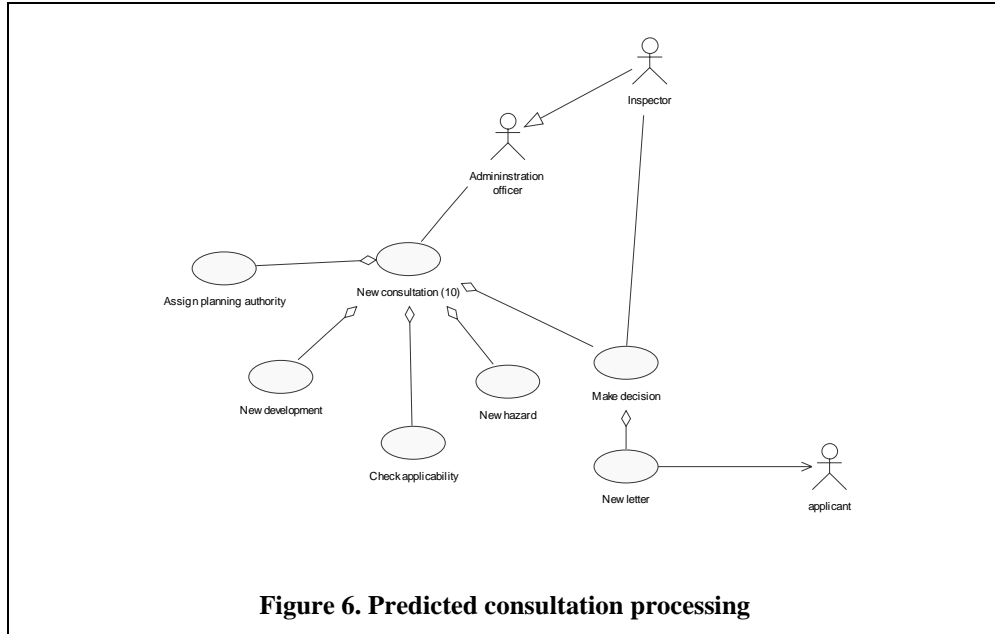


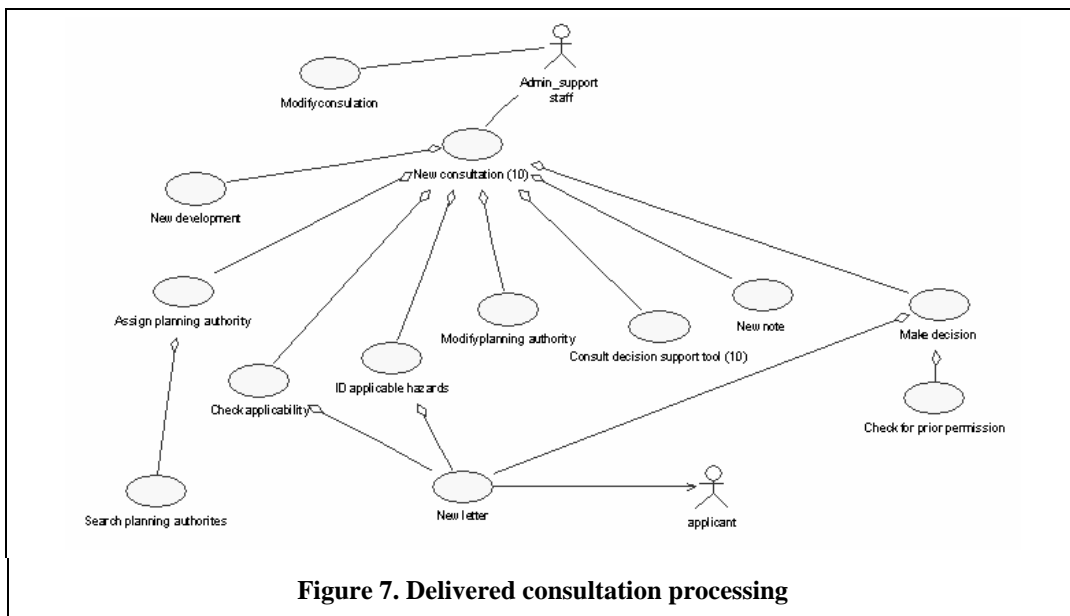
Figure 4 represents only the Use Cases *Login\_Logout* and *Search consultations*. Additionally, Figure 5 introduces *Modify user (password)*, *View help* and *Modify Consultation*.

