A Use Case Description Inspection Experiment

Karl Cox[1], Aybüke Aurum[2], Ross Jeffery[1]

[1]School of Computer Science and Engineering
[2]School of Information Systems, Technology and Management
University of New South Wales, National ICT Australia
Sydney, Australia
aybuke@unsw.edu.au; karlc, rossj@cse.unsw.edu.au

UNSW-CSE-TR-0414

THE UNIVERSITY OF
NEW SOUTH WALES

The School of Computer Science & Engineering
The University of New South Wales
UNSW SYDNEY 2052

Abstract

Achieving higher quality software is one of the aims sought by development organizations worldwide. Establishing defect free statements of requirements is a necessary prerequisite to this goal. 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. 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.

1 Introduction

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. The industry wide uptake of use cases [1] means they should also be unambiguous, correct, consistent and complete. But because of the informal nature of use case descriptions, it is easy to introduce ambiguity [2]. Also, because one use case can fulfill several requirements and one requirement can be described by several use cases, it is often difficult to achieve completeness and consistency.

Defect detection in requirements documents is seen as one of the most effective and efficient quality assurance techniques in software engineering [e.g. 3, 4]. Such defect detection should also be applied to use cases. However, there has been relatively little literature specifically on use case inspections. Bittner and Spence [5], for example, provide a chapter on conducting reviews but they do not suggest more than general advice on how to do inspections, not particularly what to look for. Adolph et al. [6] present a two-tiered review pattern. Its first review attempts to eliminate "'noise' caused by spelling, grammatical, formatting and technical errors" (p.65). The second review 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 [7] provides a pass/fail test for use case descriptions that addresses whether the description meets its goals, is really required and is feasible.

Anda and Sjoberg [8] present a use case inspection experiment. Results showed that experienced inspectors found more defects without the checklist; but students found more defects with the checklist. Different stakeholders also found different types of defects. Since Anda and Sjoberg address the use case model, their study differs to ours; we focus only on the description. Their results show that practitioners were interested in defects relating to actors, triggers and pre- and post-conditions. They did not find many defects in the descriptions.

Despite the lack of interest in inspections in the use case literature, it is clear that inspections are very valuable. Numerous approaches to inspections have been suggested. Due to word restrictions we refer the interested reader to Aurum et al. [9].

Requirements inspections are different from code inspections. Travassos et al. [10] suggest that requirements inspection should focus on semantics and less on syntax. However, in a large experiment, Rombach et al. [11] found the proportion of syntactic and semantic defects identified did not significantly differ.

The literature indicates that conducting inspections is important for requirements documents [4, 10, 12] and there is an interest in using use cases in conducting inspections of documents e.g. [13, 14] but the assumption is that the use cases have already been validated. There, therefore, appears to be a lack of inspection of use case descriptions themselves, though some checklists have been suggested (section 2.1). This paper therefore presents, compares and tests a use case description checklist employed in a formal two-stage inspection against an ad hoc approach by means of an experiment.

The next section discusses and compares use case checklists available in the literature. Section three presents the experimental design. Section four gives the results from the experiment. Section 5 discusses conclusions and future work.

2 Use Case Description Checklists

Our use case checklist is presented in Table 1. The checklist acts as the instrument in the experiment. Our checklist is only about the flows of events in a use case description. It thus differs from other use case checklists in that it does not address elements of the use case template in detail or the diagram at all. The derivation of the checklist is adapted from a means of assessing the understandability of use case descriptions [e.g. 15, 16, 17].

2.1 Comparing Use Case Description Checklists

Table 2 compares the Use Case Checklists available [7, 8, 18, 19, 20, 21, 22] in terms of how they refer to the Description only. The top row in the table shows how the checklists compare to the elements in ours (Table 1). The last column in Table 2, 'Other', is for those aspects that are in other checklists that we do not consider.

Table 1. Use Case Description Checklist

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?

