

Capturing the Logical Structure of Requirements for the Automatic Generation of Test Specifications

Michael R. Donat

Intrepid Critical Software Inc.
P.O. Box 39207
Vancouver, B.C. Canada
+1 604 733 4058
Michael.Donat@intrepid-cs.com

Kendra M.L. Cooper

The University of British Columbia
Department of Electrical and
Computer Engineering
2356 Main Mall
Vancouver, B.C. Canada
+1 604 822 5103
kendrac@ee.ubc.ca

Mabo R. Ito

The University of British Columbia
Department of Electrical and
Computer Engineering
2356 Main Mall
Vancouver, B.C. Canada
+1 604 822 4572
mito@ee.ubc.ca

ABSTRACT

This paper presents a new, alternate technique for automatically generating system-level requirements-based test specifications. The proposed technique does not require the construction of a mathematical model of the requirements. Instead, a parsable structuring of textual requirements is used to capture the logical relationships between conditions. The dramatic reduction in modeling effort together with automating a substantial portion of the test generation process provides the potential for reducing costs in system-level testing.

Keywords

requirements-based testing, test generation, formal methods

INTRODUCTION

The current practice in industry for generating system-level test cases from a requirements specification is a manual, time consuming process that is prone to errors. Test engineers use a requirements document, typically written in natural language, to develop test specifications by extracting related conditions from the text. The flexibility in the grammar and the inherent ambiguities in natural language make this task both challenging and expensive. The analysis of the requirements to produce the test cases is systematic, making it a good candidate for automation.

Techniques have been proposed to automatically generate tests from requirements specifications [1,4]. These automated techniques are based on mathematically modeled descriptions of the requirements. These models, or formal specifications, are developed to describe the semantics of the requirements in mathematical terms. Tool support is available to use the formal requirements specification and generate formal test specifications. Although these techniques automate the generation of the test specifications, they are not widely used in industry. A significant drawback to using these techniques is that their application is restricted to use by formal methods experts. In addition, creating and maintaining a formal requirements specification is expensive. These disadvantages contribute to the resistance of incorporating formal methods in industrial projects.

A new, alternate requirements-based testing technique is proposed in this paper which uses a readable and formal notation to capture the logical structure of the requirements, rather than attempting to describe the semantics of the requirements. The purpose of such a technique is to avoid some of the disadvantages of using formal methods, such as the lack of readability, while retaining the advantages of having a formally described requirements

specification, such as the automated generation of test specifications. The combination of readability and formality are desirable characteristics for use in industry.

The structure of this paper is as follows. After the introduction, the motivation for developing the new technique is discussed. A review of existing automated requirements-based testing techniques is presented, followed by the presentation of the new technique, conclusions and future work. An example specification and the test specifications automatically generated from it using the new technique are given in Appendices A and B, respectively.

MOTIVATION

In this paper, the term "system level, requirements based testing" refers to a particular level of testing where the purpose of each step in the test procedure is to verify one or more requirements. Each test step involves the application of an external stimulus to the system and the comparison of the actual response of the system against the expected response specified by the requirements. This level of testing is "system level" in the sense that the internal structure of the system is not visible; all of the testing must be performed by means of externally visible stimuli and responses. It is "requirements based" to distinguish it from other kinds of system-level testing such as performance testing.

In the context of requirements-based testing, special distinction is given to those conditions that represent an external stimulus and the responses of the system. These conditions are referred to as stimulus and responses, respectively. The term conditions, when used in conjunction with stimulus and responses, refers to those conditions other than the stimulus and responses.

A test suite is a set of test procedures that are made up of a sequence of test steps, or test cases. A test

step is composed of a stimulus, zero or more conditions, and one or more responses. In a typical large system, each test procedure serves as a script for a test session that would usually require no more than a few hours to execute.

To manually derive test steps from a requirement, test engineers are generally guided by words and phrases that occur in the text of the requirement. 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 steps: one for when Condition C1 is true and another separate test step for when Condition C2 is true. The test steps may be described in a tabular format for convenience, as below:

Step	Stimulus	Conditions	Response
1	S	C1, not C2	R
2	S	not C1, C2	R

The analysis of requirements for the purpose of manually deriving test steps for system-level testing is generally limited to the lexical analysis of the text used to express the requirements. Requirements-based testing 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. In this work, the derivation of test steps at the system level does not involve an analysis of the ranges of data values as is the case when using techniques such as Boundary Analysis and Equivalence Partitioning.

