

Experimental Evaluation of the Stimulus Response Requirements Specification Notation

Kendra M.L. Cooper and Mabo R. Ito
The University of British Columbia
kendrac@ece.ubc.ca, mito@ece.ubc.ca

Abstract

In order to transfer a new tool or technique to industry, project managers need some kind of an evaluation to determine if its benefits outweigh the costs. There are several ways to evaluate a new tool or technique including cases studies, pilot projects, or experiments. In this work, an experimental evaluation is selected to objectively evaluate a new, formal requirements specification notation, Stimulus Response Requirements Specification (SRRS). The SRRS notation is designed for the specification of large, software intensive systems with complex data requirements. An overview of the experimental design, results, and the conclusions based on the experimental data are presented in this paper. The results are very encouraging and indicate the SRRS notation is a cost effective way to develop requirement specifications. The time to write, review, and correct the specification is reduced as are the number of defects detected in a peer review process. There is, however, an increase in the training time for the authors.

1. Introduction

The evaluation of requirements engineering techniques have been performed using case studies, pilot projects, and experiments [Bri94, Cra94, Lar96, Por95, Por98]. Evaluations are used to discover strengths and weaknesses in a technique, determine the costs of applying a technique, or determine the benefits of using a technique. The evaluations provide qualitative and/or quantitative results that can be used by project managers to decide if the technique is going to be applied on a particular project [Bea91]. For new techniques developed in the academic world, such evaluations are very important to encourage transferring the technique to industry.

The purpose of this experiment is to objectively evaluate the costs and benefits of using a formal version of a software requirements specification notation, SRRS [Coo01], in comparison to its semi-formal version. The costs of introducing a formal notation include the cost of training employees in the tools and in the formal notation. To measure the costs, the amount of time spent in classroom and on the job training is recorded. The benefits include the availability of tools to assist the authors in automatically parsing, typechecking, and analyzing the specifications. The tool support is expected to reduce the time to develop the specifications and improve their quality. To measure the benefits, the amount of time used to write, review and correct the specifications is recorded in addition to the number and type of defects recorded in the peer review process.

The organization of this paper is as follows. An overview of the experimental design is in Section 2. The experimental results are summarized in Section 3. A complete description of the experimental design and results are in [Coo01]. Conclusions and future work are in Sections 4 and 5.

2. Experimental Design Overview

In this experiment, the treatment variable is the requirements specification notation used to write the specification units. The control group uses a semi-formal version of the SRRS notation and the experimental group uses the formal version. The experiment evaluates the effort required (training time, writing time, reviewing time, correcting time) and the quality (total number and category of the defects found) in developing a set of specification units.

The subjects for both the control and the experimental groups are undergraduate computer science and computer engineering students. There are three subjects for each group. The subjects are hired based on their qualifications for the task including a background in software engineering and their communication skills. The subjects are not randomly selected.

The overall structure of the experimental design is based on the work done by [Por95]. The experiment begins with a formal (classroom) training session for the subjects. As part of the formal training, a hands-on practice exercise is given to the subjects to complete. When the formal training is done, the subjects begin work on the experimental exercise.

The formal training for both the control and the experimental groups is structured as a set of four modules that are presented in a lecture format. Module one provides a brief introduction to the project, reviews the waterfall and spiral software development lifecycle models, and describes the characteristics of a “good” requirement. Module two covers the initial phases of the software development lifecycle. It clarifies the distinction between a system level specification (SLS) and a software requirements specification (SRS), describes the process to develop a SRS, and reviews traceability. Module three introduces the requirements specification notation used by the group and describes the six step process used to describe the functional requirements in an SRS. Module four reviews the requirements to system level testing interface and emphasizes the impact the quality of an SRS has on the development of system level test cases. The four modules are composed of approximately 100 presentation slides.

The experiment uses a six step process for writing a specification unit. The inputs for the first step in the process are the SLS, the title of the specification unit to be developed, and a list of requirement object identifiers (ROIDs) that are allocated to the title. The first step in the process is to ensure that each of the allocated ROIDs should be allocated and to determine if any ROIDs are missing from the allocation. After the ROIDs are checked, the author drafts an outline of the specification unit in step two. The section headers are entered and each section is outlined in point form. In the third step, the author completes the version of the specification unit by refining the outline and filling in the details of the specification unit. In addition to completing the first version, a traceability report is generated to indicate how the allocated ROIDs have been addressed in the specification unit. The specification unit and the traceability report are submitted for a peer review in step four. The author’s peers review the specification unit and record defects according to type, category, defect checklist identifier, and a brief description. When the peer review is complete the specification unit, traceability report, and the defect recording sheet(s) are returned to the author. The author corrects the specification unit and revises the traceability report as necessary in step five. The review cycle is complete in the experiment when the specification unit has no defects detected or when the review cycle has occurred twice. The final version is submitted in the last step of the process.

A package of support materials is provided at the beginning of the formal training session to each subject. The package consists of the presentation slides for the four module training course, samples of the forms used to collect data, the checklist used to assist data collection, and a hands-on practice

exercise. The experimental group is also provided with a copy of the SRRS user manual and the tutorial [Coo99].