Coverage: all authors ask that the description has enough information to ensure completeness (Span). Only Wiegers [18], Kamphan [20] and Klariti [21] consider if there is too much information (Scope) [23], asking whether the whole use case is a discrete task. Kamphan goes further to our depth: are all the events necessary? Tervonen [22] asks that only a typical way of using the system be described. Cockburn's understanding of Scope is more concerned with abstraction [7, p.212].

Aspects of Cogent are considered by all. However, it is only Anda and Sjoberg [8] that address all three attributes. All others address Rational Answer. Cockburn and Kamphan also

consider Dependencies. Tervonen looks at the overall use case document, asking that it have a proper structure of a beginning, middle and end.

Compared to ours, no checklists address the Structure facet stating that variations should be kept out of the main flow, except McBreen [19]. To allow alternative paths as part of the main flow of events will add complexity and increase the risk of misunderstanding. Cockburn's pass/fail test is unclear about variations in the main flow but he states there should be a separate section elsewhere in his book (p.206).

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 [7, 19].

Though it can be seen that all checklists address Language, most are ambiguous as to what this might be. Tervonen states only that the use cases should be "clear". McBreen suggests the use of active verb phrases. Cockburn asks that it should be clear who is responsible for actions (active voice) and elsewhere suggests the structure "subject verb direct object prepositional phrase" (p.90), but not as part of the pass/fail test.

Table 2. Comparison of Checklists that Address the Description

Checklist	Coverage	Cogent	Structure	Abstraction	Language	Alternatives	Other
Anda and Sjoberg [8]	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 [18]	Span – complete events; Scope – discrete task	Rational – Feasible	No.	Essential level only, no design or implementa-tion	Clearly written, unambiguous	Viable – Feasible and Verifiable	Measurable result; Pre- and post-conditions
McBreen [19]	Span – are all failure conditions considered?	Rational – does the UC do what is expected?	Variations - All recoverable failures recorded as Extensions	Precision of detail consistent – same conceptual level (e.g. internal design / external design only)	Style should be consistent. Active verb phrase.	Viable – Recoverable failures recorded	Goal success / failure. Presentation, formatting consistent.
Kamphan [20]	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 implementa-tion. 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 [21]	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 implementa-tion)	Clearly written, unambiguous and complete.	Viable – Is each course feasible?	Verifiable?
Tervonen [22]	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 [7] – pass/fail test	Scope – is everything in scope as a black box or internal scenario? – (Consistent Abstraction)	Dependencies – Does UC run to success?	Variations – Keep these out of main flow p.206.	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

Almost all authors suggest the Alternatives / Exceptions should be feasible or at least required; except McBreen, who notes that all recoverable failures should be recorded, and Tervonen, who does not address this facet.

2.2 Types of Defects

To quantify the differences in our checklist (Table 1) in terms of their considered ease of detection, we decompose the list into three components in terms of their impact upon the problem: No Impact (internal to the use case); Specification Impact (though the use case is still expected to meet the requirement); and Requirements Impact (the effects desired in the problem domain are not achieved – this is the most severe). *Use Case Description impact.* Language is important for readability. Structure: Sequence refers to correct numbering of events as does Numbering in Consideration of Alternatives.

Specification impact. Abstraction is concerned with the risk of mixing internal design detail into the specification task. Structure: Variations might affect the specification by assuming transitive dependencies.

Requirements impact. Coverage: Scope and Span consider completeness. Cogent: Text Order, wrong ordering could affect the requirement. Cogent: Dependencies does a use case terminate as expected? Cogent: Rational Answer, does a step / procedure make sense, is it implementable? [6]. Alternatives: Viable explores whether the alternative / exception solution is feasible and complete.

3 Experimental Design

We conducted an experiment that took the form of a two-stage review process, with the treatment being the checklist of Table 1. The control group applied an ad hoc approach where they were provided with no guidance except requirements elicitation notes and the use case model. The first stage was an individual inspection (55 minutes, plus 5 to complete a questionnaire). Subjects were given a ten-minute break and then conducted the second stage, a group inspection (30 minutes allowed, plus 5 to complete a questionnaire) – see Table 3.

