An Experiment in Inspecting the Quality of Use Case Descriptions

Karl Cox

National ICT Australia/School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
karlc@cse.unsw.edu.au

Aybüke Aurum

School of Information Systems, Technology and Management, University of New South Wales, Sydney, Australia
aybuke@unsw.edu.au

Ross Jeffery

National ICT Australia/School of Computer Science and Engineering, University of New South Wales, Sydney, Australia
karlc, rossj@cse.unsw.edu.au

Achieving higher quality software is one of the aims sought by development organizations worldwide. Establishing defect free statements of requirements is a key strategy for achieving improvements in quality. In this paper we present the results of a laboratory experiment that explored the application of a checklist in the process of inspecting use case descriptions. We compare the checklist with others in the literature then report experimental findings. A simple experimental design was adopted in which the control group used an ad hoc approach and the treatment group was provided with a six-point checklist. The defects identified in the experiment were classified at three levels of significance: i. Internal to the use case ii. Specification impact, and iii. Requirements impact. It was found that the identification of requirements defects was not significantly different between the control and treatment groups, but that more specification and internal defects were found by the groups using the checklist. In the paper we explore the implications of these findings.

ACM Classification: D.2.1 (Software – Software Engineering – Requirements/Specifications)

1. INTRODUCTION

Requirements are critical to system validation as they guide all subsequent stages of systems development. Inadequately specified requirements generate systems that require major revisions or cause system failure entirely. The requirements validation activity endorses the specification document as an adequate external description of the system to be implemented. Providing complete, unambiguous, correct and consistent requirements and ensuring all requirements satisfy industry standards improves the chances for a higher quality software product. However, since the general industrial acceptance of use cases (Jacobson *et al*, 1992; OMG, 2003) as one of the most common means of describing and analysing requirements (Neill and Laplante, 2003), achieving unambiguous, correct, consistent and complete requirements description has been made both easier and more difficult. The engineer can now apparently write use cases at ease and the customer can now apparently easily validate their requirements through the use cases. There are problems too. Because of the informal

Manuscript received: 15 December 2003
Communicating Editor: Didar Zowghi

nature of use case descriptions, it is inherently easy to introduce ambiguity into the documentation (Gause and Weinberg, 1989). Also, because one use case can satisfy several requirements and one requirement can be described by several use cases, it is often difficult to achieve completeness and consistency. These problems, if not resolved, can cause later design and implementation difficulties.

Defect detection in requirements documents is seen as one of the most effective and efficient quality assurance techniques in software engineering (Porter *et al*, 1995; Parnas and Lawford, 2003). Such defect detection should also be applied to use cases. Therefore, it seems reasonable to establish an inspection process for use case descriptions since these are a major requirements document. However, there has been relatively little literature specifically on use case inspections. Bittner and Spence (2002), for example, provide a chapter on conducting typical reviews but they do not suggest more than general advice on how to do inspections, not particularly what to look for. Adolph *et al* (2003) present a two-tiered review pattern. Its first review attempts to eliminate "'noise' caused by spelling, grammatical, formatting and technical errors" (p.65) in the use cases. The second round of reviews explores whether the use case meets business needs, the specification and if it can be built. But no specific checklists, for example, are provided. Cockburn (2001) provides a pass/fail test for use case descriptions which can be taken as a quasi-inspection checklist that addresses whether the description meets its goals, is really required and is feasible (Section 2.1 provides more details).

Despite this apparent disinterest in inspections in the use case literature, it is clear that inspections are a very valuable exercise. Numerous approaches to inspections have been suggested since Fagan's original work (Fagan, 1976). Porter *et al* (1995) conducted an experiment concluding that in terms of defect detection rate the Scenario approach was superior to both the Ad Hoc and Checklist-based approach. In replication, Miller *et al* (1998) found that the defect detection rate of the Scenario approach was once again superior to that of Checklist-based reading. However, Fusaro *et al* (1997) obtained different results. They could not find any empirical evidence of better defect detection performance when using Scenarios. Sandahl *et al* (1998) also could not support the superiority of the Scenario method in their study. In a later replication of the same experiment, Halling *et al* (2001) found results that were quite different from the other studies. Their large-scale experiment (150+ undergraduate students) led to the conclusion that the Checklist-based reading was overall more effective on the individual level, whereas the Scenario approach gained effectiveness when applied to a certain target focus. Rombach *et al* (2003) investigated the usage of roles in ER-Modelling inspection meeting, comparing Ad Hoc based reading with checklist-based reading. Their findings showed that the participants using a checklist had a significantly higher detection rate than the Ad Hoc groups. Wohlin and Aurum (2003), who studied the impact of an individual reviewer on the effectiveness of an inspection team, concluded that a tailored Checklist may indeed be cost effective.

Brykczynski (1999), in an extensive overview of checklists for inspections, points out that that many checklists are too generalized and need to be tailored. Travassos *et al* (1999) pointed out that the focus in inspection when reviewing requirements should be more on verifying the content (its semantics) and less on verifying the syntax. Lauesen and Vinter (2001), Wohlin and Aurum (2003) and Rombach *et al* (2003) all classified defects according to the context they were in. The literature indicates it is considered important to (i) tailor the checklist to the product under inspection, i.e. the use case description, and (ii) focus more on the semantics (requirements) more than the syntax.

There is an interest in using use cases and scenarios in conducting inspections of documents (Dunsmore *et al*, 2003; Thelin *et al*, 2003) but the assumption is that the use cases have already been validated. Thus there appears to be a lack of formal inspection of use case descriptions themselves prior to using them as tools in the inspection process. Though some checklists have been suggested as a means of inspecting use case descriptions (see Section 2.1), there is a greater body of work

making use of descriptions in inspections rather than addressing the quality of the descriptions themselves. This paper presents and then compares a use case checklist against those available in the literature. Our proposed checklist is then used in a formal two-stage inspection experiment against an ad hoc approach.

The next section discusses the use case checklist that was the instrument for the experiment and compares it to other use case checklists in the literature. Section three presents the experimental design. Section four presents the results from the experiment. Section five discusses conclusions, implications for use case inspections and future work.