As noted above, the analysis required to produce test cases is of a lexical nature and is systematic, based on the logical relationships among the

conditions (including stimuli and responses). This makes the process a good candidate for automation. A first step towards automating this process is the capture of the requirements in a form that allows the conditions to be processed mechanically. The advantage to automating the generation of test specifications is that the process has the potential to significantly reduce the cost of software development. To support this process, a requirements language with a formal syntax is necessary. The correctness of the process is essential, therefore there is also a need for formal semantics. It is important to note that what must be formalized are the relationships between conditions. A fully modeled specification is not necessary.

EXISTING FORMAL TECHNIQUES

This section discusses two existing techniques for automatically generating tests from specifications and compares these to the current industrial practice introduced in the previous section. Each of these automated techniques generates tests from a formal model of the requirements specification. The techniques differ in the type of tests they produce. The first technique produces test specifications which describe both an output and the conditions on the inputs that cause the output to occur. The second technique produces test cases which specify actual instances that satisfy the conditions of a test specification. The disadvantages of these techniques and how they can be addressed is also discussed.

Dick and Faivre's technique [4] generates test specifications from a formal model of the requirements. In this technique, an operation is unfolded using definitions within the formal specification in order to obtain detailed conditions. The resulting logical expression is then reduced to a disjunctive normal form (DNF). Each disjunct represents a test specification. This technique also prescribes a method for sequencing the test specifications. The issue of test sequencing is not

discussed in this paper.

Consider Dick and Faivre's VDM-SL example in Figure 1. This specification models a triangle as a sequence of three positive integers with the restriction that twice any of these integers is always less than their sum. The operation *CHARACTERIZATION* is defined in terms of this model. Applying Dick and Faivre's technique to the post condition $type = classify(sides)$ results in 22 disjuncts that represent the unfolded DNF. Due to dependencies between conditions, 14 of the disjuncts represent infeasible test specifications. An example of an infeasible test specification is one containing a contradiction such as $(len\ sides' = 3) \wedge (sides' = [])$. Decision procedures are used to remove infeasible test specifications from the DNF. A typical test specification generated by this technique is

:

$$\begin{aligned}
 &2 = \text{card}\{sides'(2), sides'(3)\} \\
 &type = \text{ISOSCELES} \\
 &\text{elems } sides' = \{sides'(2), sides'(3)\} \\
 &\text{inds } sides' = \{1, 2, 3\} \\
 &sides'(1) = sides'(2) \\
 &sides'(2) + sides'(2) > sides'(3) \\
 &sides'(2) \in N_1 \\
 &sides'(3) \in N_1
 \end{aligned}$$

The conditions involving the input *sides* specify one set of properties *sides* must have in order for the triangle it represents to be classified as isosceles. The output is the classification $type = \text{ISOSCELES}$.

Another approach is Blackburn and Busser's T-VEC system [1,2] which produces test cases. Blackburn and Busser refer to their test cases as test vectors, distinguishing T-VEC from code-based test data generators. Test data generators only provide the input data and do not include details for outputs. As in Dick and Faivre's technique, T-VEC processes a formal model of the requirements specification. T-VEC specifications also associate data ranges with types. This allows T-VEC to select appropriate data values for input. The specifications are themselves executable which provides the means of computing

```

types
Triangle = N1*
    inv t  $\triangle$  len t = 3  $\wedge$  ( $\forall i \in \text{elems } t \bullet (2 \times i) < \text{sum}(t)$ )
TriangleType = SCALENE | ISOSCELES | EQUILATERAL | INVALID

functions
sum : N1*  $\rightarrow$  N1
sum(seq)  $\triangle$  if seq = [] then 0 else (hd seq) + sum(tl seq)

classify : N1*  $\rightarrow$  TriangleType
classify(sides)  $\triangle$  if is Triangle(sides) then variety(sides) else INVALID

variety : Triangle  $\rightarrow$  TriangleType
variety(sides)  $\triangle$ 
    cases card(elems sides):
        (1)  $\rightarrow$  EQUILATERAL,
        (2)  $\rightarrow$  ISOSCELES,
        (3)  $\rightarrow$  SCALENE
    end

operations
CHARACTERISATION(sides : N1*) type : TriangleType
    post type = class(sides)

