Advantages of Stimulus Response Requirement Specification Techniques for System Testing

Kendra Cooper
The University of British Columbia
Department of Electrical and Computer Engineering
2356 Main Mall Vancouver, B.C. V6T 1Z4
kendrac@ee.ubc.ca

Mabo Ito
The University of British Columbia
Department of Electrical and Computer Engineering
2356 Main Mall Vancouver, B.C. V6T 1Z4
mito@ee.ubc.ca

Abstract. This paper presents the advantages of using a Stimulus Response Requirement Specification (SRRS) technique to reduce costs, development time, and errors in the generation of system level test cases in comparison to using a Structured Analysis (SA) technique. A SRRS technique describes requirements in terms of externally visible inputs (stimuli), processing, and externally visible outputs (responses). The SRRS techniques may be viewed as a general category of requirements specification techniques that are based on concepts originating in system test methods. Currently used, popular SRRS techniques include Use Cases and Scenario driven techniques. This paper first reviews a basic process for developing system test cases. An example from a library system is used to illustrate the first step of this process with a conventional SA requirements specification and a SRRS. The use of a SRRS technique to document the requirements specification simplifies the test case generation process by eliminating the first step in the three step process.

INTRODUCTION

Requirement specification techniques may be categorized based on how the requirements are partitioned. The taxonomy used in this work is based on two broad partitioning categories: the internal, or construction view, and the external, or stimulus response view, of the requirements. The internal view of partitioning the requirements is used in Structured Analysis (SA) techniques whereas the external view is used in Stimulus Response Requirement Specification (SRRS) techniques such as Use Cases and Scenarios.

SA techniques have been in use in industry for over twenty years (Ross 1977, Demarco 1978, Hatley 1988) and are now considered to be a well established, or conventional, technique for specifying requirements. The SA techniques use an internal, or construction view, of the system based on functional decomposition to "divide and conquer" the complexity of the system. However, this approach to decomposing the system suffers from a number of problems:

- There is a lack of user's perspective and guidance during project development (Hsia 1988). The system is developed from a system generation perspective, not a system acceptance perspective. Users find it difficult to identify the parts of the decomposition that map to their specific requirements.

- There is a lack of progress visibility (Hsia 1988). Users are left out of the software development process until the product is integrated and demonstrated to them. Documentation and code are the typical products which reflect the progress of the project.

- There are maintenance problems (Hsia 1988). Minimality, the goal of documenting one requirement in one place, usually reduces the understandability of the model because it increases the interactions among the components. This increased coupling makes maintaining the requirements specification model more difficult, as changes to the model are not localized.

- The functional decomposition may imply a design (Alford 1977). The functional decomposition may be viewed as a specific design, with subroutines to implement subfunctions. If an object oriented design is proposed, the mapping of the requirements to the design may be very time consuming.

- There are difficulties testing the requirements (Alford 1977). A fundamental difficulty with the functional decomposition is that the processing for one stimulus might be described by a number of the subfunctions in the model. In addition, the input to exercise a specific subfunction may be difficult to construct. Since most requirements are stated at the subfunction levels, it is difficult to test such a specification.

Although SRRS techniques and tool support have also been available for over twenty years (Davis 1977, Deutsch 1988), they have been overshadowed by the SA techniques. This is changing as Use Case and Scenario driven techniques (Jacobson 1992, Regnell 1996), varieties of the SRRS category, are being incorporated into software development methodologies. SRRS techniques are seen as a means of overcoming the shortcomings of the SA techniques. The SRRS techniques focus on structuring the requirements from an external, or user's, view of the system. This partitioning has a number of advantages:

- The requirements specification is driven by the user's perspective. The mapping from their needs to the requirements specification document is straightforward. This makes the document well suited for use as a contract and communication tool between the customer and the developers.

- The requirements may be clustered for an incremental, or iterative, development lifecycle, allowing subsets of the system to be implemented and demonstrated to the customer (Hsia 1988). The progress of the project may be monitored by the customer using these demonstrations.

- The requirements are straightforward to maintain because the user's tasks are decoupled and documented separately in the specification.

- The requirements specification does not imply a design, and may be mapped to either a functional or object oriented design.

- The system level test cases are simpler to generate. The external view of the system is well suited for use as a working document by the system level test authors, as the external stimulus and response pairs are documented in a single place.