Although *Modify user* was originally identified (table 1) by NSM it was assigned to a different diagram (Figure 8). The Use Case *View help* was not predicted. In the case of *Modify Consultation* it was specifically discounted, as described in the original requirements statement, although it was a member of the candidate set generated through the application of the NSM pattern. This is an example of requirements changing over the lifetime of the project.



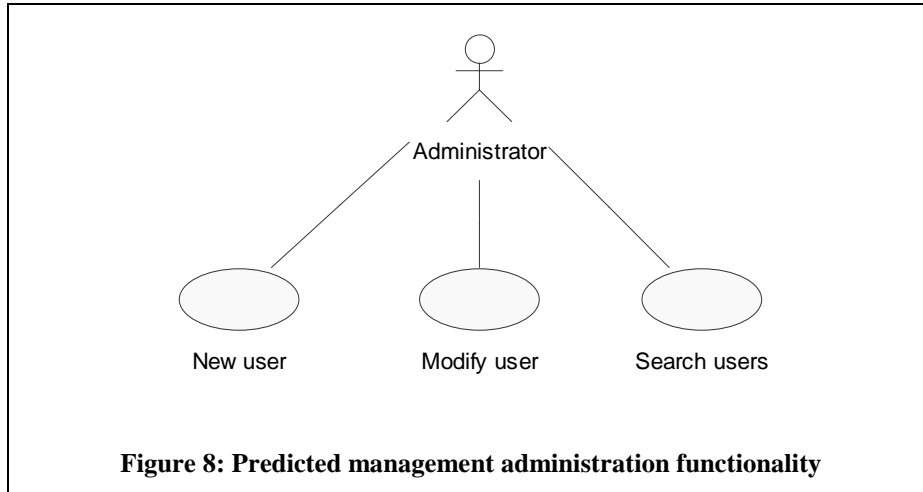
**Figure 6. Predicted consultation processing**

The predicted general workflow associated with the creation of a *New Consultation* is captured in this diagram (Figure 6). It includes the assignment of a planning authority, the definition of a new development, a check on the applicability of the application and identification of relevant hazards. Once this information is defined, a decision can be made as to whether permission should be granted. Figure 6 includes the *Inspector* actor, as the original business requirements stated some consultations would require a direct decision to be made because the rules captured by the system would prove insufficient in a minority of cases.



**Figure 7. Delivered consultation processing**

Whereas Figure 6 captures the predicted functionality associated with the creation and processing of a new consultation, Figure 7 captures the delivered functionality. The diagrams are similar. Among the differences is the ability in the final system to modify the details of a planning authority (for this consultation only). The deployed application includes a decision support tool which allows the user to gain online advice with respect to the likely result of a specific consultation prior to an official decision being generated. Additionally, consultations can have notes attached to them. Aggregates are not described as differences, such as the necessity to check for prior planning permission, as this is treated as a component functionality of *Make decision*.



In Figure 8, the Use Cases defined over the entity *user* by the *NSM* pattern are assigned to the *Administrator* actor.