```

Figure 1. Dick and Faivre’s VDM-SL Example

the output values for each input. An excerpt of the example T_VEC specification in [2] is:

```

...
LEVEL {
RELATIONSHIP Safety_Injection = ON;
RELEVANCE PREDICATE {
    DISJUNCTION {Pressure2 = TooLow,
        Overridden2 = false,
        Overridden Term};
    DISJUNCTION {Pressure2 = TooLow,
        Overridden2 = false,
        Overridden1 = false,
        NOT:At_T_Reset TooLow,
        NOT:At_T_Reset Permitted,
        NOT:At_T_Inmode High,
        NOT:At_T_Inmode TooLow Permitted,
        NOT:At_T_Block_On};
}
...

```

Figure 2 illustrates the differences between the industrial process and the two formal test generation techniques discussed in this section. The advantages of the industrial process are:

1. No time is spent on formal modeling
2. Test designers do not need training in formal methods
3. There is one document describing the system-level requirements.

The primary disadvantage of the industrial process is that a significant amount of manual effort is required to produce and validate the tests.

The advantages of using a formal test generation technique such as those described above are:

1. A high degree of assurance can be given as to the correctness of the tests produced¹, relative to the formal model
2. Tests are generated automatically
3. Specification errors can be found during the formalization process.

The disadvantages of these techniques are:

1. Special training is required to produce a formal model of the requirements
2. A significant amount of time is required to produce the formal model and ensure that it cor-

1. The tests produced rely on the correctness of the test generation tool. If the tool is correct, the correctness of the tests is guaranteed.