Table 3. Experimental design, subject numbers and time scales

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 (22 groups)	30	Group Inspection (27 groups)	30
Individual Questionnaire (on group inspection activity)	5	Individual Questionnaire (on group inspection activity)	5

3.1 Experimental Subjects

There are two groups, group A (control), 73 subjects and group B (treatment), 79 subjects (Table 3). They are final year undergraduate computer and software engineering students taking a course in Total Quality Management (TQM), of which software inspection forms a normal part. The subjects were taught about use cases as a normal part of their course and many are experienced users. The experiment was conducted in the normal tutorials of the TQM course over a period of four days. Nine tutorial groups were randomly assigned to either the control or the treatment with the only proviso a relatively equal number of subjects in each experimental group.

3.2 Experimental Tasks

The subjects were presented elicitation notes and a use case model of a simplified ATM. 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. All experimental material and data can be found in the appendix.

3.3 Hypotheses

There are six null hypotheses:
$H1_0$: Use of the checklist will find similar numbers of Use Case defects to an ad hoc approach in the Individual inspection task.
$H2_0$: Use of the checklist will find similar numbers of Use Case defects to an ad hoc approach in the Group inspection task.
$H3_0$: Use of the checklist will find similar numbers of Specification defects to an ad hoc approach in the Individual inspection task.
$H4_0$: Use of the checklist will find similar numbers of Specification defects to an ad hoc approach in the Group inspection task.

$H5_0$: Use of the checklist will find similar numbers of Requirements defects to an ad hoc approach in the Individual inspection task.
$H6_0$: 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 [8].

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 4 shows that the ad hoc approach (A) identifies few Use Case defects for individual and group inspections: 23 and 36 percent respectively. 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 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.

Table 4. Breakdown of totals, means, percentages and standard deviations for defect identified

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

The standard deviations (SD) show a similar variance for both groups. Individually, both groups A and B are similar. However, B has almost half the deviation from the mean for specification defects. As a much larger number of these defects were found, this is unsurprising. Group A has

more deviation for Use Case and Requirements defects than group B. The deviations from Individuals to Groups are also similar. Independent sample two-tailed t-tests were applied (alpha = 0.05), as shown in Table 5, to test the hypotheses (section 3.3).

Table 5. Significance values for Defect Types

Data Tested			Significance
Individual	H1	A, B Use Case	p <= 0.0001
	H3	A, B Specification	p <= 0.0001
	H5	A, B Requirement	p = 0.64
Group	H2	A, B Use Case	p = 0.0003
	H4	A, B Specification	p <= 0.0001
	H6	A, B Requirement	p = 0.22

The Null hypotheses are rejected for H1-H4. The Treatment group (B) found significantly more defects that the Control (A). However, for Requirements defects (H5-H6), the Null is accepted: there is no significant difference in number of defects found.

It is interesting to explore the distributions for the Requirements defects to assess whether any skewness has affected the outcome of the tests. Figure 1 (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 comparing the number of requirements defects found in the group task, Figure 1 (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 one outlier of two. Its removal does not alter the significance and thus is included.

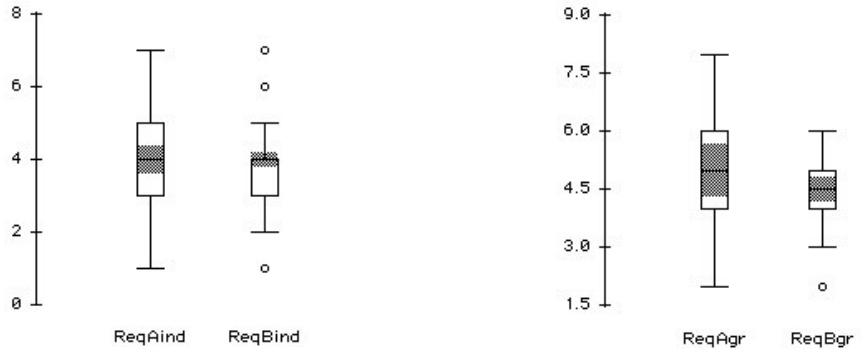


Fig. 1. Boxplots for Individual (left) and Group (right) Requirements Defects for A and B

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. The counterpoint 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 should be reduced so that we can focus on semantic rather than syntactic errors [10].