2. USE CASE DESCRIPTION CHECKLISTS

Our use case checklist is presented in Table 2. We then compare it to others published in the literature (Section 2.1). The checklist acts as the instrument in the experiment. Our checklist is only about the use case description. Importantly, it thus differs from other use case checklists in that it does not address elements of the use case template, such as triggers and pre- and post-conditions in detail or the diagram at all. (We do not address the diagram and other aspects of the template in this checklist since we take an incremental approach to devising and testing checklists. We will consider a checklist for these in future work. It is also the case that the diagram and template elements have received a large amount of attention in the literature in comparison with use case descriptions.) The derivation of the checklist is presented in Table 1 and is adapted from a means of assessing the

Checklist Elements		**Derivation**
Coverage	Scope	Jackson (1995) – refined notions of completeness for requirements.
	Span	
Cogent	Text Order	Gernsbacher (1996; 1997) – structure building framework Graesser *et al* (1996) – inference building (Question -> Reply to Question)
	Dependencies	Jacobson *et al* (1992), OMG (2003) – representing a complete transaction. e.g. Trabasso *et al* (1989), Goldman et al. (1996) – local and global coherence. Garnham and Oakhill (1996) – referential continuity. Graesser *et al* (1996) – inference building (Question -> Reply to Question)
	Rational Answer	Gernsbacher (1996; 1997) – structure building framework Grasser *et al* (1996) – inference building (Question -> Reply to Question) Anda *et al* (2001) – the realism of the use case
Consistent Abstraction		e.g. Anda *et al* (2001)
Consistent Structure	Variations	Kulak and Guiney (2000), Achour *et al* (1999) – keep variations to a separate section
	Sequence	e.g. Schneider and Winters (1998), Achour *et al* (1999) – consistent sequential numbering
Consistent Language		e.g. Pooley and Stevens (1999) – avoid passive voice; the consensus is there are many grammatical elements to avoid. Some structures might improve comprehension e.g. Graham (1998)
Consideration of Alternatives	Separation	Kulak and Guiney (2000) – keep variations to a separate section. Alexander (2003) – failure to deal with exceptions leads to system failures.
	Viable	Alexander (2003) – failure to deal with exceptions leads to system failures.
	Numbering	Cockburn (2001), Achour *et al* (1999) – there should be consistency in numbering.

Table 1: Derivation of the Checklist (adapted from Cox, 2002)

understandability of use case descriptions and of use case writing guidelines; we refer the interested reader to these references (Cox and Phalp, 2000; Cox *et al*, 2001; Phalp and Cox, 2002; Cox, 2002).

2.1 Comparing Use Case Description Checklists

Anda and Sjoberg (2002) present an inspection checklist for the use case model. In assessing the checklist, experimental results, however, showed that experienced inspectors found more defects without the checklist; but that in another experiment, students found more defects with the checklist than without. Their findings also showed that different stakeholders found different types of defects. Since Anda and Sjoberg address the entire use case model, their study differs to ours in that we address only the use case description, not the diagram. Their checklist addresses, primarily, actors, completeness, correctness, inputs and outputs. Their results show that practitioners were interested in finding defects related to actors, triggers and pre- and post-conditions. They did not find many defects in the descriptions. The checklist for the description part of the use case model considers completeness, rationality, abstraction and understandability.

McBreen (2001) proposes a use case checklist that covers a range of aspects from project team to goal achievement/failure. Wiegers (2003) presents a use case inspection checklist that considers

1. Coverage.
   - 1.1. Span: The use case should contain all that is required to answer the problem. That is, is there enough information in the description or is some detail missing?
   - 1.2. Scope: The use case should contain detail only relevant to the problem statement. Extra unnecessary information provided is out of problem scope and is not required.
2. Cogent.
   - 2.1 Text Order: The use case should follow a logical path. Is this path logical or are events in the description in the wrong order?
   - 2.2 Dependencies: The use case should complete as an end-to-end transaction (which can include alternative/exceptional flows). Does the actor reach a state that stops the transaction from terminating as we expect?
   - 2.3 Rational Answer: The logic of the use case description should provide a plausible answer to the problem. Are there any events that appear out of place or you recognise as incorrect?
3. Consistent Abstraction. The use case should be at a consistent level of abstraction throughout. Mixing abstraction levels (problem domain, interface specification, internal design mixes) will cause difficulty in understanding. Is abstraction consistent?
4. Consistent Structure.
   - 4.1 Variations: Alternative and exceptional events should be excluded from the main flow and should be in a separate section.
   - 4.2 Sequence: Numbering of events in the main flow should be consistent. Are there any inconsistencies?
5. Consistent Language. Simple present tense should be used throughout. Adverbs, adjectives, pronouns, synonyms and negatives should be avoided. Have they been used?
6. Consideration of Alternative Flows and Exceptional Flows of Events.
   - 6.1 Viable: Alternatives and exceptions should make sense and should be complete. Are they?
   - 6.2 Numbering: Alternative and exception numberings should match the numbers in the main flow. Do they or is there inconsistency?

Table 2: Use Case Description Checklist

ambiguity (in an ambiguous way – is the description "clearly written, unambiguous"?). His checklist also explores completeness and abstraction. Klariti (2003) presents a checklist that is similar to that of Wiegers. Kamthan (2003) builds upon Wieger's checklist and the guidelines set out by Cockburn (2001). Kamthan considers the importance of scope and focuses specifically on completeness. Tervonen (2003) presents a seven-point checklist for use cases. This addresses the traceability and conformity of the use cases to other requirements documents as well as whether the right functional requirements have been described. Cockburn presents a pass/fail test for use case descriptions. He addresses the abstraction level of the description, whether the description completes successfully and its intention is clear. Cockburn is also concerned that the interests of the stakeholders involved in or affected by the use case are protected – a different consideration to other checklists. It is the case that use cases have been used as requirements documents in inspections experiments (Fusaro *et al*, 1997; Miller *et al*, 1998; Porter *et al*, 1995; Regnell *et al*, 2000). However, the focus of these papers is on generic checklists and scenarios of ways in which to approach inspecting the documents, i.e. perspective-based reading. In contrast, our checklist is *tailored* to the use case description. There is only one other reported empirical study that we are aware of that examines a tailored use case checklist in experimental conditions (Anda and Sjoberg, 2002).

Table 3 compares the Use Case Checklists available in terms of how they refer to the Description only. Therefore, other elements such as those relating to Diagrams are not considered here since they are out of scope for our specific purposes. The top row in Table 3 shows how the checklists compare to the elements in ours (Table 2). If a checklist considers Coverage but only its Span element, then there will only be reference to Span and a short description of how it is addressed (though the authors may not use the terminology we do). For example, Anda and Sjoberg (2002) address the Coverage aspect in terms of Span by asking whether there are any missing inputs or outputs or missing variations. The last column in table three, 'Other', is for those aspects that might be covered by other checklists that we do not consider.

In terms of Coverage, all authors ask whether the description has enough information to ensure its completeness (Span) though only Wiegers (2003), Kamphan (2003) and Klariti (2003) consider if too much information is present (Scope). Indeed, Wiegers and Klariti address the use case as a whole in terms of whether the use case is a discrete task. Kamphan agrees and goes further to the depth that we consider: are all events necessary for communicating the transaction? Tervonen (2003) considers Span, asking that only a typical way of using the system is described. Though Table 3 shows that Cockburn (2001) explicitly addresses the Scope of the description, he has a different interpretation of its meaning. Cockburn describes Scope as: "Does the use case treat the system mentioned in scope as a black box? (The answer must be yes if it is a system requirements document, but may be no if the use case is a white box business use case). If the system in scope is the system to be designed, do the designers have to design everything in it and nothing outside it?" (p. 212). This does not necessarily address whether enough information has been included in the description to reach its goal and is more concerned with abstraction.

