CREWS Validation Frames: Patterns for Validating Systems Requirements

Neil Maiden

Centre for Human-Computer Interface Design
City University, London, EC1V OHB, UK
Tel: +44-171-477-8412
Tel: +44-171-477-8859
E-mail: N.A.M.Maiden@city.ac.uk

Abstract

This presentation proposes a pattern language for socio-technical system design to inform validation of system requirements. The development of this language takes inspiration from Alexander's pattern language for building design in architecture. Our pattern language identifies different types of patterns which fulfil different roles in the requirements engineering process. This pattern-based validation approach has been operationalised in the CREWS-SAVRE software prototype. CREWS-SAVRE applies patterns to both scenarios and requirements documents to detect missing and incorrect system requirements as well as to recommend new requirements which can improve the design of the socio-technical system.

1. Patterns in Requirements Engineering

Patterns are a novel alternative technique to help us better acquire, model and validate system requirements. In simple terms, patterns enable people to reuse knowledge about old solutions to solve similar new problems. However, there is little reported research into patterns for requirements engineering, in spite of the considerable current interest in software patterns for system design and implementation (e.g. Gamma et al. 1995). Indeed, each existing requirements engineering research initiative still tends to have a singular focus on processes, or domains or languages. In contrast, patterns for requirements engineering, as we shall see, cuts across the divisions between process, domain and language. After all, experienced engineers do not separate them when acquiring, modelling and validating system requirements (Guindon 1990).
Patterns which describe the common elements of complex structures were first documented in the field of building architecture. Christopher Alexander, in his book "The Timeless Way of Building" (Alexander 1979), argues that "Beyond its elements, each building is defined by certain patterns of relationships amongst its elements." Patterns can be used to abstract away from the details of particular buildings and captures something essential to the design, for example the principles underlying the building, and the reason why the elements of the building are successful or unsuccessful (Kelly & McDermid 1997).

In this presentation, I advocate the use of patterns that capture something essential to the design of socio-technical systems which include at least one significant software sub-system. The patterns can then be reused to guide the acquisition, modelling and validation of requirements for both socio-technical and software system design. The presentation discusses examples of patterns which capture the essential elements of socio-technical system design, introduces a software prototype which operationalises these patterns to validate system requirements, and proposes an agenda for future development of patterns to aid requirements engineers.

2. CREWS Patterns and Validation Frames

Alexander's original patterns focus on the interactions between the physical form of the built environment and how this form inhibits or facilitates various sorts of individual and social behaviour in it. Bayleet al. (1998) report that this facilitation is more subtle than the simple detection of certain properties which afford actions. Rather, the emphasis is on the characteristics of the environment which might facilitate or inhibit action. For example, in the original 'Beer Gardens' pattern, Alexander suggests that local pubs should have activities around the edges and large tables in the middle to encourage people to cross through the centre, sit at tables and converse with their neighbours. As such, the physical form, or design, of the pub facilitates desirable behaviour in the customers. Furthermore, a pattern also captures the essentials of a 'good design', in that it maximises those characteristics which facilitate desirable actions over those that inhibit these actions.

If these pattern "characteristics" are applied to socio-technical system design patterns, a good pattern must capture the essential elements of the software system, and how the form of this system facilitates and inhibits desirable individual or social behaviour and because of the system. The form of the design can include that of the software system and the physical and social environments of the system's use. Indeed, the pattern can sometimes "design" individual and social behaviour in the context of the software system. Furthermore, because the pattern captures the notion of a 'good' design rather
than a 'bad' design based on past experiences (e.g. prototypical design examples), a requirements engineering team will have more confidence that the design of the software and socio-technical systems will facilitate desirable actions.

As a starting point for designing socio-technical system patterns, we map the key elements of Alexander's patterns to CREWS system requirements, scenarios and patterns:

- the form of the software system is expressed as functional and non-functional requirements statements in a requirements document;
- desirable individual and social behaviour in the environment is expressed as scenarios which are sequences of events and actions which describe desirable future system use in the environment;
- 'good' socio-technical system design is expressed in patterns which capture elements of the software system form (i.e. the requirements) which facilitate or inhibit desirable behaviour (i.e. the scenarios).

Thus validation is achieved by matching the desirable behaviour in the environment (scenarios) to the form of the software system (requirements) using models of good design (patterns). We can view each pattern as being 'superimposed' on different parts of the scenario and requirements document to detect requirements which are missing or which inhibit desirable behaviour. This solution has been implemented in the CREWS-SAVRE prototype software tool developed as part of the European Union-funded ESPRIT 21903 'CREWS' (Co-operative Requirements Engineering With Scenarios) long-term research project. Before describing this software tool, let us first demonstrate the nature of CREWS's socio-technical system design patterns using a prototypical example.