4.1 Validity Threats

Construct Validity. 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.

Conclusion Validity. The random heterogeneity of subjects might affect the outcome. However, the large sample of subjects is drawn from the same community: 4th year undergraduate students who have studied similar courses in their degrees.

Internal Validity. 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 was no diffusion of treatments). The experiment lasted the length of a normal tutorial: two hours. Thus the subjects were used to working that length of time on individual and group tasks. The experiment followed that pattern. Therefore, 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. 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 location was familiar to the subjects since the experiment took place in normal tutorials.

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 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) and specification defects. It is also the case that the control found more requirements defects in the group task than the treatment (54 percent of defects to 49, respectively).

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). They were not guided in this direction and thus had to rely on the requirements document supplied and their own experiences. Qualitative feedback (from post-experiment questionnaires) suggests that the majority of control subjects did just this (45 and 24 subjects respectively from 67 who responded), whereas the majority of treatment subjects relied on the checklist (46 subjects from 77 respondents) and only 33 subjects on the requirements, with 23 relying on their experience to identify the defects. The results of this experiment support the work of Travassos et al. [10] 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 on the success of the project is vastly greater. As such, we are pursuing a course of research to propose an entirely requirements-focussed use case checklist and will conduct further experiments to assess its efficacy when compared to other more typical checklists, such as the ones discussed in this paper.

References

1. Jacobson, I., Christerson, M., Jonsson, P., Overgaard, G.: Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley (1992)
2. Gause, D., Weinberg, G.: Exploring Requirements: Quality Before Design. Dorset House (1989)
3. Porter, A., Votta, L., Basili, V.: Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. IEEE Trans. Sw Eng. **21** 6 (1995) 563-575
4. Parnas, D., Lawford, M.: Inspection's Role in Software Quality Assurance – Guest Editorial. IEEE Software. July/August (2003) 16-20
5. Bittner, K., Spence, I.: Use Case Modelling. Addison-Wesley (2002)
6. Adolph, S., Bramble, P., Cockburn, A., Pols.: Patterns for Effective Use Cases. Addison-Wesley (2003)
7. Cockburn, A.: Writing Effective Use Cases. Addison-Wesley, (2001)
8. Anda, B., Sjoberg, D.: Towards an Inspection Technique for Use Case Models. 14th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'02), Ischia, Italy, 15-19 July, (2002) 127-134
9. Aurum, A., Petersson, H., Wohlin, C.: State-of-the-art: Software Inspections After 25 Years. Software Testing Verification Reliability J. **12** 3 (2002) pp133-154
10. Travassos, G., Shull, F., Fredericks, M., Basili, V.: Detecting Defects in Object Oriented Designs: Using Reading Techniques to Increase Software Quality. Proceedings of OOPSLA'99. Denver, Colorado, November (1999)