Aspects of Cogent are considered by all. However, it is only Anda and Sjoberg (2002) that address all three attributes. The other authors all address Rational Answer. Kamphan and Cockburn also consider Dependencies. Tervonen looks at the overall use case document, asking that it have a proper structure of a beginning, middle and end. Whether this refers to the Text Order is unclear.

The clear difference in the checklists compared to ours, is that none suggest variation paths should be kept out of the main flow of events, except McBreen (2001), who states that all recoverable failures should be recorded as extensions. Though Anda and Sjoberg consider whether all Variations have been documented, they do not state that these should not appear in the main flow

Checklist	Coverage	Cogent	Structure	Abstraction	Language	Alternatives	Other
Anda and Sjoberg (2002)	Span – missing inputs/outputs, missing variations	Text Order – expected inputs and outputs. Dependencies – each input has an output. Rational – events relate to goal of Use Case	No.	No interface or design detail	Use concrete terms	Viable – each event relates to goal of use case.	Triggers; Pre- and post-conditions
Wiegers (2003)	Span – complete events; Scope – discrete task	Rational – Feasible	No.	Essential level only, no design or implementation	Clearly written, unambiguous	Viable – Feasible and Verifiable	Measureable result; Pre- and post-conditions
McBreen (2001)	Span – are all failure conditions considered?	Rational – does the UC do what is expected?	Variations – All recover-able failures recorded as Extensions	Precision of detail consistent – same conceptual level (eg. internal design/ external design only)	Style should be consistent. Active verb phrase	Viable – Recoverable failures recorded	Goal success/failure. Presentation, formatting consistent.
Kamphan (2003)	Span – all necessary detail. Scope – are all events pertinent to the task? Is the UC one discrete task?	Dependencies – internal and external should be documented. Rational – Is event useful/feasible?	No.	No internal design or implementation. Essential, general level.	Clearly written, unambiguous	Viable – documented and feasible?	Verbose – more than one discrete action per event should be removed. Are goals met? Pre-conditions identified?
Klariti. com (2003)	Scope – discrete task Span – all alternative/exception courses documented	Text Order – Is every step pertinent to that task? Rational – Is each course feasible?	No.	Essential level (no design or implementation)	Clearly written, unambiguous and complete	Viable – Is each course feasible?	Verifiable?
Tervonen (2003)	Span – typical way of using the system but nothing more?	Text Order – well structured: beginning, middle, end (?)	No.	Concrete, specify the most important requirements.	Clear	No.	Traceable to requirements specification. Based on roles of users.
Cockburn (2001) – pass/fail test	Scope – is everything in scope as a black box or internal scenario? – Consistent Abstraction	Dependencies – Does UC run to success?	Variations – Does it permit the right variations in sequencing? (Allowed in main flow?)	Is the event goal level correct, black box, not user interface design?	Is the information clear in the events? Active phrase clear: '– who kicks the ball'	Viable – Is this what the system actually needs?	Triggers, stakeholders, pre + post-conditions, guarantees

Table 3: Comparison of Checklists that address the Description

of events. To allow alternative paths as part of the main flow of events will add complexity and increase the risk of misunderstanding the 'happy day scenario' (Schneider and Winters, 1998). Cockburn explicitly checks whether the use case permits the right variations in the sequencing of the description but does not address whether these need to be in a separate section to the main flow (though his pass/fail list does address extension conditions).

All authors consider Abstraction, noting that design and implementation should not be part of use case descriptions (unless the use case is about internal design – McBreen and Cockburn). Tervonen asks that the use cases are concrete and specify the most important functional requirements.

Though it can be seen that all checklists address Language, most are ambiguous as to what unambiguous language might be. Tervonen considers that the use cases be "clear", presumably unambiguous. McBreen, though, suggests the use of active verb phrases. Cockburn asks that it should be clear who is responsible for actions; typically this is clearer in an active sentence, rather than passive. In an earlier work, Anda *et al* (2001) present style guidelines as suggested by Achour *et al* (1999) in a reduced format as recommended by Cox and Phalp (2000). Cockburn also presents grammar guidelines elsewhere in his book.

Consideration of Alternatives: almost all authors suggest the Alternatives/Exceptions should be feasible or at least required (Cockburn), except McBreen, who notes that all recoverable failures should be recorded. Tervonen does not address this facet.

Finally, all the authors consider other aspects, such as pre- and post-conditions and goal success and failure. We have chosen to ignore these simply because we are interested in the dialogues or flows of events and have assumed that pre- and post-conditions and triggers are correct, consistent and complete in this instance.

2.2 Types of Defects

To quantify the differences in the checklist (Table 2) in terms of their considered ease of detection, we decompose the list into three components in terms of their impact upon the problem: (a) Minimal Impact (internal to the use case), (b) Specification Impact (though the use case is still expected to meet the requirement), and (c) Requirement Impact (the effects desired in the problem domain are not achieved) – this is the most severe. Impact refers to the distance from the use case in the problem space that defect can affect. The further the distance, the greater the risks. Clearly if there is simply a typographical/grammatical mistake, then this is less likely to affect the overall success of the project. However, if there is something just wrong in the use case description such as a missing piece of critical information (span in Coverage) then its impact is at the requirement level and it is more severe. Table 4 shows the categorization of defects according to our checklist.

Use Case Description impact. Consistent Language is important for readability of the use case description. It is unlikely that grammar issues cannot be resolved from within the use case model. Consistent Structure: Sequence is concerned with the correct labelling (numbering) of events in the

Use Case Description Impact	Specification Impact	Requirements Impact
Consistent Language	Consistent Abstraction	Coverage
Consistent Structure: Sequence	Consistent Structure: Variations	Cogent
Alternatives: Numbering		Alternatives: Viable

Table 4: Categorization of Checklist Defect Types

main flow and is entirely internal to the description. Consideration of Alternatives: concerns the numbering of alternative events in relation to main flow equivalents and is entirely internal to the description.

Specification impact. Consistent Abstraction is considered a specification issue in that there is a danger of mixing problem domain / internal design detail into the specification task where it is not always appropriate. Consistent Structure: Variations in the main flow might affect the specification by wrongly ordering events or assuming transitive dependencies where there are none.