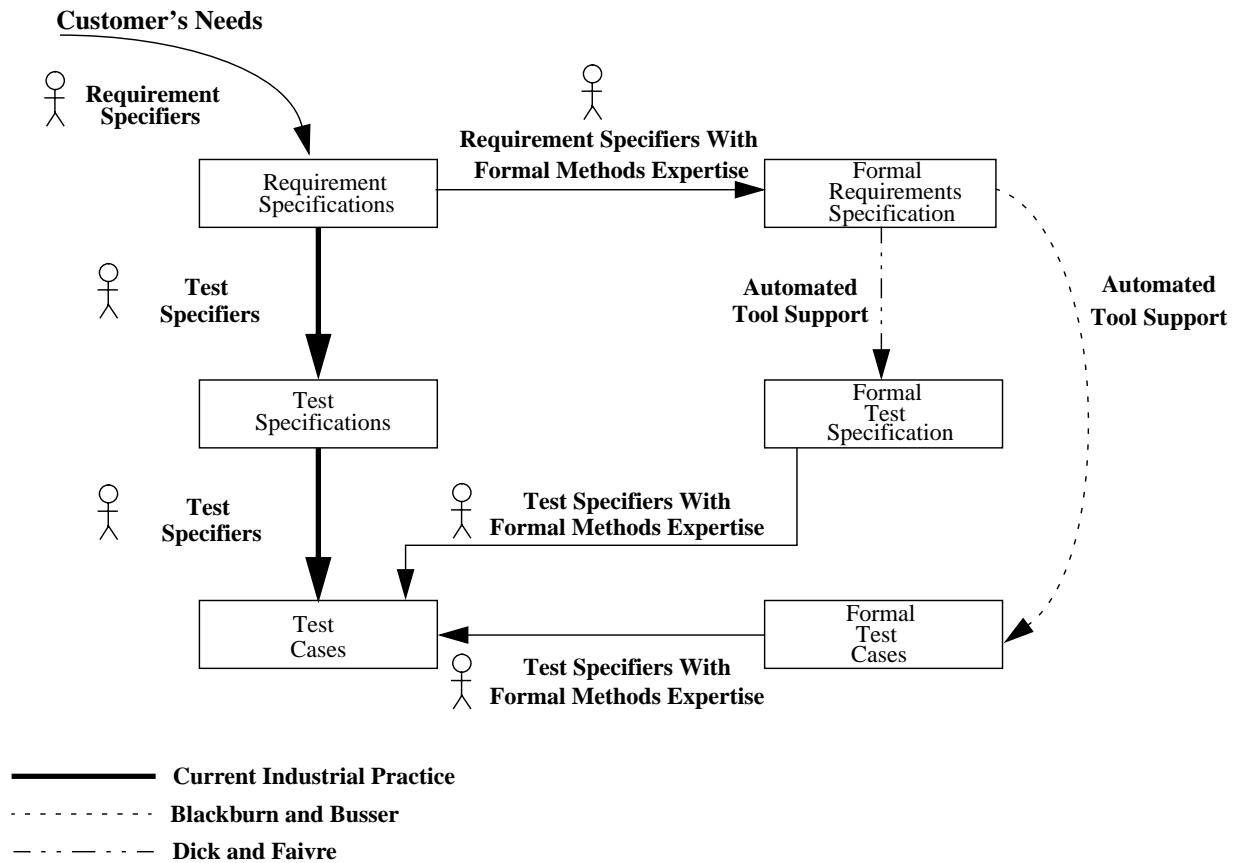


Figure 2. Alternative Techniques for Generating Test Cases

rectly reflects the given requirements specification

3. The requirements specification and its formal model must be synchronized as changes occur
4. The formally specified tests need to be interpreted and understood by those conducting the tests.

The third disadvantage is due to the necessity of having two requirements documents. While formal models are useful abstractions, they are limited in the information they can express. Hence, there is typically some information within the requirements specification that cannot be formally modeled. The last disadvantage arises because system-level testing requires that the system interact with its environment rather than an automated test harness.

The disadvantages of using formal techniques similar to those discussed here are brought about by the amount of modeling performed to produce the formal specification. A formal model captures two things: the logical relationships that define the different tests, and information that can be used to determine dependencies among conditions. This amount of formal detail is useful in eliminating infeasible tests which arise due to dependencies among conditions. At the unit level these types of dependencies are common, however, studies have shown that they are relatively rare at the system level [6,7,9]. This allows for a technique with less emphasis on formal modeling. To automatically generate tests at the system level only the logical relationships connecting conditions are required to be formalized.

PROPOSED TECHNIQUE

The analysis of the existing requirements-based testing techniques demonstrates the need for a new technique which removes the formal modeling of the requirements. The new formal technique is simpler to apply in industry than existing formal requirements-based testing techniques as it is not restricted for use by formal methods experts. The requirements notation is readable and tool support is available to assist the author in checking the

structure of the requirements specification. The proposed technique is suitable for automatically generating test specifications but not test cases. Producing instantiated test cases requires a substantial amount of formal modeling, which is a costly activity.

The automatic generation of the test specification uses a requirement transformation tool and a test specification generation tool (refer to Figure 3). The