Two forms are used to assist in the collection of the experimental data: time and defect recording sheets. Time recording sheets are used to record the subject name, date, specification unit identifier, specification version number, and the amount of time spent on an activity. The possible activities are: formal training; informal training; writing version one of a specification unit; reviewing a specification unit; revising a version of a specification unit; and “other”. Defect recording data sheets are used to record the subject name, date, specification unit identifier, specification version number, and the defects detected. The defects are reported with an identification number, type of error, category of error, checklist identifier, and a textual description. A checklist of different defects is provided to assist the reviewer (Refer to [Coo01] for samples of the data collection forms and the defect checklist).

The experimental exercise uses a SLS derived from a request for proposal for an integrated on-line library system [Cor87]. The SLS consists of 652 requirement objects. From this SLS, 52 specification units are identified and 432 software requirements are allocated to them. 18 of the specification units are randomly selected and written by both the control and the experimental groups.

3. Experimental Results

The experimental results including defect rates, training time, and the time to write, review, and correct the specification units are summarized in this section. Group 1 in the results refers to the control group using the semi- formal version of the requirements specification notation. Group 2 refers to the experimental group using the formal notation.

The experimental results for the defect rates are summarized in Table 1. The results show a reduction in the syntax, type, and the analysis defects detected for Group 2. The % difference between the Group 1 and Group 2 for the total number of defects detected per allocated ROID shows an 81% reduction in detected defects.

Table 1. Summary of Defects Recorded

	Number of syntax defects per ROID	Number of type defects per ROID	Number of analysis defects per ROID	Number of total defects per ROID
Group 1	0.99	0.74	0.88	2.61
Group 2	0.09	0.01	0.39	0.49
% Difference	-90.91	-98.65	-55.68	-81.23

The training time is a metric of interest for individuals considering the use of the formal notation in comparison to its semi-formal notation. In this experiment, the formal training time includes the time the subjects are in the lecture style format plus the amount of time they spend on the hands-on practice exercise. In addition, the amount of time it takes the subjects in the experimental group to work through the tutorial for the SRRS tool is considered as formal training. The formal training time is recorded as the number of minutes of formal training per author. The informal training includes the time the subjects use to review their training material or ask questions about the notation as they write the specification units. The informal training is recorded with respect to the number of allocated ROIDs, because the informal training continues as the subjects write their specifications. The total training time recorded is the sum of the formal training time and the informal training time per author. The experimental results for training

time are summarized in Table 2. They show an increase in both the formal and informal training time for Group 2. The % difference between Group 1 and Group 2 is a 186% increase in total training time.

Table 2. Summary of Training Time

	Formal Training Time Minutes/author	Informal Training Time minutes/ROID	Total Training Time minutes/author
Group 1	420.00	0.32	448.33
Group 2	835.00	5.02	1285.00
% Difference	98.81	1468.75	186.62

The effort is a metric of interest for individuals considering the use of the formal notation in comparison to its semi-formal notation. The effort to write and put the specification units through a peer review is summarized in Table 3. The experimental results show a decrease in the amount of time to write and review requirements for Group 2. The % difference between Group 1 and Group 2 for the total amount of time to write and review requirements is a reduction of 39%.

Table 3. Summary of Writing and Reviewing Time

	Average time to write per ROID in minutes	Average time to review and correct per ROID in minutes	Average total time per ROID in minutes
Group 1	17.58	15.28	32.86
Group 2	10.58	9.42	20.00
% Difference	-39.82	-38.35	-39.14

4. Conclusions

The formal and informal training time both increased as expected for the Group using the formal notation in comparison to the group using the semi-formal notation. The additional burden of working through a tutorial, learning how the tool support works, and understanding the organization of the user manual all contribute to this increase.

The large increase in the informal training time is an interesting result. An explanation for this large increase is the additional complexity of having a concrete syntax and the tool support for the notation. Since the syntax must be conformed to and the validation checks all passed, the authors of the requirements may need to check the user manual, training material, and the tutorial documents more frequently as they work on the specifications.

With these experimental results an estimate of the training time for a project can be made. Given the notation proposed, number of authors, and number of allocated ROIDs to be written the following calculation can be used to estimate the training time: $\text{training time} = A * T + R * D$, where A is the number of authors, T is the training time per author for a given notation, R is the number of ROIDs and D is the development time per ROID for a given notation. For example, if the formal notation is used, the T is 835 minutes/author and D is 5.02 minutes/ROID.

The experimental results show a reduction in the number of defects detected in the peer review process. The concrete syntax in addition to the tool support that enforces the syntax and provides validation checks on the specification units allow the author to check their specifications before submitting them for review. Since only specifications that have a clean validation run (no errors are reported) with the tool support are allowed to go through the peer review process in the experiment, the

reviewers receive a version that has had defects removed. As a result the specification units have a lower detected defect rate and have a higher quality. This reduces the overall project costs, since correcting defects in the software increases an order of magnitude for every phase the error is propagated. The sooner defects are discovered and corrected the better [Hum95]. The goal is to correct the defects in the same phase they are introduced in. The most expensive errors to correct are those that are detected by the customer after delivery, and are traced back to being introduced in the software requirements phase. Multiple levels of code, design, and requirement products must be updated.