Requirements impact. Coverage: Scope and Span refer to the completeness of the requirement. If too much or too little information is included in the description it is possible that part of the requirement will be missed or a new, unwanted requirement added. Cogent: Text Order has an impact on the order of events at the interface of the machine. The requirement may or may not be met but the order in which things are done as a result of the wrong ordering could at its most simple just reorder a task, affecting usability of the tool, or it might go further into the World and affect the requirement. An example from a use case description in the reported experiment (Section 3) shows the ejection of cash occurring before the ejection of the card from the ATM. This does not meet the security requirements since the risk of forgetting the card is greatly increased if the money comes first. Indeed, this was a recognised design flaw in the earliest ATMs. Cogent: Dependencies considers whether a use case terminates as expected. If it does not then the requirement will not be met. Cogent: Rational Answer examines whether a step or series of steps makes sense in terms of the requirements for the system or even if the steps are implementable (Adolph *et al*, 2003). Consideration of Alternatives: Viable is similar to Cogent: Rational Answer. It explores where the alternative / exception path is feasible and complete. If it is not, it will not meet its requirement.

3. EXPERIMENTAL DESIGN

We conducted an experiment that took the form of a two-stage review process, with the treatment being the use of the checklist of Table 2. The control group applied an ad hoc approach where they were provided with no guidance except requirements elicitation notes and the use case model[1]. A pilot study was conducted prior to the experiment to assess the suitability of the experimental design, the instrument, the time slots, the questionnaires and the clarity and difficulty of the tasks. The pilot subjects were PhD students experienced in inspections. Minor adjustments were made to the materials such as simplification of the use case descriptions and checklist, including examples, and the time slots were deemed sufficient for the experiment.

3.1 Experimental Subjects

As can be seen in table five, there are two groups, group A (the control) of 73 subjects and group B (the treatment) of 79 subjects. The subjects are final year undergraduate computer engineering and software engineering students undertaking a course in Total Quality Management (TQM), of which software inspection is taught. The subjects were taught about use cases as a normal part of their course and many are experienced users. As part of this tuition, subjects were taught about defects in use case descriptions in general – no reference was made to the checklist. The experiment was conducted in the normal tutorials of the TQM course over a period of four days. Nine tutorial groups

[1] Note that although we compare an ad hoc approach against our checklist, we do not suggest that the other checklists discussed in Section 2 of the paper might be inferior to ours in locating defects; they are simply not tested here.

Group A (control – ad hoc) (73 subjects)		**Group B (treatment - checklist)** (79 subjects)	
Task	Time (min)	Task	Time (min)
Individual Inspection	55	Individual Inspection	55
Individual Questionnaire (on individual inspection activity)	5	Individual Questionnaire (on individual inspection activity)	5
Group Inspection	30	Group Inspection	30
Individual Questionnaire (on group inspection activity)	5	Individual Questionnaire (on group inspection activity)	5

Table 5: Experimental design, subject numbers and time scales

were randomly assigned to either the control or the treatment with the only proviso a relatively equal number of subjects in each experimental group.

All subjects were presented with a consent form asking their permission to use any data from the experiment prior to the experiment. Those who did not sign the consent forms (four students) are not included in the analysis. Three other subjects' results are also discarded because they arrived late for the experiment.

3.2 Course of the Experiment

The experiment is in two parts, as described in table five, showing a two-stage review: individual review followed by group review, with one group, B, the treatment, and group A, the control. In the first part, subjects were presented with a 'typical' requirements elicitation document – this took the form of meeting notes between the development team and the customers (in a form suggested in Bray, 2002). The subjects were also presented with a use case model (diagram and corresponding descriptions) derived from the requirements elicitation document. The treatment group was also given the checklist as shown in Table 2. The treatment group subjects were not aware of the instrument (checklist) prior to the experiment but all experimental subjects were familiar to the idea of checklists, having been taught about and used a checklist for code inspections in an earlier part of the TQM course. Subjects had 55 minutes to complete the individual inspection activity where they recorded the defects on a purpose-made defect form. And possible solutions – in an attempt to make sure the subjects avoided recording defects in a 'casual' manner and to check whether false positives were identified. If they were, this was not taken into account in the final analysis but was checked upon to make sure the subjects were providing credible answers. The subjects then had five minutes to complete a short questionnaire about the inspection they had just conducted.

The subjects were then given a ten-minute break before conducting the second part of the experiment. This was a group inspection activity. Subjects were randomly assigned into groups of three (or four, rather than two) at that moment. They were given thirty minutes to complete the inspection. The subjects were not assigned roles at the start of the inspection since we were interested in how the groups would work together to complete the task. After the thirty minutes, the subjects were asked to complete a second questionnaire asking about the group inspection activity. In the following two weeks, after analysis of the defect data, the subjects received feedback regarding the experiment and their success in the tasks.

The subjects were presented meeting notes as an initial requirements elicitation document. They were also presented a use case model of a simplified ATM (derived – except for introduced defects

– from the elicitation notes). Though the ATM is occasionally considered inappropriate for requirements case studies (Lauesen, 2002), we find the ATM suitable for use cases since use cases are primarily a means of describing system external behaviour. There are five use cases described in total. The defects are not evenly placed among the five. The number of defect types introduced are, seven use case, four specification and nine requirements defects. The experimental materials *and* data can be found at Cox *et al* (2004).

3.3 Hypotheses

There are ten null hypotheses, as described below. Hypotheses $H1_0$-$H4_0$ refer to the overall results of the experiment – there is no distinction made between types of defect. The subsequent six hypotheses address the different defect types separately.

$H1_0$: Use of the checklist will find similar numbers of defects to an ad hoc approach in the ***individual*** inspection task.
$H2_0$: Use of the checklist will find similar numbers of defects to an ad hoc approach in the ***group*** inspection task.
$H3_0$: The ***ad hoc group*** inspection task will find similar numbers of defects to the ***ad hoc individual*** inspection task.
$H4_0$: The ***checklist group*** inspection task will find similar numbers of defects to the ***checklist individual*** inspection task.

$H5_0$: Use of the checklist will find similar numbers of ***Use Case defects*** to an ad hoc approach in the ***Individual*** inspection task.
$H6_0$: Use of the checklist will find similar numbers of ***Use Case defects*** to an ad hoc approach in the ***Group*** inspection task.

$H7_0$: Use of the checklist will find similar numbers of ***Specification defects*** to an ad hoc approach in the ***Individual*** inspection task.
$H8_0$: Use of the checklist will find similar numbers of ***Specification defects*** to an ad hoc approach in the ***Group*** inspection task.

$H9_0$: Use of the checklist will find similar numbers of ***Requirements defects*** to an ad hoc approach in the ***Individual*** inspection task.
$H10_0$: Use of the checklist will find similar numbers of ***Requirements defects*** to an ad hoc approach in the ***Group*** inspection task.