3. The COLLECT-FIRST-OBJECTIVE-LAST pattern

This pattern is true to Alexander's notion of a pattern, in that it captures something essential to the good design of a mechanical device with which a person interacts using one or more personal items to achieve an objective. It is called the COLLECT-FIRST-OBJECTIVE-LAST pattern. The objective is that the person should not leave the personal items behind at the device at the end of a transaction with it. To ensure this, the device imposes a prescriptive sequence so that the person must first collect all items to achieve the objective. Consider a passenger who uses a travel ticket to pass through automatic gates at the entrance to a London Underground station:

OBJECTIVE: to pass through the automatic gates and enter the station;
PROBLEM: passengers sometimes forget to take their valid ticket with them;
SOLUTION: A passenger must collect the ticket from the machine for the gates to open.

This pattern can be seen in the design of other automatic gate machines, for example at entrances to stations on the Paris Metro and Tyne & Wear Metro networks, as well as to the Eurostar train network. It can also be seen in the design of ticket collection points where a customer uses their credit card to collect pre-purchased cinema tickets.

The COLLECT-FIRST-OBJECTIVE-LAST pattern also fulfils the prerequisites for a socio-technical system design pattern. First, the form of the device design has characteristics which facilitate desirable behaviour, that is the user shall not leave their personal items with the device. Second, the pattern includes the rationale as well as successful and unsuccessful elements of the design. In CREWS, we can use this pattern to inform high-level design of the socio-technical system and to acquire system requirements for that design, as well as validate the current requirements document for the system.

4: CREWS-SAVRE's validation frames: implementing patterns

CREWS-SAVRE supports two approaches to scenario-based requirements validation. The first is a guided walkthrough of scenarios by a team of requirements engineers and other stakeholders. In the second approach, CREWS-SAVRE delivers each pattern as one or more validation frames to validate system requirements in a requirements document using an automatic checker. The output is an agenda of issues to be addressed by the requirements engineering team. Each issue identifies a possible missing or incorrect requirement in the requirements document, a missing or incorrect event/action in the scenario, and advice often in the form of new requirements to resolve these errors and omissions. The team uses these issues to change the requirements document, the scenario or both, then to recheck the requirements document again with the automatic checker. It continues this iterative change-and-check process until there are either remaining issues for that scenario and requirements document, or the remaining issues have been recorded and can be tolerated by the requirements engineering team.

The algorithm which is used to validate one or more system requirements with a scenario is a simple, two-pass algorithm. First, for each event/action in the scenario, the algorithm applies one or more validation frames to check the completeness and correctness of the system requirements. Second, for each system requirement, the algorithm applies one or more validation frames to check the completeness of the events/actions in the scenario. This two-pass approach ensures that CREWS-SAVRE
can detect incompleteness in both the requirements document and the scenario. The 'intelligence' of the approach, however, is in the validation frames which encapsulate 'good' design practice from the CREWS patterns. Let us look at these validation frames in more detail.

CREWS-SAVRE is a prototype software tool which has been designed to guide systematic scenario-based requirements engineering. It has been developed on a Windows-NT platform using Microsoft Visual C++, Visual Basic 5 and Access, thus making it compatible for loose integration with commercial requirements management and computer-aided software engineering software tools. The current version has 5 components:

- the domain modeller, which the requirements engineering team uses to model a problem domain as a collection of agents, objects and actions;
- the use case modeller, which the team uses to model each use case from the domain model as well as parameters for generation of scenarios from that use case;
- the scenario generator, which automatically generates one or more scenarios from a use case model;
- the scenario presenter, which the team uses to walk through a generated scenario and guide validation of requirements with that scenario;
- the requirements validator, which the team uses to apply validation frames to validate system requirements using scenarios.

Furthermore, CREWS-SAVRE has been designed to be integrated with Rational's RequisitePro requirements management tool. This tool handles system requirements, thus enabling the development team to focus on novel implementations such as use case modelling, scenario generation and scenario-based requirements validation.

The component supports four main functions. The validation dialogue function manages dialogue with the requirements engineering team to select the requirements, scenarios and frames for validation, and to handle the list of issues which arises from validation. The validation algorithm applies the chosen validation frames to the chosen scenario and requirement(s) to detect all scenario and requirement omissions and errors. The natural language checker is a simple function which detects possible semantic equivalence between one requirement statement and one scenario event/action using the syntactic names of actions, agents and objects referenced in both. We have refined the function to handle plurals, and the next version of the component will provide a problem domain lexicon to refine the function even further. The last function, validation frame administration, enables a user to add, change and remove validation frames from the component.

Acknowledgements
The authors wish to thank everybody who have contributed to the development of the patterns and software tool. They also thank the other members of the CREWS project for comments and advice. This research is funded by the European Commission ESPRIT 21903 'CREWS' long-term research project.

References