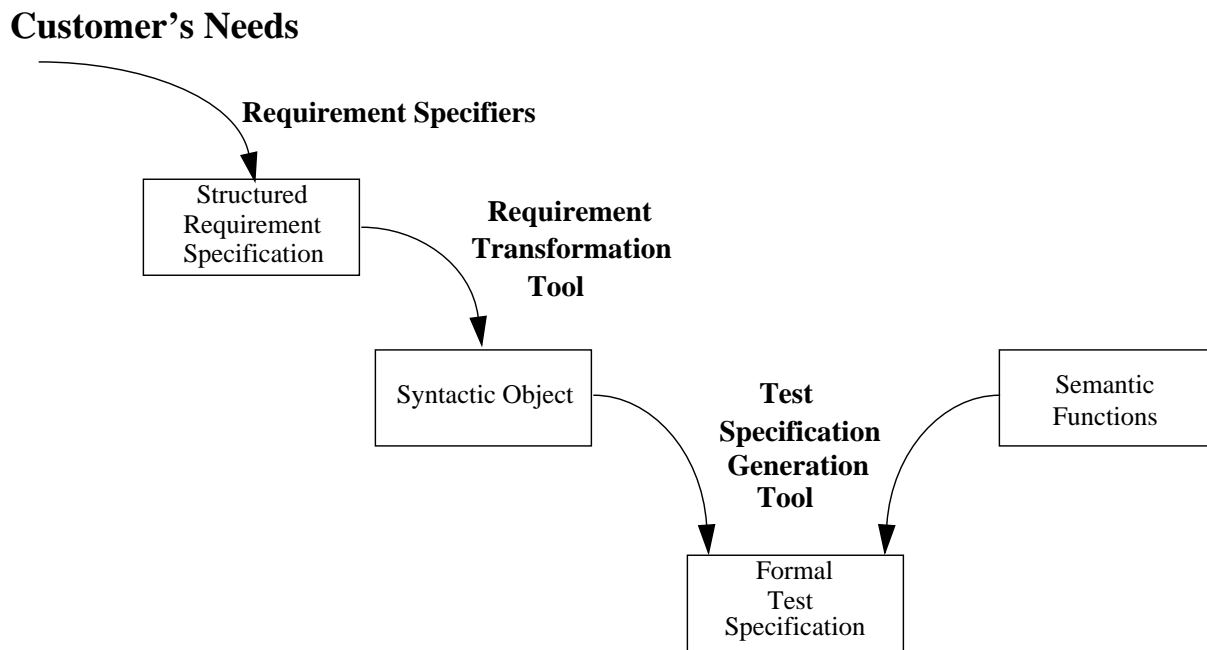


Figure 3. The Automatic Generation of Test Specifications

tools discussed below are implemented using Lex, Yacc, and the C programming language [12,13].

Requirement Transformation Tool and Example

The requirement transformation tool uses a requirements specification document written in a notation called "small" as its input¹. The notation reads like English, but has both a formally defined syntax and semantics. The syntax of "small" is

1. The notation is called "small" as it is a smaller, simpler version of a requirements specification notation under development.

defined in the Backus-Naur Form notation and its semantics are defined with semantic functions. This technique of using a syntactic object and the semantic functions is described in [10]. A complete requirements specification example is available in Appendix A to illustrate the readability of the notation. A partial example is provided below.

1. Upon 'book loan request', if ('this users privileges are suspended' or 'the book is restricted to in library use only'), the system shall 'return a rejection with reason'.

The requirement transformation tool processes the

requirements specification and generates a syntactic object. The syntactic object is used as one of the inputs to the test case generation tool. The other inputs are the semantic functions that formally define the "small" notation.

A requirements specification, written in "small" is scanned and parsed by the transformation tool. The parse tree generated in this step represents the abstract syntax of the notation. The parse tree is traversed and the syntactic object for the requirements specification is generated. The syntactic object is written in the S [11] notation.

Test Case Generation Tool and Example

The work presented in this paper uses Donat's test frame generation tool (TFG) [5,6,7,8,9] to calculate test specifications. As illustrated in Figure 3, TFG reads an S specification of the syntactic object and semantic functions and calculates test specifications using logical manipulation.

When calculating test specifications, the `or` connective causes TFG to produce two test specifications (referred to as *frames* by TFG). An example of these test specifications is given in Figure 4 .

```
--Test Frame 1:
```

Stimulus	Conditions	Criteria
'book loan request'	1) 'the book is restricted to in library use only'	1) 'return a rejection with reason'

```
--Test Frame 6:
```

Stimulus	Conditions	Criteria
'book loan request'	1) 'this users privileges are suspended'	1) 'return a rejection with reason'

Figure 4. Example Test Specifications

The *stimulus* is the action that triggers an operation. The *criteria* is the expected response of the system that must be observed for the system to pass the test. The *conditions* column contains the remaining conditions that are not designated as stimuli or criteria. When calculating test specifications, there is no distinction between stimuli and conditions. The stimulus designation is given to some conditions according to the requirements specification. TFG has been altered to allow this designation to be propagated to the TFG output.

Appendix B contains the complete set of test specifications generated from the library specification in Appendix A.

CONCLUSIONS

This paper introduces a new requirements-based testing technique that supports the automatic generation of test specifications from a requirements specification by capturing the logical structure of the requirements in a readable notation. The requirements are formalized, but not fully modeled in the new technique, as they are in existing, automated requirements-based testing techniques.

Removing the mathematical modeling of the requirements specification while retaining its formality is a significant advantage of this technique. The effort involved in developing a specification written in the "small" notation is

potentially much less than the effort necessary to write a fully modeled specification. The natural language style of the notation and the tool support for checking the syntax of the requirements minimizes training costs. Since “small” does not enforce a fully modeled specification and has a natural language style, the “small” specification can serve as the sole requirements specification. This eliminates synchronization between a requirements specification and its formal model. These factors reduce the cost of incorporating this formal technique in industrial projects.

The proposed technique automates a portion of the current test generation process. The test specifications generated are correct with respect to the requirements specification, assuming the correctness of the tools. This reduces the cost of generating and validating the test specifications.

FUTURE WORK

The next step in the work presented here is to develop the full version of the “small” requirements specification language. The full version is called the stimulus response requirements specification (SRRS) language [3]. Currently, the English-like concrete syntax for the language has been defined and a scanning, parsing, validation, and transformation tool is under development. In addition, an experiment is underway to objectively evaluate the SRRS language.