The alternative hypotheses are two-tailed. We would like there to be a positive directional significance – the checklist approach is expected to find more defects than the ad hoc – but we cannot expect this, especially since there is contradictory evidence on the effectiveness of use case inspection checklists (Anda and Sjoberg, 2002). The hypotheses were tested by taking counts of the defects identified and comparing them. As this is a simple comparison of samples, we used the t-test for significance testing, but checked for an even distribution beforehand.

4 RESULTS

Table 6 describes the mean number of defects found and standard deviations (SD) by both the control (individual and group A) and the treatment (individual and group B). It can be seen that the

Group A (control)	Mean	SD	Group B (treatment)	Mean	SD
Individual inspection	5.6	2.48	Individual inspection	7.7	2.56
Group inspection	7.8	2.61	Group inspection	10.4	1.89

Table 6: Mean and standard deviation of defects found

Hypothesis	Significance
H1: B individual Total - A Individual Total != 0	p <= 0.0001
H2: B group Total – A group Total != 0	p = 0.0004
H3: A group Total – A individual Total != 0	p = 0.0012
H4: B group Total – B individual Total != 0	p <= 0.0001

Table 7: Results of t-tests on the hypotheses

application of the checklist found more defects than an ad hoc approach. Note that there are 20 defects in total in the use case descriptions. (One of the authors independently assessed the use case descriptions for defects. The other authors checked the overall statistical results for validity.)

The mean number of defects found is relatively low for all inspections. Only the treatment group inspection found on average half the number of defects. However, it is clear that the use of the checklist does find more defects than the ad hoc approach. Group B's *individual* inspection mean (7.7) is only marginally small than Group A's *group* inspection mean (7.8). The Standard Deviations show a large amount of variance from the means, especially for group A's individual inspection (SD: 2.48), since its mean is only 5.6. This indicates a wide variation in numbers of defects found by the subjects. The hypotheses are significance tested by independent sample t-tests. Alpha is set at 0.05. The results are described in Table 7.

Table 7 shows that the null hypotheses have been refuted. Alternative hypothesis one is found to hold true. The individual mean of the checklist group was significantly higher than that of the ad hoc individual scores. This is also true for alternative hypothesis two, when comparing groups. This seems a straightforward case of the checklist being better than the ad hoc. The distribution is relatively normal for both groups in terms of the total number of defects identified (Figure 1, left)

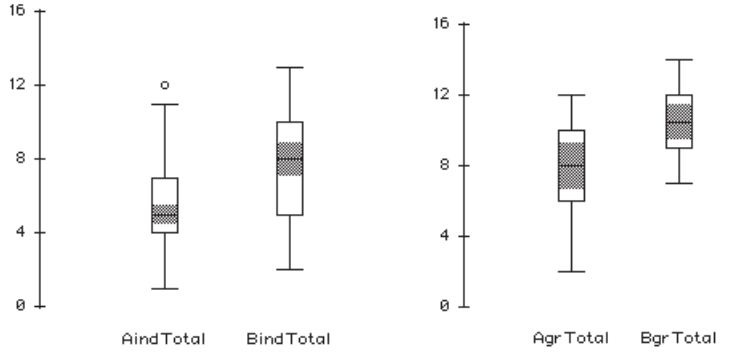

Figure 1: Boxplots for total defects identified by individuals (left) and groups (right)

in the individual task. Group A has two outliers of 12 defects found. This does not affect the analysis since there is already a significant difference. Clearly the median for group B is higher than for A.

In terms of group performances, Figure 1 (right) shows a similar difference in medians though the distribution for group B is much narrower and more even than in the individual part, indicating more agreement between groups than individuals (though there are many more individuals than groups, of course). Note that group A no longer has outliers showing a more even distribution. There is no overlap in confidence intervals.

4.1 Comparison of Real and Nominal Groups

Nominal group defects are the number of different defects identified by individuals prior to the group meetings. Research shows that the number of nominal defects identified is higher than or the same as the number of defects identified by the actual (real) groups (Porter *et al*, 1995; Votta, 1993). Biffl and Halling (2003) further show that a mix of inspection techniques yields more discovered defects for nominal teams than those employing only one type of technique. To compare the performance of real and nominal groups, we randomly selected eight groups from the control group and eight from the treatment group to assess whether this was the case. We found that the unique number of defects identified by the individuals (the nominal) was in fact higher than those identified by the same individuals as a group in all but two groups (one a control group, the other a treatment group) where the same numbers of defects were identified.

4.2 Categorizing the Defects

We now categorize the defects as described in Section 2.2. The defects are categorized by the severity of impact: Use Case impact (least severe), Specification impact and Requirements impact (most severe).

An example of Use Case impact taken from the experimental material:

ACCESS ATM

Actors: User
Context: User wants to use the ATM.
Pre-condition: ATM in ready state for new User

1. User inserts card into ATM.
…
5. **Customer** selects account. (*Consistent Grammar : synonym: User **UC***)
6. ATM displays User options.
…

Event 5 shows a *Customer* actor whereas the Actor is listed in the template above as *User*. It is unlikely that this Grammar mistake in this particular problem context will affect the success of the machine in meeting the requirements.

An example of Specification impact taken from the experimental material:

For the same use case *Access ATM*:
…

2. ATM asks for a PIN.
3. User types in the numbers of **his** PIN and presses the Enter button (*Consistent Abstraction **SPEC** – this is interface design, should just be: User enters PIN into ATM*; *Grammar: his: remove **UC***)
…
Event 3 contains two errors, one a use case problem and the other a specification concern: "presses the Enter button". This describes interface design and is a Consistent Abstraction mistake.

An example of Requirements impact taken from the experimental material:

CHANGE PIN

Actors: User
Context: User wants to change their PIN.
Pre-condition: User already logged onto the ATM

Main Flow of Events:

1. User selects 'Change PIN'. (*Span : no reference to enter current PIN **REQ***)
2. ATM prompts her to enter new PIN. (*Grammar: pronoun; User **UC***)
…
There is an event missing between 1 and 2 where the ATM should prompt the User to re-enter their current PIN, as stated in the requirements.

It is not always straightforward to determine whether a defect is a use case, specification or requirement defect, since this is dependent upon the problem being solved. As an example, the following snippet is taken from a use case in the experiment (use case 2):

5. ATM releases cash.
6. User takes cash.
7. ATM releases card.
8. User takes card.

The order of the events here is incorrect. This appears to be a specification issue, ordering the way in which materials are released from the machine because, after all, the User gets both the cash and the card back. However, this is a requirement problem. Some of the first ATMs worked in the order described. There was a high card loss rate or cards being retained by the ATMs. Users were taking their money and leaving, forgetting about the card. Banks soon realised this and made it a requirement that the card must be returned to the User before anything else was ejected, be it cash or receipt. However, some ticket machines, such as at railway stations, will issue a ticket before returning any change, others will return change with the ticket and some issue change whilst the ticket is being printed. The problem you are trying to solve will determine what is a critical requirement and what is not.