This paper examines the benefits of using a SRRS technique to simplify the test case generation process. The basic steps in a test case generation process are reviewed in this paper. Following the review, a comparison of using the process with a SA and a SRRS technique is made using an example from an automated library system requirement specification. The process is shown to be simpler when a SRRS technique is used and is amenable to partial automation.

PROCESS OF DEVELOPING SYSTEM LEVEL TEST CASES FROM REQUIREMENTS

A systematic process used to derive system level test cases from a requirements specification has three basic steps. The first step is the extraction of the system level test threads from the requirements specification. A thread is an end-to-end description of the requirement that needs to be verified and is composed of the external stimulus, conditions that must be met, and external responses. The second step uses the test threads to write the test frames. Each test thread is decomposed into one or more test frames based on the logical connectives used in the condition. Thirdly, the test frames are instantiated with values and sequenced to generate a test case. The process is described using the Integrated Definition Methodology (IDEF0) in Appendix A.

EXAMPLE OF GENERATING TEST THREADS

The example provided in this section is derived from a library system specification described in (Cortez 1987) as the Integrated On-line Library System (IOLS). The portion of the requirements used as an example in this work is the "Discharge Item" function which allows a borrower to return an item (book, magazine, cd, video, etc.) to the library. In the IOLS a request to return a library item is rejected if the item is not in the catalogue database, the borrower is not in the borrower catalogue, or the status of the item in the catalogue database is not "charged out" or "missing". If the discharge request passes these validation checks, then the status of the item is updated to "discharged" and the borrower's account is updated to indicate the item has been returned. If the item is returned late, the fine is calculated and the borrower's account is updated to include the new fine.

Although this part of the library system is small, it is useful to illustrate the differences in the first step in the test case generation process (extracting the test threads) using a specification written with a SA technique and a SRRS technique. The test threads generated are the same when extracted from the SA and the SRRS techniques. There is a difference in the process, however, and as a result in the amount of effort necessary to extract the test threads from the requirements specification. The test threads are summarized in Table 1. After the test threads are available, the second and third steps of the test case generation process are the same.

SA Requirements Specification Example. The requirements to discharge an item are described in this section using an SA technique. The level one diagram in the IOLS SA model has eight data flow diagram bub-

Test Thread	Stimulus	Condition	Response
1	discharge item request	the item identifier is not found in item catalogue *or* the borrower identifier is not found in the borrower catalogue *or* the item with item identifier does not have an item status of "missing" *or* the item with item identifier does not have an item status of "charged out".	rejection with reason.
2	discharge item request	the discharge request is not rejected *and* the discharge date occurs on or before the due date *or* the discharge request is not rejected *and* the discharge date occurs after the due date.	update the item status to "discharged" in the item catalogue, update the borrower's account to discharge the item, update the borrower's fine/fee owing.

Table 1: System Level Test Threads for the Discharge Item Library Example

bles: 1. Maintain Catalogue, 2. Manage Acquisitions, 3. Manage Circulation, 4. System Services (including security, communication, and date/time management), 5. Statistical Gathering and Analysis, 6. Manage Accounting, 7. Maintain Serials, and 8. Maintain Reference. In order to extract the end-to-end test thread for the discharge item functionality, six primitive level process specifications (PSPECS) are used. The PSPECS have been identified using the heuristics in (Jorgensen 1995). Each PSPEC has a unique identifying number (using the standard SA numbering convention) title, input, output, and description section and are outlined below:

1.4.6.1 Obtain Catalogue Item's Status
Input: item identifier
Output: rejection/item status
Description: This PSPEC obtains the status of a catalogue item. Values allowed include "missing", "charged out", "hold", "discharged", "on shelf", "in process". If the item identifier is not found in the catalogue, the IOLS shall output a rejection. Otherwise the IOLS shall output the item's status.

1.4.6.2 Update Catalogue Item's Status
Input:item identifier, item status
Output: rejection/acceptance
Description: This PSPEC updates the status of a catalogue item. Values allowed include "missing", "charged out", "hold", "discharged", "on shelf", "in process". If the item identified is not in the catalogue or the status is not one of the allowed values, the IOLS shall output a rejection. Otherwise the IOLS shall output an acceptance.

1.4.2.1 Obtain Catalogue Item's Due Date
Input:item identifier
Output: rejection/due date
Description: This PSPEC obtains the catalogue item's due date. If the item identified is not in the catalogue or the status of the item identifier is not "charged out", the IOLS shall output a rejection. Otherwise the IOLS shall output the due data of the catalogue item.