11. Rombach, C., Kude, O., Aurum, A., Jeffery, R., Wohlin, C.: An Empirical Study of an ER-Model Inspection Meeting. Accepted for, 29th Euromicro Conference. Antalya Turkey, 3-5 September (2003)
12. Wohlin, C., Aurum, A.: An Evaluation of Checklist-Based Reading for Entity-Relationship Diagrams. 9th Int. Software Metrics Symposium (Metrics 2003), Sydney, Australia, 3-5 September (2003) pp286-296
13. Thelin, T., Runeson, P., Wohlin, C.: Prioritized Use Cases as a Vehicle for Software Inspection. IEEE Software. July/August (2003) 30-33
14. Dunsmore, A., Roper, M., Wood, M.: Practical Code Inspection Techniques for OO Systems: An Experimental Comparison. IEEE Software. July/August (2003) 21-29
15. Cox, K., Phalp, K.: Replicating the CREWS Use Case Authoring Guidelines Experiment. Empirical Software Engineering J. **5** 3 (2000) 245-268
16. Cox, K., Phalp, K., Shepperd, M.: Comparing use case writing guidelines. In: Achour-Salinesi, C., Opdahl, A., Pohl, K., Rossi, M. (eds), 7th International Workshop on Requirements Engineering: Foundation for Software Quality. Interlaken, Switzerland, 4-5 June. Essen, Essener Informatik Beitrage (2001) pp101-112
17. Cox, K.: Heuristics for Use Case Descriptions - PhD Thesis. Bournemouth University, UK, (2002)
18. Wiegers, K.: Inspection Checklist for Use Case Documents. http://www.processimpact.com/process_assets/use_case_checklist.doc. Process Impact, (2003)
19. McBreen, P.: http://www.mcbreen.ab.ca/papers/QAUseCases.html. (2001)
20. Kamthan, P.: Use Case Specification Checklist. http://cmvl.cs.concordia.ca/courses/soen-341/winter-2003/use_case_checklist.pdf. (2003)
21. Klariti.: Inspection Checklist for Use Case Documents, www.klariti.com/templates/use-case-checklist.shtml. (2003)
22. Ternvonen I.: http://tol.oulu.fi/~tervo/Checklist.pdf, accessed 28/7 (2003)
23. Jackson, M.: Software Requirements and Specifications. Addison-Wesley, (1995)

Appendix – Experimental Material and Data

Elicitation Notes

Notes taken during initial requirements workshop meeting 27/3/2003, 3pm, with Bank team to discuss the proposed Automated Teller Machine (ATM).

Stakeholders present:

Development team: Systems analyst; project manager; requirements engineer (acting as scribe).

Bank team: banking consultant, bank marketing representative, bank finance representative.

Project Manager: Can you give us an outline of what services the ATM should provide?

Bank Marketing: Sure. We'd like the ATM to be a modern service provider. We'd *only* like to offer the most popular services. We've found that Users typically only need three or four functions when using our ATMs.

Banking Consultant: We've found that Users really only want to withdraw cash. We've tried to simplify how Users can do this but of course we still need to follow some security procedures. They'll have to enter their PIN first of course and then select the account they are going to use.

Project Manager: How do you want that represented on the screen?

Systems Analyst: Let's not worry about that for now. So the User sticks their card into the ATM, keys in their PIN and chooses their account. That's pretty standard. We have to legally provide this functionality anyway, so what else does the User want to be able to do?

Bank Finance: Well, we'd like to provide these services: withdraw cash but not with a receipt. Typically Users just throw these away or forget to take them. So if they want to know what they've just withdrawn they'll have to ask someone inside the bank or wait for their monthly statement. We'd also like these functions: we'd like Users to be able to check their account balance; they should be able to change their card PIN if they like and we've found that Users like to make deposits – it avoids queuing in the bank – so they should be able to do that from an ATM. That's it, I think.

Project manager: So let's get this right. You want these services: withdraw cash – without a receipt option; choose to view the User's account balance; change the Personal Identification Number, the PIN; and make deposits. Would that be cash or cheque or something else or all of the above?

Bank consultant: We'd like just cash (notes) or cheques. We'd expect the User to put the deposit into an envelope, fill out on the envelope the details of their deposit – if it's cash and then how much and what denomination: 20s, 10s, 50s etc. Then they'd stuff the envelope in a deposit slot. Of course, the User would have to put their PIN in first and choose to make a deposit. Then when they've made the deposit they should get a receipt that says their account number and that a deposit has been received.

Systems analyst: Do you want them to enter the amount they've deposited so that gets printed on the receipt?

Bank marketing: We thought about that and we think it is more secure if we don't do that. If they get a receipt saying they've deposited $500 but we only found $200 in the envelope, we'd be liable to pay the difference.

Project manager: So let's get this all down. The functional requirements are this: 1. The User can withdraw cash. One of the requirements is that the ATM checks the amount the User has in their account before they are allowed to withdraw anything?