Table eight shows that the ad hoc approach (A) identifies few Use Case defects: 23 and 36 percent of Use Case defects were identified respectively for individual and group inspections. When compared to the checklist approach (B), 47 and 64 percent respectively, it is clear that B identifies more Use Case internal defects. This is not the case for Requirements defects. The ad hoc approach

Individual A	Use Case	Spec.	Req.	**Individual B**	Use Case	Spec.	Req.
Total	120	14	275	Total	240	79	289
Mean	1.64	0.19	3.77	Mean	3.29	1.08	3.96
Percentage	23.43	4.75	**41.89**	Percentage	47	27	**44**
SD	1.67	0.43	1.46	SD	1.56	0.72	1.37
Group A	Use Case	Spec.	Req.	**Group B**	Use Case	Spec.	Req.
Total	55	9	108	Total	121	40	120
Mean	2.50	0.41	4.91	Mean	4.5	1.5	4.4
Percentage	35.7	10.3	**54.6**	Percentage	64.29	37.50	**48.89**
SD	1.97	0.59	1.48	SD	1.40	0.80	1.05

Table 8: Breakdown of totals, means, percentages and standard deviations for defect types identified

identifies only slightly fewer defects in the individual task (42 to 44 percent respectively) than the checklist approach. In the group task, however, the ad hoc group A identifies a larger percentage of requirements defects than group B (55 to 49 percent respectively). The identification of Specification defects is not too successful by either ad hoc or treatment.

The standard deviations (SD) show a similar variance for both groups. Individually, both groups A and B are similar. Though they are similar, the total counts of defects are quite different between the groups except for Requirements defects. The variation in individual defects counted is minimal (on average 1.5 from the mean for group A, which is 3.8, and 1.4 for group B, whose mean is 4), indicating that the variation in numbers of defects found is slightly higher for A. Group A's deviation in the group task is larger than group B's though it does find more defects on average. Group B has less variation in the numbers of defects found in the group task since its deviation is only one away from the mean. The question now arises as to whether there is a significant difference between the ad hoc and checklist approaches in the types of defects identified. Independent sample two-tailed t-tests were applied (alpha = 0.05), as shown in Table nine.

It is interesting to explore the distributions for the Requirements defects to assess whether any skewness has affected the outcome of the tests. Figure 2 (left) shows boxplots for Individual marks for groups A and B. As can be seen from the confidence intervals (shaded areas) in the box plots, there is overlap showing that there is no significant difference between the groups. However, it is clear that there is an uneven distribution of defects for the treatment group (B). The third percentile and the median are the same (4). There are also a number of outliers. However, when outliers are removed from the analysis the result is still insignificant; as such they remain in the equation. When

Data Tested			**Significance**
	H5	A, B Use Case	p <= 0.0001
Individual	H7	A, B Specification	p <= 0.0001
	H9	A, B Requirement	p = 0.64
	H6	A, B Use Case	p = 0.0003
Group	H8	A, B Specification	p <= 0.0001
	H10	A, B Requirement	p = 0.22

Table 9: Significance values for Defect Types

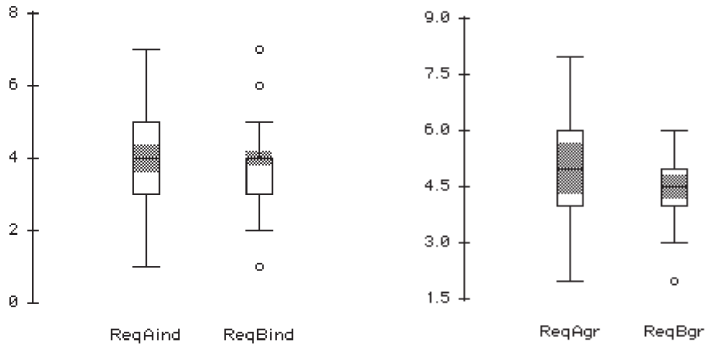

Figure 2: Boxplots for Individual (left) and Group (right) Requirements Defects for A and B

comparing the number of requirements defects found in the group task, Figure 2 (right) shows there is no significant difference: the confidence intervals overlap. The distribution is relatively normal for both A and B. Group A has a wider range than B. B also has an outlier of two. Its removal does not alter the significance and thus is included.

The treatment has found far more use case related and specification defects than the control. This is good in terms of clarifying the description, for instance in removing grammar problems that might introduce ambiguity. However, the checklist has not improved defect detection where it really matters, in the identification of requirements defects. However, the counter point argues that since similar numbers of requirements defects were found *and* significantly more use case and specification defects also identified, the checklist has been at least a partial success. This implies that the checklist needs to be tailored towards addressing requirements concerns, and aspects that address the use case and specification might be reduced or removed entirely.

4.4 Validity Threats

Construct validity primarily addresses the instrument (our checklist) whilst the other threats address concerns with the experiment.

Construct Validity. This threat considers whether the design of the instrument is flawed or that the outcomes were biased by the design of the instrument. There is one concern for this experiment: the instrument itself (the derivation of the checklist). The type of defect found might be as a consequence of the design of the checklist. It is suggested that this is not a threat because the number of requirements defects found by the treatment does not differ significantly from those found by the control group. Indeed, the treatment found significantly more specification defects than the control. Though the treatment also found significantly more use case (syntactic) defects than the control, this is appears to be somewhat symptomatic of use case checklists since there is an inevitable concern with comprehension of any natural language text. Whether a defect is a requirement problem, a specification problem or, indeed, just a use case problem can be dependent upon the problem being examined. However, it is certainly the case that aspects of Grammar, for instance, are hardly likely to cause noticeable problems in the real world. Missing information or unnecessary information will though.

Conclusion Validity. Conclusion validity considers whether the conclusions drawn from the statistical results are valid or are biased by the issues affecting the treatment and the outcome. The

random heterogeneity of subjects might affect the outcome. However, the large sample of subjects is drawn from the same community: Fourth year undergraduate students who have studied similar courses in their degrees.

Internal Validity. Internal validity is threatened by unknown influences on the causal relationship between the treatment and the outcome. If they are not accounted for they may invalidate the results. The History of the experiment is a potential threat since it spanned a number of days. However, there is no indication that subject results improved (the threat that subjects having taken part in the experiment might pass their knowledge gained onto those awaiting participation – there appears to be no diffusion of treatments). The experiment lasted the length of a normal tutorial: two hours. The subjects were used to working that length of time on individual and group tasks. The experiment followed that pattern. Thus maturation was not a problem. However, a few subjects noted that the second part of the experiment needed a little more time for fuller discussion. No subjects dropped out of the experiment once they had begun – mortality was not an issue.