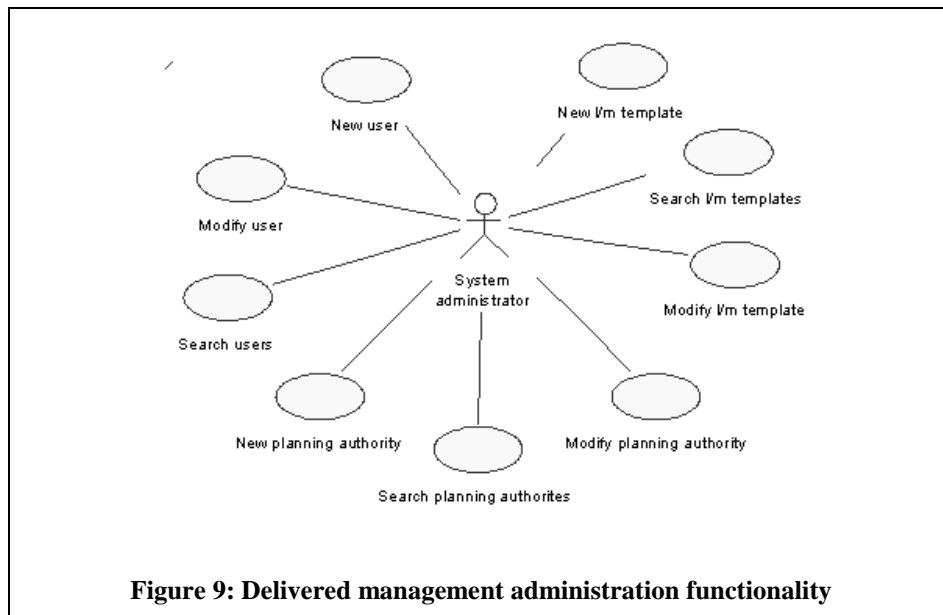


Figure 9 includes the *NSM* Use Cases defined with reference to the *user* entity introduced in Figure 8, plus an additional six Use Cases that follow the same pattern but which are defined with reference to entities not identified by the *Minimal Representation* pattern; namely *planning authority* and letter templates (*l/m templates* – letters and memos).

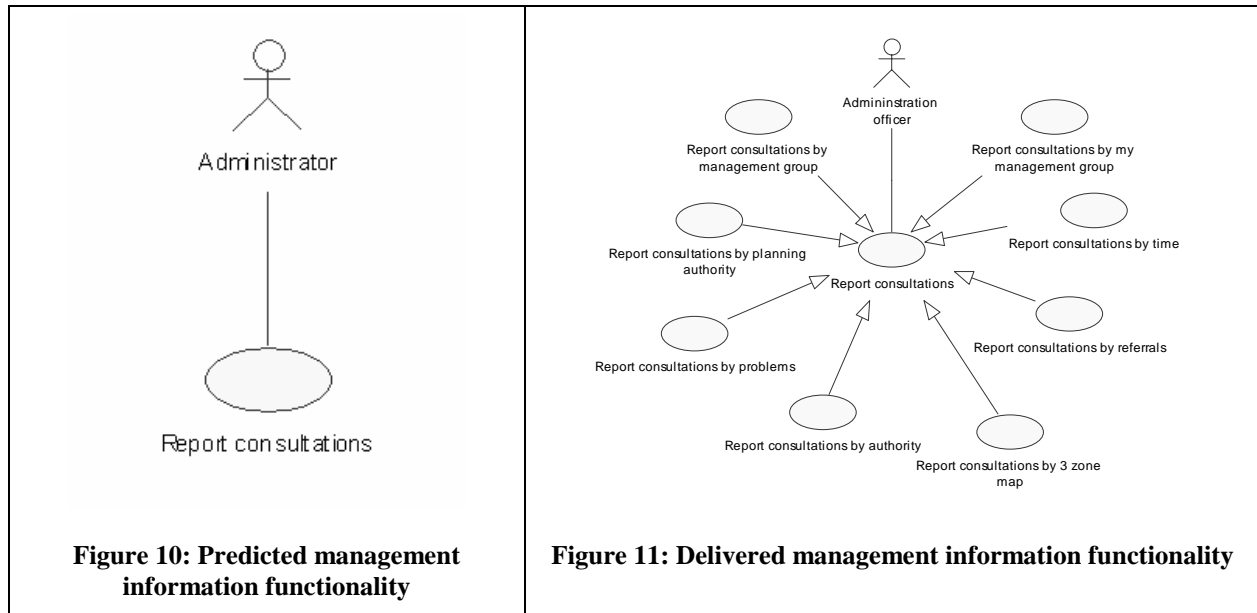


Figure 11 is a version of Figure 10 that includes considerably more detail. The Use Case *Report consultations* has been further elaborated through defined specializations that allow reporting based on specific attributes. Figure 11 employs generalization relationships making explicit that *Report consultations* is abstract, realized by eight concrete specializations.

## Results

The results of this case study are based on a comparison of Use Case models produced in isolation from one another designed to mimic the ‘before and after’ view of an application. The objective was to test how effective the pattern language (RPL) was in predicting the functionality of the final system. It is reasonable to assume that, as a project progresses more requirements are discovered and more detail will become available. On first inspection it is evident that the ‘after’ diagrams contain a larger number of Use Cases than those diagrams produced through the application of patterns. However, before any conclusion is drawn, it is necessary to define the basis of the comparison more rigorously.