Systems Analyst: That's a design issue.

Bank marketing: Well, we'd really like to be certain that Users can't take out more than their allowed. Whether the check happens there or when the User's card and PIN are checked, it doesn't matter for now. So long as the check is made to the Bank before they take anything out that's fine. We'll discuss this later in the project in more detail. Let's not worry about it for now. When the User chooses the Change PIN option, we'd expect them to have to re-enter their PIN –

Project manager: Why do we have to always do that?

Bank consultant: Standard security measure.

Bank marketing: Yes. The User would then have to enter their new PIN twice, each time it being acknowledged by the ATM (and made obvious to the User).

Systems Analyst: Does the User get the option of how their balance can be represented?

Bank marketing: On screen or on paper.

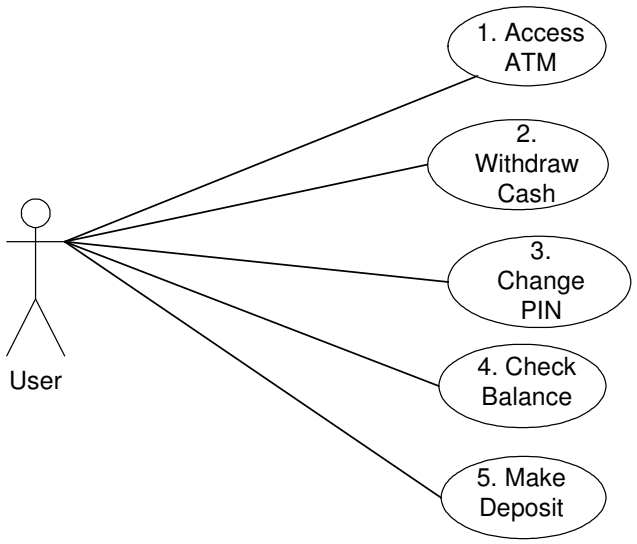
Project manager: OK. So the chief functionality should be: 1. Withdraw cash (as said); 2. Change the card PIN; 3. Make a deposit and 4. Check the balance. Is that right?

Bank consultant: Yes. Don't forget the access requirement – the PIN check.

Project manager: That's right. Accessing the ATM in the first place. The User can't do anything with an invalid card or PIN. OK. Let's get working on these and meet in a couple of days. Is Thurday 29th OK with you? Same time 3pm, same place.

<Meeting ended>

ATM Use Cases



Use Case Diagram for ATM

Problem Statement

A supplier of ATMs is to produce a new cash point for a major bank. A requirements analysis has revealed a number of important functions. As well as withdrawing cash, a customer can check their balance, make a deposit and change their PIN number.

Actor Specification

User

The User has a bank account at the Bank. The User has a valid bankers, credit or debit card. The User should have sufficient funds in their account (though this will have to be checked before each withdrawal transaction is allowed).

Use Case Descriptions (with defects identified)

The following descriptions detail the perceived usages of the new ATM based on the use case diagram above.

Use Case 1. 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.
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 ATM; Grammar: his: remove **UC***)
4. ATM asks for account type. (*Coverage : Scope events 4 and 5 should be in other UCs, not here **REQ**)*
5. **Customer** selects account. (*Consistent Grammar : synonym: User **UC***)
6. ATM displays User options.

Exceptional flow of events:

4a. ATM rejects unidentifiable card. (*Consideration of Alternatives : numbering: 2e **UC***)

Post-condition: 1. User access granted 2. PIN and card validated.

Use Case 2. WITHDRAW CASH

Actors: User, Bank
Context: User wants to withdraw money.
Pre-condition: User has already logged onto the ATM.