External Validity. Since the experimental subjects are students, the results are potentially not generalisable to practitioners. The nature of the problem itself was not unknown to the subjects; a large number commented that they used their experience of ATMs to help in finding defects. The setting for the experiment might be considered a threat in that it is not in the confines of an industry location such as an office or meeting room. However, the locations themselves were familiar to the subjects since their tutorials are regularly at these locations and take a very similar format to that of the experiment. In any case, the classroom is a typical location for laboratory experiments of this nature.

5. CONCLUSIONS, IMPLICATIONS AND FUTURE WORK

This paper described an experiment that compared inspection techniques for detecting defects in use case descriptions. It is shown that the checklist found more defects than the ad hoc approach and that groups found more defects than individuals (though less than nominal groups). However, when the defect types are categorized, there was no significant difference between the control and treatment groups in the identification of requirements defects. The treatment found significantly more use case defects (typically syntactic in nature). It is also the case that the control found more requirements defects in the group task than the treatment.

The implications are that when one presents a checklist to be used in use case inspections, if it contains elements of a syntactic nature, these are the defects that will be discovered. Syntactic defects are easier to find than semantic defects. It is unsurprising then that the control group found more requirements defects (semantic) than use case (syntactic). (By semantics we do not mean the underlying rules that govern use cases, such as the work of the OMG (2003) but requirements –the real world effects and implications for the problem being addressed.) The control group was not guided in this direction and thus had to rely on the requirements document supplied and their own experiences. Qualitative feedback (from post-test questionnaires) suggests that the majority of control subjects did just this, whereas the majority of treatment subjects relied on the checklist and their experience to identify the defects, not upon the requirements document itself. The results of this experiment support the work of Travassos *et al* (1999) who stated that requirements-related inspections should focus on semantics and not syntax. We add a note of caution to this: a poorly written requirement or use case description is likely to be misunderstood or contain ambiguities and so there is a necessary relationship between semantics and syntax, or requirement and its

expression. As such, the checklist was not an entire failure. Its use helped identify significantly more use case and specification defects *and* similar numbers of requirements defects to the control group.

Nonetheless, it is more important to find and resolve requirements problems than syntax problems since their impact of the success of the project is vastly greater. As such, our intention is to pursue a course of research to propose an entirely requirements-focussed use case checklist and will conduct further empirical studies to assess its efficacy when compared to other more typical checklists, such as the ones presented in this paper. This is on-going research and we will present it in the future.

REFERENCES

ACHOUR, C., ROLLAND, C., MAIDEN, N. and SOUVEYET, C., (1999): Guiding use case authoring: results from an empirical study, *4th IEEE International Symposium on Requirements Engineering*, Limerick, 7–11 June.

ADOLPH, S., BRAMBLE, P., COCKBURN, A. and POLS. (2003): *Patterns for Effective Use Cases*, Addison-Wesley.

ALEXANDER, I. (2003): Misuse cases: Use cases with hostile intent, *IEEE Software*, Jan/Feb, 58–66.

ANDA, B. and SJOBERG, D. (2002): Towards an inspection technique for use case models, *14th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'02)*, Ischia, Italy, 15–19 July, 127–134.

ANDA, B., SJOBERG, D. and JORGENSEN, M. (2001): Quality and understanding of use case models, In: LINDSKOV KNUDSEN, J. (editor), *15th European Conference on Object-Oriented Programming*, Budapest, June 18–22, Berlin, LNCS, Springer-Verlag, 402–428.

BIFFL, S. and HALLING, M. (2003): Investigating the defect detection effectiveness and cost benefit of nominal inspection teams, *IEEE Trans. Software Engineering*, 29 (5): 385–397.

BITTNER, K. and SPENCE, I. (2002): *Use Case Modelling*, Addison-Wesley.

BRAY, I. (2002): *An Introduction to Requirements Engineering*, Addison-Wesley.

BRYKCZYNSKI, B. (1999): A survey of software inspection checklists, *ACM Sigsoft: Software Engineering Notes*, 24(1): 82–89.

COCKBURN, A. (2001): *Writing Effective Use Cases*, Addison-Wesley.

COX, K. (2002): *Heuristics for Use Case Descriptions*, PhD Thesis, Bournemouth University, UK, November.

COX, K., AURUM, A. and JEFFERY, R., (2004): A use case description inspection experiment, School of Computer Science and Engineering, University of New South Wales, Technical Report: UNSW-CSE-TR-0414, ftp://ftp.cse.unsw.edu.au/pub/doc/papers/UNSW/0414.pdf.

COX, K. and PHALP, K. (2000): Replicating the CREWS use case authoring guidelines experiment, *Empirical Software Engineering Journal*, 5(3): 245–268.

COX, K., PHALP, K. and SHEPPERD, M. (2001): Comparing use case writing guidelines, In: ACHOUR-SALINESI, C., OPDAHL, A., POHL, K. and ROSSI, M. (editors), *7th International Workshop on Requirements Engineering: Foundation for Software Quality*, Interlaken, Switzerland, 4–5 June, Essen, Essener Informatik Beitrage, 101–112.

DUNSMORE, A., ROPER, M. and WOOD, M. (2003): Practical code inspection techniques for object-oriented systems: An experimental comparison, *IEEE Software*, July/August, 21–29.

FAGAN, M. (1976): Design and code inspections to reduce errors in program development, *IBM Systems Journal*, 15(3): 182–211.

FUSARO, P., LANUBILE F. and VISAGGIO, G. (1997): A replicated experiment to assess requirements inspection techniques, *Empirical Software Engineering Journal*, 2: 39–57.

GAUSE, D. and WEINBERG, G. (1989): *Exploring Requirements: Quality Before Design*, Dorset House.

GARNHAM, A. and OAKHILL, J. (1996): The mental models theory language of comprehension, In: BRITTON, B. and GRAESSER, A. (editors), *Models of Understanding Text*, Mahwah, NJ, Lawrence Erlbaum Associates, 313–339.

GERNSBACHER, M. (1996): The structure-building framework: what it is, what it might also be and why, In: BRITTON, B. and GRAESSER, A. (editors), *Models of Understanding Text*, Mahwah, NJ, Lawrence Erlbaum Associates, 289–311.

GERNSBACHER, M. (1997): Two decades of structure building, *Discourse Processes*, 23(3): 265–304.

GOLDMAN, S., VARMA, S. and COTE, N. (1996): Extending capacity-constrained construction integration: toward 'smarter' and flexible models of text comprehension, In: BRITTON, B. and GRAESSER, A. (editors), *Models of Understanding Text*, Mahwah, NJ, Lawrence Erlbaum Associates, 73–113.

GRAESSER, A., SWAMER, S., BAGGETT, W. and SELL, M. (1996): New models of deep comprehension, In: BRITTON, B. and GRAESSER, A. (editors), *Models of Understanding Text*, Mahwah, NJ, Lawrence Erlbaum Associates, 1–32.