REFERENCES

1. Mark R. Blackburn and Robert D. Busser, “T-VEC: A Tool for Developing Critical Systems”, *Compass '96: Eleventh Annual Conference on Computer Assurance*, National Institute of Standards and Technology, Gaithersburg, Maryland, 1996, pp. 237-249.
2. Mark R. Blackburn, Robert D. Busser and Joseph S. Fontaine, “Automatic Generation of Test Vectors for SCR-Style Specifications”, *Compass'97: Twelfth Annual Conference on Computer Assurance*, National Institute of Standards and Technology, Gaithersburg, Maryland, 1997, pp.54-67.
3. Kendra Cooper, Jeffrey Joyce and Mabo R. Ito, *Stimulus Response Requirements Specification-Technique*. CICS Technical Report 97-001, The University of British Columbia, December 1997.
4. Jeremy Dick and Alain Faivre, “Automating the Generation and Sequencing of Test Cases from Model-Based Specifications”, editors J.C.P. Woodcock and P.G. Larsen, *Formal Methods Europe '93*, volume 670 of *Lecture Notes in Computer Science*, Springer-Verlag, 1993, pp. 268-284.
5. Michael R. Donat, “Automating formal specification-based testing”, In Michel Bidoit and Max Dauchet, editors, *TAPSOFT '97: Theory and Practice of Software Development*, 7th International Joint Conference CAAP/FASE, volume 1214 of *Lecture Notes in Computer Science*, Springer-Verlag, April 1997.
6. Michael R. Donat, *Automatically Generated Test Frames from an S Specification of Separation Minima for the North Atlantic Region*. Technical Report 98-04, Department of Computer Science, University of British Columbia, April 30, 1998.
7. Michael R. Donat, *Automatically Generated Test Frames from a Q Specification of ICAO Flight Plan Form Instructions*. Technical Report 98-05, Department of Computer Science, University of British Columbia, April 30, 1998.
8. Michael R. Donat, *A Discipline of Specification-Based Test Derivation*, Ph.D. dissertation, Department of Computer Science, University of British Columbia, September 1998.
9. Michael R. Donat and Jeffrey J. Joyce, *Applying an Automated Test Description Tool to Testing-Based on System Level Requirements*. INCOSE

'98: International Council on Systems Engineering, July 1998, Vancouver B.C., Canada

10. Jeffrey Joyce, "A Verified Compiler for a Verified Microprocessor", University of Cambridge Computer Laboratory, Technical Report No. 167, March 1989.
11. Jeffrey Joyce, Nancy Day and Michael R. Donat, "S: A Machine Readable Specification Notation based on Higher Order Logic, editors Thomas F. Melham and Juanito Camilleri, Higher Order Logic Theorem Proving and Its Applications, 7th International Workshop, volume 859 of Lecture Notes in Computer Science, Springer-Verlag, 1994, pp. 285-299.
12. Brian Kernighan and Dennis Ritchie, The C Programming Language, Prentice Hall, USA, 1988.
13. John Levine, Tony Mason, and Doug Brown, lex & yacc, O'Reilly & Associates, Inc., USA, 1992.

APPENDIX A. Library Example Requirements

The following is the library system specification written in small.

%{ rejection processing }%

1. Upon 'book loan request', if ('this users privileges are suspended' or 'the book is restricted to in library use only'), the system shall 'return a rejection with reason'.

2. Upon 'book loan renewal request', if ('this users privileges are suspended' or ('there are holds on this book' and 'this user is not the user with the first hold')), the system shall 'return a rejection with reason'.

3. Upon 'interlibrary loan request from another university', if ('the book title is restricted to in library use only' or 'the requesting university is not allowed to borrow a book'), the system shall 'return a rejection with reason'.

%{ acceptance processing }%

4. Upon 'book loan request' the system shall 'commit the book loan'.

5. Upon 'book loan renewal request' the system shall 'commit the book renewal'.

6. Upon 'loaned book returned' the system shall 'commit the book return'.

7. Upon 'borrowed book returned' ,if ('the book is overdue'), the system shall 'commit the outstanding fine for this user'.

8. Upon 'loaned book returned' ,if ('there are holds on this book'), the system shall 'notify the user with the first hold'.