Main Flow of Events:

1. User selects 'Withdraw Cash'.
2. ATM prompts for amount.
3. User enters amount.
4. ATM verifies with the Bank that the User has enough money in account. (*Consistent Abstraction: this is internal design and not necessary **SPEC***)
   4.1 If insufficient funds in **her** account, (*Consistent Structure : Variations: **SPEC** events 4.1 – 4.3 should be in a separate Exceptional flow section; Grammar : pronoun: User **UC***)
   4.2 ATM returns card to User.
   4.3 User takes card. <end use case>
5. ATM releases cash. (*Cogent : Text Order : 7, 8, 5, 6 **REQ**)*
6. User takes cash.
7. ATM releases card.

8. User takes card.

Alternative flows of events:

7a. ATM eats card. (*Consideration of Alternatives : viable: why would the ATM eat the card here? **REQ***)

Post-condition: ATM ready for next User.

Use Case 3. 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***)
3. It enters new PIN. (*Grammar: pronoun; User **UC***)
4. ATM prompts User to re-enter new PIN.
5. User re-enters new PIN.
6. ATM displays 'New PIN Successful' message.
7. ATM displays list of options.

Exceptional flow of events:
4e ATM refuses new PIN.
4.1e User asked to re-enter new PIN. (*Cogent : Dependency : loop **REQ**; Consideration of Alts : Numbering: 5.1e **UC***)

(Span : need to do this exception again for second PIN entry REQ)

Post-conditions: New PIN read to card and Bank account

Use Case 4. CHECK BALANCE

Actors: User
Context: The User wants to check their account balance before withdrawing money.
Pre-condition: User already logged onto the ATM

Main flow of events:

1. User selects balance of account.
2. User selects On Screen option.
3. ATM displays current balance on screen.
4. **Bank** retrieves User's current balance from their account. (*Consistent Abstraction **SPEC** – all internal design; Cogent: Text Order: 3 and 4 **REQ***)
5. ATM prompts for new option.

Alternative flow of events:
2a. User selects On Paper option.
    2.1a ATM prints balance on receipt
    2.2a User takes receipt
    <end of use case>

Post-condition: 1. Balance no longer displayed; 2. ATM ready for a transaction.

Use Case 5. MAKE DEPOSIT

Actor: User
Context: The User wants to deposit cash into the ATM
Pre-condition: The User has logged onto the ATM

Main flow of events:

1. User selects Deposit.
2. Selects envelope (*Coverage : Span; no subject **REQ***)
3. ATM accepts deposit
4. User takes deposit receipt. (*Coverage : Span; this should be event 6 and event 5 should say that ATM prints or produces receipt **REQ***)

Exceptional flow of events:
3e. ATM rejects deposit envelope.
    3.1e ATM signals User of rejection.

Post-condition: ATM ready for a new transaction.

DATA

Individual Answers

Control Individual (group A)					Treatment Individual (group B)				
Subject	Total	Use Case	Specification	Requirement	Subject	Total	Use Case	Specification	Requirement
1	1	0	0	1	1	5	1	1	3
2	7	3	1	3	2	4	0	1	3
3	7	1	0	6	3	5	0	1	4
4	4	0	0	4	4	5	1	1	3
5	8	3	0	5	5	5	1	1	3
6	4	1	0	3	6	6	1	1	4
7	7	3	0	4	7	9	4	1	4
8	6	1	0	5	8	8	6	1	1
9	3	0	0	3	9	8	2	1	5
10	6	3	0	3	10	8	4	2	2
11	3	0	0	3	11	8	3	1	4
12	8	5	0	3	12	6	2	1	3
13	3	1	0	2	13	6	3	0	3
14	8	3	0	5	14	5	3	0	2
15	2	0	0	2	15	6	1	1	4
16	4	1	0	3	16	5	2	1	2
17	8	5	1	2	17	9	4	1	4
18	5	3	0	2	18	7	3	1	3
19	6	1	0	5	19	8	1	1	6
20	10	6	0	4	20	8	4	0	4
21	5	1	0	4	21	6	2	0	4
22	4	0	0	4	22	9	4	0	5
23	2	0	0	2	23	8	2	2	4
24	5	0	0	5	24	8	3	1	4
25	6	1	0	5	25	3	2	0	1
26	4	2	0	2	26	6	2	1	3
27	4	0	0	4	27	12	7	0	5
28	6	2	1	3	28	12	5	1	6
29	1	0	0	1	29	10	4	1	5
30	2	1	0	1	30	8	4	1	3
31	2	0	0	2	31	11	6	0	5
32	6	1	0	5	32	11	4	2	5
33	3	0	0	3	33	9	5	1	3