Use Cases can be represented at different levels of abstracted goals; therefore it is necessary to agree on the levels being employed to enable the comparison of like with like. According to Cockburn (Cockburn 2001), Use Cases exist on a goal centred scale, ranging from ‘very high level’ to ‘too low’ [very high summary, summary, user-goal, sub-function, too low]. (The choice of *sub-function* as a classification might better be termed *sub-goal* (Merrick and Barrow 2003).) Most of the Use Cases defined in the first modelling iteration are at the level of *user-goal*. In the second modelling iteration more Use Cases have been introduced at both the *user-goal* and the *sub-goal* level. Where Use Cases at the level of sub-goal have been defined in the first modelling iteration they are included in the comparison with the second iteration. Where sub-goal Use Cases are introduced in the second iteration they are not included in the comparison. The justification for making comparisons in this manner is a recognition that it is not always possible to descend to the sub-goal level at a very early stage in modelling due to a lack of detail.

The Use Case models generated before the system was built are a good reflection of the final Use Case models as measured by their ability to predict functionality that would be implemented in the final system. Throughout the set of predictive models, 17 Use Cases were defined. All of

these Use Cases were present in the second set of Use Case models representing the built system. This suggests the patterns are capable of accurately predicting Use Cases that will be present in the final system. Discounting additional aggregate Use Cases introduced in the second modelling iteration, there were 26 Use Cases in the final system. By this measure, the patterns were able to predict approximately 65% of the total system functionality. This result is consistent with an earlier experiment conducted on a system built for the Learning and Skills Council in the U.K. (Merrick 2004 (accepted for publication)).

## *Lessons learned*

This experiment has brought to light a number of issues that are likely to affect the efficacy with which this approach to system definition can be applied. There are a number of areas in which the uninformed or over-zealous application of the patterns may lead to less accurate results. By its very nature, a pattern based approach to system definition will vary with the interpretation of the practitioner (Alexander, Ishikawa et al. 1976). Many of the potential pitfalls may be mitigated through consultation with the project owner over assumptions that are made in early modelling iterations. For instance, the *NSM* pattern will produce 3 Use Cases for every principle data entity identified by the *Minimal Representation* pattern. Although these Use Cases are candidates for inclusion in the model, they are not mandatory. Guidelines as to which Use Cases should be part of the model include an assessment of whether the entity is volatile and if an end-user interface is required for its manipulation. The circumstances of this experiment precluded checking the initial models with the customer as modeling took place after the fact; an artificial situation required by particular circumstances as previously described.

A comparison of Figure 10 with Figure 11 suggests it is desirable to develop the pattern to more accurately predict the number of specializations that may result from the original abstract Use Case *Report Consultations*. There are clear examples in this case study that illustrate how original requirements may change over the course of a project. For instance, originally it was specified that an Inspector was required to preside over certain difficult consultations that were beyond the ability of an automated system, which ultimately was not the case. In addition, although it is principally true that *consultations* are not modified (as originally specified); in the built application some minor modifications are possible (Figure 5).

## *Threats to validity*

The method is dependent on the initial construction of a minimal logical data representation which is, in turn, dependent on the skill of the practitioner. Also, the entire method has been applied by the authors, and it remains to be seen if other practitioners will find it so successful. The experiment could have been improved if a different party had undertaken one of the modeling iterations, an approach that will be tried in future.

The modeling depends on Use Cases being captured at the same level of abstraction. If a Use Case is decomposed to a level that is closer to implementation, and then included in a comparison, the results will be misleading. Another prerequisite to using this pattern language is that a *sufficient* requirements specification exists. This must state the intended purpose of the system, although how well requirements are captured, written and understood will vary from document to document. The quality of the original specification statement will have an impact on the quality of the results where candidate nouns must be present. Although the *Logical Whole* pattern exists to provide a check for missing entities, it does not provide a guarantee that

the logical model is complete.

## ***Directions for future work***

This is the second application of this approach applied with commercial partners. It has shown sufficient promise to sanction the application of further work to predict the size of a new system expressed in Use Cases and to then measure the results as part of the wider effort estimation improvement project being conducted.