9. Upon 'loaned book returned' , if ('the book is overdue' and 'the fine exceeds \$20.00') , the system shall 'commit the suspension of this users account'.

10. Upon 'loaned book returned' ,if ('the book is on interlibrary loan'), the system shall 'commit the return of the interlibrary loan book'.

11. Upon 'user fine payment', if ('the difference between the outstanding fine for this user and the amount of the fine payment received is less than or equal to \$20.00'), the system shall 'commit the unsuspension of this users privileges'.

12. Upon 'interlibrary loan request from another university', if ('the book is available'), the system shall 'commit the interlibrary book loan'.

APPENDIX B. Library Example Test Frames

Each of the following test frames is a test specification. The frames are grouped into test classes that share a common response criteria. The *stimulus* is the action that triggers an operation. The *criteria* is the expected response of the system that must be observed. The *conditions* column contains the remaining conditions that are not designated as stimuli or criteria. When calculating test specifications, there is no distinction between stimuli and conditions.

Test Class 1 (focused, DNF) antecedent DNF/CNF = 6/12 :

--Test Frame 1:

Stimulus	Conditions	Criteria
'book loan request'	1) 'the book is restricted to in library use only'	1) 'return a rejection with reason'

--Test Frame 2:

Stimulus	Conditions	Criteria
'book loan renewal request'	1) 'there are holds on this book' 2) 'this user is not the user with the first hold'	1) 'return a rejection with reason'

--Test Frame 3:

Stimulus	Conditions	Criteria
'interlibrary loan request from another university'	1) 'the book title is restricted to in library use only'	1) 'return a rejection with reason'

--Test Frame 4:

Stimulus	Conditions	Criteria
'interlibrary loan request from another university'	1) 'the requesting university is not allowed to borrow a book'	1) 'return a rejection with reason'

--Test Frame 5:

Stimulus	Conditions	Criteria
'book loan renewal request'	1) 'this users privileges are suspended'	1) 'return a rejection with reason'

--Test Frame 6:

Stimulus	Conditions	Criteria
'book loan request'	1) 'this users privileges are suspended'	1) 'return a rejection with reason'

Test Class 2 (focused, DNF) antecedent DNF/CNF = 1/1 :

--Test Frame 1:

Stimulus	Conditions	Criteria
'book loan request'		1) 'commit the book loan'

Test Class 3 (focused, DNF) antecedent DNF/CNF = 1/1 :
 --Test Frame 1:

Stimulus	Conditions	Criteria
'book loan renewal request'		1) 'commit the book renewal'

Test Class 4 (focused, DNF) antecedent DNF/CNF = 1/1 :
 --Test Frame 1:

Stimulus	Conditions	Criteria
'loaned book returned'		1) 'commit the book return'

Test Class 5 (focused, DNF) antecedent DNF/CNF = 1/2 :
 --Test Frame 1:

Stimulus	Conditions	Criteria
'borrowed book returned'	1) 'the book is overdue'	1) 'commit the outstanding fine for this user'

Test Class 6 (focused, DNF) antecedent DNF/CNF = 1/2 :
 --Test Frame 1:

Stimulus	Conditions	Criteria
'loaned book returned'	1) 'there are holds on this book'	1) 'notify the user with the first hold'

Test Class 7 (focused, DNF) antecedent DNF/CNF = 1/3 :
 --Test Frame 1:

Stimulus	Conditions	Criteria
'loaned book returned'	1) 'the book is overdue' 2) 'the fine exceeds \$20.00'	1) 'commit the suspension of this users account'

Test Class 8 (focused, DNF) antecedent DNF/CNF = 1/2 :
 --Test Frame 1:

Stimulus	Conditions	Criteria
'loaned book returned'	1) 'the book is on interlibrary loan'	1) 'commit the return of the interlibrary loan book'

Test Class 9 (focused, DNF) antecedent DNF/CNF = 1/2 :
 --Test Frame 1:

Stimulus	Conditions	Criteria
'user fine payment'	1) 'the difference between the outstanding fine for this user and the amount of the fine payment received is less than or equal to \$20.00'	1) 'commit the unsuspension of this users privileges'

Test Class 10 (focused, DNF) antecedent DNF/CNF = 1/2 :
--Test Frame 1:

Stimulus	Conditions	Criteria
'interlibrary loan request from another university'	1) 'the book is available'	1) 'commit the interlibrary book loan'