34	8	5	0	3	34	5	3	1	1
35	6	1	0	5	35	2	1	0	1
36	7	2	0	5	36	5	1	1	3
37	11	4	1	6	37	3	0	1	2
38	12	6	0	6	38	11	5	2	4
39	3	0	0	3	39	6	4	0	2
40	6	1	0	5	40	7	2	1	4
41	8	1	0	7	41	11	5	0	6
42	7	3	0	4	42	10	2	2	6
43	7	1	0	6	43	11	5	2	4
44	4	1	1	2	44	10	3	2	5
45	4	0	0	4	45	10	5	1	4
46	5	0	0	5	46	3	2	0	1
47	4	1	0	3	47	8	4	1	3
48	3	0	1	2	48	9	3	3	3
49	4	0	0	4	49	3	2	0	1
50	4	0	0	4	50	9	4	1	4
51	11	4	0	7	51	7	2	1	4
52	5	0	0	5	52	5	3	0	2
53	8	3	0	5	53	4	1	0	3
54	4	1	0	3	54	10	5	1	4
55	3	0	1	2	55	10	3	1	6
56	8	4	0	4	56	10	4	2	4
57	3	0	0	3	57	8	3	1	4
58	7	2	1	4	58	13	5	2	6
58	7	1	0	6	58	5	2	1	2
60	5	3	0	2	60	7	2	1	4
61	7	2	0	5	61	6	2	1	3
62	12	4	1	7	62	9	4	1	4
63	8	5	0	3	63	9	4	1	4
64	7	3	0	4	64	10	4	2	4
65	3	1	0	2	65	8	4	0	4
66	8	4	1	3	66	9	2	2	5
67	5	1	0	4	67	12	5	2	5
68	4	0	0	4	68	13	5	1	7
69	8	3	0	5	69	6	1	1	4
70	5	1	0	4	70	5	1	1	3
71	8	2	2	4	71	12	5	3	4
72	6	1	1	4	72	5	2	0	3
73	4	1	1	2	73	9	4	1	4

74	7	3	1	3
75	8	4	2	2
76	9	5	2	2
77	8	3	1	4
78	8	2	0	6
79	9	3	1	5

Group Answers

Groups Control

	Correct Defects				**Groups Treatment**				
						Correct Defects			
Group #	total	Use Case	Specification	Requirement	Group #	total	Use Case	Specification	Requirement
1	6	3	1	2	1	7	2	0	5
2	10	4	0	6	2	7	3	1	3
3	9	3	1	5	3	11	6	1	4
4	5	1	0	4	4	12	6	1	5
5	8	3	1	4	5	7	2	1	4
6	2	0	0	2	6	9	4	2	3
7	6	0	0	6	7	9	2	1	6
8	10	4	0	6	8	14	6	2	6
9	5	0	0	5	9	9	3	1	5
10	6	0	0	6	10	11	4	1	6
11	8	3	1	4	11	9	5	1	3
12	8	0	0	8	12	9	6	1	2
13	4	0	0	4	13	13	6	3	4
14	8	3	0	5	14	11	4	2	5
15	11	5	0	6	15	12	5	1	6
16	12	5	0	7	16	11	3	2	6
17	6	1	0	5	17	12	6	2	4
18	11	5	1	5	18	9	4	1	4
19	11	6	2	3	19	12	6	2	4
20	10	4	0	6	20	13	7	1	5
21	9	3	1	5	21	10	4	1	5
22	7	2	1	4	22	10	4	2	4
					24	10	4	2	4
					25	10	5	1	4

26	11	5	2	4
27	10	4	1	5
28	13	5	4	4