GRAHAM, I. (1998): *Requirements Engineering and Rapid Development*, Harlow, Addison-Wesley.

HALLING, M. BIFFL, S., GRECHENIG T. and KOEHLE, M. (2001): Using reading techniques to focus inspection performance, *IEEE Proceedings of Euromicro Conf.*, 248–257.

JACKSON, M. (1995): *Software Requirements and Specifications*, Addison-Wesley.
JACOBSON, I., CHRISTERSON, M., JONSSON, P. and OVERGAARD, G, (1992): *Object-Oriented Software Engineering: A Use Case Driven Approach*, Wokingham, Addison-Wesley.
KAMPHAN, P. (2003): Use case specification checklist, http://cmvl.cs.concordia.ca/courses/soen-341/winter-2003/use_case_checklist.pdf
KLARITI (2003): Inspection checklist for use case documents, www.klariti.com/templates/use-case-checklist.shtml
KULAK, D. and GUINEY, E. (2000): *Use Cases – Requirements in Context*, Harlow, Addison-Wesley.
LAUESEN, S. (2002): *Software Requirements: Styles and Techniques*, Harlow, Addison-Wesley.
LAUESEN, S. and VINTER, O. (2001): Preventing requirements defects: An experiment in process improvement, *Requirements Engineering Journal*, 6: 37–50.
MCBREEN, P. (2001): http://www.mcbreen.ab.ca/papers/QAUseCases.html
MILLER, J., WOOD, M. and ROPER, M. (1998): Further experiences with scenarios and checklists, *Empirical Software Engineering*, 3: 37–64.
NEILL, C. and LAPLANTE, P. (2003): Requirements engineering: the state of the practice, *IEEE Software*, Nov/Dec, 20(6): 40–45.
OMG – OBJECT MANAGEMENT GROUP (2003). *Unified Modeling Language Notation Guide v1.5*, Document 03-03-10. http://www.omg.org/
PARNAS, D. and LAWFORD, M. (2003): Inspection's role in software quality assurance – Guest Editorial, *IEEE Software*, July/August, 16–20.
PHALP K. and COX, K. (2002): Supporting communicability with use case guidelines: An empirical study, *EASE 2002, 6th International Conference on Empirical Assessment and Evaluation in Software Engineering*, Keele University, Staffordshire, UK, April 8–10.
POOLEY, R. and STEVENS, P. (1999): Using *UML – Software Engineering with Objects and Components*, Harlow, Addison-Wesley.
PORTER, A., VOTTA, L. and BASILI, V. (1995): Comparing detection methods for software requirements inspections: A replicated experiment, *IEEE Transactions on Software Engineering*, 21(6): 563–575.
REGNELL, B., RUNESON, P. and THELIN, T. (2000): Are perspectives really different? – Further experiences on scenario-based reading of requirements, *Empirical Software Engineering Journal*, 5(4): 331–356.
ROMBACH, C. D., KUDE, O., AURUM, A., JEFFERY, R. and WOHLIN, C. (2003): An empirical study of an ER-model inspection meeting, *29th Euromicro Conference*, Antalya, Turkey, 3–5 September, 308–315.
SANDAHL, K., BLOMKVIST, O., KARLSSON, J., KRYSANDER, C., LINDVALL M. and OHLSSON, N. (1998): An extended replication of an experiment for assessing methods for software requirements inspections, *Empirical Software Engineering*, 3: 327–354
SCHNEIDER, G. and WINTERS, J. (1998): *Applying Use Cases: A Practical Guide*, Harlow, Addison-Wesley.
TERVONEN I. (2003): http://tol.oulu.fi/~tervo/Checklist.pdf, accessed 28/7/2003.
THELIN, T., RUNESON, P. and WOHLIN, C. (2003): Prioritized use cases as a vehicle for software inspection, *IEEE Software*, July/August, 30–33.
TRABASSO, T., VAN DEN BROEK, P. and SUH, S. (1989): Logical necessity and transitivity in causal relations in stories, *Discourse Processes*, 12: 1–25.
TRAVASSOS, G. H., SHULL, F., FREDERICKS, M. and BASILI, V. (1999): Detecting defects in object oriented designs: Using reading techniques to increase software quality, *Proceedings of the 14th ACM conference Object-Oriented Programming Systems Languages and Applications - OOPSLA'99*, Denver, Colorado, USA, 1-5 November, 47–56.
VOTTA, L. (1993): Does every inspection need a meeting?, *Proceedings of the ACM SIGSOFT 1993 Symposium on foundations of software engineering*, 18 December, 107–114.
WIEGERS, K. (2003): Inspection checklist for use case documents, http://www.processimpact.com/process_assets/use_case_checklist.doc, Process Impact
WOHLIN, C. and AURUM, A. (2003): Evaluating cost-effectiveness of checklist-based reading of entity-relationship diagrams, *Proceeding of 9th International Software Metrics Symposium, METRICS'03'*. Sydney, Australia, 3–5 September, 286–296.

BIOGRAPHICAL NOTES

Karl Cox

Dr Karl Cox is a research fellow at the University of New South Wales in the School of Computer Science and Engineering, and an affiliated researcher with the Empirical Software Engineering program at National ICT Australia (NICTA). He received his PhD in Computer Science at Bournemouth University, UK, in 2002, where he was a member of the Empirical Software Engineering Research Group. Prior to this, his background was in history and English language, which he taught in Bognor Regis, Budapest and Madrid. His main research interests are in requirements engineering for business needs, and specifically in problem frames as a potential paradigm to address those needs.

Aybüke Aurum

Dr Aybüke Aurum is a senior lecturer at the School of Information Systems, Technology and Management, University of New South Wales. She received her BSc and MSc in geological engineering, and MEngSc and PhD in computer science. She is the founder and group leader of the Requirements Engineering Research Group (ReqEng) at the University of New South Wales. She also works as a visiting researcher in National ICT, Australia (NICTA). Her research interests include management of software development process, software inspection, requirements engineering, decision making and knowledge management.

Ross Jeffery

Dr Ross Jeffery is Professor of Software Engineering in the School of Computer Science and Engineering at UNSW and Program Leader in Empirical Software Engineering in National ICT Australia Ltd. (NICTA). His current research interests are in software engineering process and product modelling and improvement, electronic process guides and software knowledge management, software quality, software metrics, software technical and management reviews, and software resource modeling and estimation. His research has involved over fifty government and industry organizations over a period of 15 years and has been funded from industry, government and universities. He has co-authored four books and over one hundred and twenty research papers. He has served on the editorial board of the IEEE Transactions on Software Engineering, and the Wiley International Series in Information Systems and he is associate editor of the Journal of Empirical Software Engineering. He is a founding member of the International Software Engineering Research Network (ISERN). He was elected Fellow of the Australian Computer Society for his contribution to software engineering research.