A knowledge transfer is being undertaken between the authors and the IT company charged with building bespoke software on behalf of the HSE. The next part of this work is to use the results to calculate an initial unit of efficiency in the construction of a notional unit of function. To do this, a variation on Karner's Use Case Points Method (Karner 1993) will be employed following on from earlier algorithmic effort estimation approaches such as Function Point Analysis (Albrecht 1979) and COCOMO (Boehm 1981).

## ***Bibliography***

- Abbott, R. J., "Program design by informal English descriptions." *Communications of the ACM*, **26** (11) pp 882-894, 1983.
- Albrecht, A. J., "Measuring Application Development Productivity." *IBM Applications Development Symposium* (Monterey, CA, 1979), pp 83-92 IBM.
- Alexander, C., S. Ishikawa, S. M., M. Jacobson, I. Fiksdahl-King and S. Angel, *A Pattern Language - Towns, Buildings, Construction*. Oxford University Press, New York, N.Y., 1976.
- Armour, F. and G. Miller, *Advanced Use Case Modelling*. Addison Wesley Longman, 2001.
- Boehm, B., *Software Engineering Economics*. Prentice Hall, Upper Saddle River, N.J., 1981.
- Cockburn, A., *Writing Effective Use Cases*. Addison Wesley Longman, Upper Saddle River, N.J., 2001.
- Connolly, T., A. Strachan and C. Begg, *Database Systems - A Practical Approach to Design, Implementation and Management*. Addison Wesley Longman, Harlow, 1995.
- Fowler, M. and K. Scott, *UML Distilled - A Brief Guide to the Standard Object Modeling Language*. Addison Wesley Longman, Upper Saddle River, N.J., 1999.
- Jackson, M., *Problem frames: analysing and structuring software development problems*. Pearson Education, Harlow, 2001.
- Jacobson, I., P. Jonsson, M. Christerson and G. Overgaard, *Object-Oriented Software Engineering - A Use Case Driven Approach*. Addison Wesley Longman, Upper Saddle River, N.J., 1992.
- Karner, G., "Resource Estimation for Objectory Projects", Masters thesis, Linköping Institute of Technology, 1993.
- Khazaei, B. and C. Roast, "The Influence of Formal Representation on Solution Specification." *Requirements Engineering*, **8** (1) pp 69-77, 2003.
- Kulak, D. and E. Guiney, *Use Cases - Requirements in Context*. Addison Wesley Longman, Upper Saddle River, N.J., 2000.
- Merrick, P., "Testing the Predictive Ability of a Requirements Pattern Language." *Requirements Engineering*, 2004 (accepted for publication).

Merrick, P. and P. Barrow, "Towards a Requirements Formalism in Procurement." *8th Annual Conference of United Kingdom Academy of Information Systems* (Warwick, England, 2003), pp

Spivey, J. M., *The Z notation: a reference manual*. Prentice-Hall, Upper Saddle River, N.J, 1989.  
staff, "Why IT Projects Fail", Office of Government Commerce, London, 2002.

Standish, "The Chaos Report", Commercial Report, The Standish Group, West Yarmouth, MA, 1994.

Sutcliffe, A., *The Domain Theory - Patterns of Knowledge and Software Reuse*. Lawrence Erlbaum Associates, Inc., London, 2002.

Taylor, A., "IT projects sink or swim." *The Computer Bulletin*, **42** (1) pp 24-26, 2001.

## **Biography**

Peter Merrick is in the final year of his PhD. The basis of his research is into improvements in government procurement and he maintains the web site [www.effortestimation.co.uk](http://www.effortestimation.co.uk). His research has been carried out in the U.K. with the Health and Safety Executive and with the cooperation of the Office of Government Commerce. Peter (along with his advisor) has recently had an article accepted for publication in the journal Requirements Engineering. He earned his first degree in Applied Computing Science at U.E.A. and holds a DipM in Marketing. He is also a qualified teacher. Peter lives with his teenage daughter and likes to sail. [peter.merrick@uea.ac.uk](mailto:peter.merrick@uea.ac.uk)

Pat Barrow's first degree was in Business Information Systems from U.E.A in 1993, and he gained his PhD in Computer Science in 2000 with research investigating stakeholder evaluation in rapid application development. [p.barrow@uea.ac.uk](mailto:p.barrow@uea.ac.uk)