3.3.1 Update Borrower's Fine/Fee Owing
Input:borrower identifier, amount
Output: rejection/acceptance
Description: This PSPEC updates the amount of money the borrower owes in fees and fines. If the borrower identifier is not in the borrower database, the IOLS shall output a rejection. Otherwise, the IOLS shall update the borrower's fines/fees owing with the amount and output an

acceptance.
3.4.2 Update Borrower's Item Charges
Input: item identifier, borrower identifier, charge/discharge
Output: rejection/acceptance
Description: This PSPEC updates the items that are charged out (i.e. signed out) to a borrower. If the borrower identifier is not in the borrower database or the item identifier is not in the catalogue the IOLS shall output a rejection. If the update is to charge an item and the item's status is not "on shelf" the IOLS shall output a rejection. If the update is to discharge an item and the item's status is not "charged out" the IOLS shall output a rejection. If the update is to charge an item and the request has passed the validation checks the IOLS shall update the borrower's account to include the item identifier. If the update is to discharge an item and the request has passed the validation checks the IOLS shall update the borrower's account to remove the item identifier from their charges.

4.3.2 Obtain Current Date
Input: date/time request
Output: date/time
Description: This PSPEC obtains the current date or time in the system.If a date request is received the IOLS shall output the system date. If a time request is received the IOLS shall output the system time.

Extracting the Test Threads from the SA Specification. The six PSPECS used to describe the test threads for discharging an item cut cross three sets of diagrams and are found at varying levels in the hierarchies. When extracting the test threads in the test case generation process, the test author must work through the SA model to obtain an end-to-end test thread. In this example, the sequence of PSPECS to determine if the discharge request is valid is PSPEC 1.4.6.1. The sequence of PSPECS used to update the status of the item, update the status of the borrower's account, and if necessary, update the borrower's fines/fees owing is: PSPEC 1.4.6.1, PSPEC 1.4.6.2, PSPEC 3.4.2, PSPEC 4.3.2, PSPEC 1.4.2.1, and PSPEC 3.3.1.

SRRS Requirements Specification Example. The following discharge item requirements are written in a SRRS technique in a single section in the software requirements specification document. The three sections in the specification are the title, overview, and requirements. The title uniquely identifies the task in the specification document while the overview provides a summary of what the task provides for the user. The requirements have the basic form *stimulus*, *condition*, *response*.

Title:
Discharge Item

Overview:
The Discharge Item specification unit describes the processing performed upon receipt of a request to discharge (i.e., return) an item to the library. If the item is found in the item catalogue then the discharge request is processed. Otherwise, the request is rejected.

Requirements:
1)Upon receipt of a discharge item request from an operator, if any of the following conditions hold:
a) the item identifier is not found in IOLS item catalogue;

b) the borrower identifier is not found in the borrower catalogue;

c) the item identifier does not have the status of "missing", or "charged out"

, then the IOLS shall return a rejection with reason to the operator.

2)Upon receipt of a discharge item request, if the request is not rejected, then the IOLS shall:
a) update the borrower's account to discharge the item;

b) update the item's status in the catalogue to "discharged";

c) if the discharge date occurs after the due date then the IOLS shall commit the discharge fine to the borrower's account.

Extracting the Test Threads from the SRRS. The test threads for the discharge item functionality are visible directly from the requirements specification and are found in a single location. The process step to extract the test threads has already been accomplished using the

stimulus response style for the requirements specification.

DISCUSSION

The testability of the SRRS techniques has been described as one of the key concepts in their development (Alford 1977). For example, the development of one of the early techniques, the software requirements engineering methodology (SREM), “is based on the observation that real-time software is tested by inputting an interface message and extracting the results of the processing -- output messages and the contents of memory” (Alford 1977). The SREM technique is based on providing support for describing requirements in terms of inputs, processing, and outputs to improve the testability of the specification. The SRRS techniques may be viewed as an application of system test methods to the requirements specification phase of the software development lifecycle.

With respect to the test case generation process, the advantages of using the SRRS techniques are evident in the identification of the system level test threads. This step is time consuming and prone to errors when the requirement specification is described using the conventional SA technique. In the small example, the author must work through three data flow diagram hierarchies (1,3, and 4) to develop the test threads. The process is simplified if the requirements are written in a more testable SRRS style because the requirements are already structured in a convenient form. The stimuli, conditions, and responses are immediately visible to the authors in a single location and test frames may be generated *directly* from the requirements specification.