The experimental results also indicate there is a reduction in effort to write, review, and correct the specification units when using the formal notation in comparison to the semi-formal notation. The reduction can be attributed to the readability of the formal notation and the tool support. The reduced writing, review, and correction time indicates that the formal notation is at least as readable as the semi-formal notation. If the formal notation is not as readable, the time to write, review, and correct is expected to exceed the time when the semi-formal notation is used. The tool support allows the author to obtain feedback on syntax, type, and a small number of analysis defects as the specification is being written. These defects can be removed before the specification is submitted for peer review. This reduces the number of defects in the specification making the remaining defects simpler and faster to identify in the peer review. A reduction in time is a benefit to the project, as it reduces the cost of developing the requirements specification.

With these experimental results an estimate of the development time for a project can be made. Given the notation proposed and number of allocated ROIDs to be written the following calculation can be used to estimate the development time with the simple calculation $\text{development time} = R * D$, where R is the number of ROIDs and D is the development time per ROID for a given notation. For example, if the formal notation is used, then D is 5.02 minutes/ROID.

The point at which it becomes feasible to use the formal notation can be estimated using the experimental results. For example, if a project has 2000 ROIDs and proposes to use 20 authors, the time to train, write, and review the specifications can be estimated for both the formal and semi-formal notations. The total time estimate is calculated as the sum of the training time and the development time (refer to Table 4).

Table 4. Example of Total Time Estimates

	Training Time	Development Time	Total Time
Semi formal notation	$20 * 420 + 2000 * 0.32$ = 9042 minutes	$2000 * 32.86$ = 65720 minutes	$9042 + 65720$ = 74760 minutes
Formal notation	$20 * 835 + 2000 * 5.02$ = 26740 minutes	$2000 * 20.00$ = 40000 minutes	$26740 + 40000$ = 66740 minutes

With these two calculations the formal notation shows a savings in time of approximately three weeks. If the project is scaled up, then a more dramatic time savings is calculated. For example, if the number of authors and the number of requirements are scaled up by an order of magnitude and the calculations shown above are repeated, the formal notation has a calculated time savings of approximately seven months. The selection of the formal notation in this case offers significant cost savings to the project. If the project is scaled down by an order of magnitude, then the formal notation

has a calculated time savings of approximately 13 hours. Given a small project involving a couple of authors and 200 requirements, there is a slight advantage in selecting the formal notation. The caveat in the estimates made here is that the data used in the calculations is derived from one project with 432 allocated ROIDs. The data has not been confirmed in different projects of different sizes.

The results of this experiment have quantified the costs and benefits of using a formal notation with tool support in comparison with a similar, semi-formal version of the notation. The costs are increased classroom and on the job training time. The benefits include a reduced effort to write and review the specification units and a reduced defect rate. The experimental results support the use of the formal notation in that the additional costs of training (in terms of time) are overcome by the gains achieved in the reduction of the amount of time to write and review the specification units.

5. Future Work

There are a number of interesting areas of work to pursue in this research. Additional replications of the experiment to validate the results would be valuable. In particular, it would be interesting to replicate the experiment using software engineering professionals as the subjects and to replicate the experiment using different SLS documents. After validating the experimental results, a small pilot project in industry is the next step in transferring and evaluating the notation for its use in industry. With feedback from industry, the SRRS notation and tool support could be appropriately modified.

References

- Bea91 S. Bear, "Managing the introduction of formal methods", IEE Colloquium, 1991, No. 131: Managing critical software projects.
- Bri94 J. Britt, "Case study: Applying formal methods to the traffic alert and collision avoidance system (TCAS) II", Computer Assurance: COMPASS '94, pp 39-51.
- Coo01 K. Cooper, Ph. D. Thesis, The University of British Columbia, expected completion May, 2001.
- Coo99 K. Cooper and M. Ito, Training Material and User Documentation for the Stimulus Response Requirements Specification Notation, CICS Technical Report TR99-001, The University of British Columbia, 1999.
- Cor87 Edwin M. Cortez, Proposals and Contracts for Library Automation: Guidelines for Preparing RFP's, Pacific Information Inc., USA, 1987.
- Cra94 D. Craigen, S. Gerhart, T. Ralston, "Case study: Darlington nuclear generating station", IEEE Software, January 1994, Volume 11, pp 30-32.
- Hum95 W. Humphrey, A Disciplined Approach to Software Engineering, Addison-Wesley Publishing Company, Inc., Canada, 1995.
- Lar96 P. Larsen, J. Fitzgerald, and T. Brooks, "Applying formal specification in industry", IEEE Software, May 1996, pp 48-56.
- Por95 A. Porter, L. Votta, and V. Basili, "Comparing detection methods for software-requirement inspections: A replicated experiment". IEEE Transactions on Software Engineering, 21 (6), June 1995, pp. 563-575.
- Por98 A. Porter and L. Votta, "Comparing detection methods for software-requirement inspections: A replication using professional subjects", Journal of Software Engineering, Volume 3, Number 4, December 1998, pp. 355-379.