In addition to simplifying the manual process, the visibility of the stimulus, conditions, and responses in the stimulus response techniques supports automating the generation of test frames directly from the requirements specification. Automating this process is of significant interest in industry as the advantages include reduced time, development costs, and errors. To support automation, a machine scannable and parseable notation with defined syntax and semantics is needed. One option for automating the generation of test frames from a SRRS is to extend the *stimulus*, *condition*, *response* phrasing concept such that the entire specification is written using a notation that reads like English, has a defined syntax and semantics, and is completely machine scannable and parseable. The automatic generation of test frames from a requirements specification using this technique is described in (Donat 1998).

CONCLUSIONS AND FUTURE WORK

Adopting a SSRS technique may reduce development time, costs, and errors in two ways. Firstly, the extraction of the system test threads from a specification is straightforward for the test authors in comparison to extracting test threads from a SA model of the requirements. Since the requirements are already described in an end-to-end stimulus response style, the step in the test case generation process to extract the test threads is eliminated.The testability of the requirements written in a SRRS technique is a result of the techniques being based on concepts originating in system testing methods. System testing is based on describing inputs (stimuli), processing, and expected results (responses). Secondly, if the step to generate test frames is automated, as described in (Donat 1998) additional reductions in development time, costs, and errors are expected.

Having recognized the advantages of the SRRS technique, the development of tools and techniques to automate the generation of test frames from a SRRS are currently being investigated in the FormalWare project at The University of British Columbia.

ACKNOWLEDGEMENTS

I would like to thank Dr. Jeffrey J. Joyce for his valuable comments and contributions to this work.

This work is supported by ***FormalWARE*** (Joyce 1998) a university-industry collaborative research project sponsored jointly by the BC Advanced Systems Institute, Hughes Aircraft of Canada Limited, MacDonald Dettwiler, The University of British Columbia, and The University of Victoria.

APPENDIX A. A Process for Generating System Level Test Cases

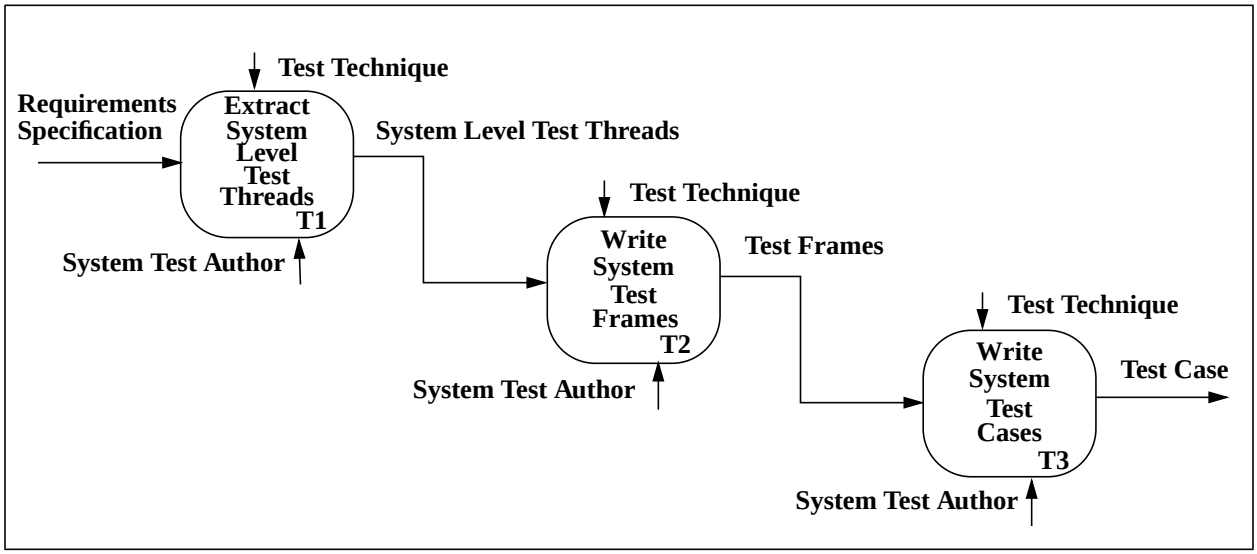

Figure 2: A Basic Process for Generating System Level Test Cases

Step T1:Generate Test Threads.
Input. The requirements specification is the input to this step. The requirements for a project are written using a particular technique. For example, requirements specified using the conventional SA technique are documented with data and control flow diagrams, a data dictionary, and the process specifications for the primitive level processes.

Process Description. To extract the threads, the system test authors weave through the requirements to obtain an externally visible, end-to-end requirement. If the requirements specification is written in a SA technique the authors may examine multiple data and control flow diagrams, process specification descriptions, and through the data dictionary to obtain the end-to-end system test thread. This process step is a time consuming and error prone process. Heuristics to aid the author in identifying the system test threads from requirements specified in a Structured Analysis model are provided in (Jorgensen 1995).

Output. The system test threads are generated. Each test thread has a stimulus and response, in addition to tracing information to the requirement specification document. The stimulus and response may be associated with logical conditions expressed in terms of logical connectives such as ‘and’ and ‘or’.

Step T2:Generate Test Frames
Input. The system test threads are the input to this step.

Process Description. The test threads are used to generate test frames. To manually derive test frames from a requirement, test engineers are generally guided by words and phrases that occur in the text of the requirement. The derivation of the test frames focuses on dissecting the logical complexity within the requirements. This complexity is brought about by the composition of logical formulae, the embedded choices (the presence of the word "or"), and the context of those choices. For example, the presence of the word "or" in the antecedent of a requirement of the form, "When Stimulus S occurs and Condition C1 or Condition C2 is true, then the system shall produce Response R." indicates that the requirement must be decomposed into at least two separate test frames: one for when Condition C1 is true and another separate test frame for when Condition C2 is true.

Output. Test frames are generated in this step. Each test frame is composed of a source, stimulus, condition, expected response, and destination along with tracing information to the requirements specification.

Step T3:Generate Test Cases.
Input. Test frames are used as the input to this step.

Process Description. The test frames are used to generate test cases. The test frames are instantiated with data values and sequenced to produce test cases.

Output. Test cases are generated in this step.

REFERENCES

Alford, Mack, “A Requirements Engineering Methodology for Real-Time Processing Requirements”, *IEEE Transactions on Software Engineering*, Volume SE-3, Number 1, January 1977, pp 60 - 69.

Cortez, E., *Proposals and contracts for library automation: guidelines for preparing RFPs*, American Library Association, USA, 1987.

Davis, C. and Vick, C., “The Software Development System”, *IEEE Transactions on Software Engineering*, Volume SE-3, Number 1, January 1977, pp 69-84.

Demarco, Tom, *Structured Analysis and System Specification*, Prentice-Hall, Inc., USA, 1978.

Deutsch, M., “Focusing Real-Time Systems Analysis on User Operations”, *IEEE Software*, September 1988, pp 39-50.

Donat, Michael and Joyce, Jeffrey, “Applying an automated test description tool to testing based on system level requirements”, in 8th Annual Symposium of the International Council on Systems Engineering, INCOSE July 1998. To appear.

Hatley, D. and Pirbhai, I., *Strategies for Real-Time System Specification*, Dorset House Publishing Co., Inc., USA, 1988.

Hsia, P. and Yuang, A., “Another Approach to System Decomposition: Requirements Clustering”, In Proceedings of the Twelfth Annual International Computer Software and Applications Conference (COMPSAC 88), Chicago, USA, pp 75-82.

Jacobson, Ivar, *Object-Oriented Software Engineering a Use Case driven approach*, Addison Wesley Longman Ltd., England, 1992.

Jorgensen, P., *Software Testing A Craftsman’s Approach*, CRC Press, Inc., USA, 1995.

Joyce, Jeffrey, FormalWARE project director, 1998. `http://www.cs.ubc.ca/formalWARE`

Jorgensen, P., *Software Testing A Craftsman’s Approach*, CRC Press, Inc., USA, 1995.

Regnell, B.,Anderson, M. and Bergstrand, J., “A hierarchical use case model with graphical representation”, in *IEEE Symposium and workshop on engineering of computer based systems*, Germany, March 1996, pp 270-277.

Ross D.and Schoman, K. Jr., “Structured Analysis for Requirements Definition”, *IEEE Transactions on Software Engineering*, Volume SE-3, Number 1, January 1977, pp 6-15

BIOGRAPHY

Kendra Cooper is a Ph.D candidate at The University of British Columbia in the Department of Electrical and Computer Engineering. Her research interests include investigating formal methods in requirements engineering.

Mabo Ito is a professor in The University of British Columbia in the Department of Electrical and Computer Engineering. His broad area of interest is computer engineering.