AC 2008-448: TEACHING BLACK-BOX TESTING TECHNIQUES THROUGH SPECIFICATION PATTERNS

Salamah Salamah, Embry-Riddle Aeronautical University, Daytona Beach

Ann Gates, University Of Texas - El Paso

Using Specification Patterns to Teach Black-Box Testing

Ann Q. Gates
Computer Science Dept., University of Texas at El Paso.

Salamah Salamah
Computer and Software Engineering Dept., Embry-Riddle Aeronautical University.

Abstract

Software verification is one of the most important activities in the software development cycle, and testing remains the most common approach to verification used in industry. The goal of black-box testing (functional testing) is to verify the system's adherence to specifications. The notion of patterns and scopes developed by Dwyer et al. provides a cohesive and rich set of examples to teach black-box testing strategies. A pattern describes a recurring software property, and a scope specifies the interval of program execution where a pattern must hold. A property specified using a pattern and scope combination has characteristics that must be satisfied if it is to hold. Based on these characteristics, there is a large set of behaviors that can be examined using black-box testing techniques. In a complementary fashion, the behaviors specified by patterns and scopes provide clear and simple examples that can enhance the understanding of these testing techniques. In this paper, we describe an approach and present general lessons and exercises that demonstrate how patterns and scopes can be used to teach boundary value analysis and equivalence class testing, which are two of the most commonly used black-box testing techniques. As a side effect of this approach, students are exposed to, and become familiar with, formally specifying system behavior.

1 Introduction

Testing remains the most natural and customary way of verifying a piece of software [6]. In software development, testing ranges from verifying the separate components of the system (methods, classes, etc.) to verifying the system as a whole (system testing). Testing approaches also differ in their techniques, which are based on the accessibility to the internal structure (code) of the system. In black-box testing techniques, test cases are defined based on system specification and without any consideration for the design or implementation of the system. Conversely, white-box techniques focus on testing the actual code of the system. As such, black-box testing is referred to as specification-based testing, and white-box testing is refereed to as implementation-based testing [6].

In this work, we focus on two of the most common black-box testing techniques: equivalence class testing, and boundary value analysis. Specifically, we discuss the Specification Pattern System (SPS) and the notions of patterns and scopes introduced by Dwyer et. al., [4] and how they are used to assist in defining system specifications. We then introduce an approach that uses patterns and scopes to teach the aforementioned testing techniques.

This paper is organized as follows; Section 2 provides the background information about the work. This includes a detailed description of SPS' patterns and scopes. In addition, descriptions of equivalence class and boundary value analysis testing techniques are provided in Section 2. The motivation behind using SPS to teach black-box testing is introduced in Section 3. Section 4 introduces the goals and desired outcomes of the introduced approach. Additionally, the section highlights the general approach and exercises used to achieve these goals and desired outcomes. The paper concludes with a summary and description of future work, followed by References.

2 Background

2.1 Specification Pattern System: Patterns and Scopes

Software specifications refer to properties that the system must adhere to. Specifications can be defined and used at the different stages of software development and can range in formality from completely informal (i.e., natural language) to completely formal (i.e., mathematical description). Informal specifications provide a simple mean by which stakeholders, who are, usually, not immersed in logic, can easily understand the desired system properties. On the other hand, formal specifications are more succinct and unambiguous. More importantly, formal specifications allow for the use of formal verification techniques such as model checking [3] and runtime monitoring [7].

Eliciting and defining system specifications, however, is a complex issue in software development, as it requires training and knowledge of the elicitation, analysis, and specification techniques. The Specification Pattern System (SPS) [4] provides patterns and scopes to assist the practitioner in formally specifying software properties. The work defined patterns and scopes after analyzing a wide range of properties from multiple industrial domains, i.e., security protocols, application software, and hardware systems. *Patterns* capture the expertise of developers by describing software behavior for recurrent situations. Each pattern describes the structure of specific behavior and defines the pattern's relationship with other patterns. Patterns are associated with *scopes*, which define the portion of program execution over which the property holds.

The main patterns defined by SPS are: $universality$, $absence$, $existence$, $precedence$, and $response$. The descriptions given below are excerpts from the SPS website [10].

- $Absence$: To describe a portion of a system's execution that is free of certain events or states.
- $Universality$: To describe a portion of a system's execution which contains only states that have a desired property.
- $Existence$: To describe a portion of a system's execution that contains an instance of certain events or states.
- $Precedence$: To describe relationships between a pair of events/states where the occurrence of the first is a necessary pre-condition for an occurrence of the second.
- $Response$: To describe cause-effect relationships between a pair of events/states. An occurrence of the first, the cause, must be followed by an occurrence of the second, the effect.

In SPS, each pattern is associated with a *scope* that defines the extent of program execution over which a property pattern is considered. There are five types of scopes defined in SPS (see [4] for a detailed description of the scopes):

- $Global$: The scope consists of all the states of program execution.
- $Before\ R$: The scope consists of the states from the beginning of program execution until the state immediately before the state in which proposition R first holds.

Table 1. Summary of Characteristics for SPS' Patterns

PATTERN	CHARACTERISTICS
Absence (P)	1) Event or condition P does not hold within the states defined by the scope of interest. 2) The *absence* property is also known as alarm.
Existence (P)	1) Event or condition P holds at least once within the states defined by the scope of interest. 2) The *existence* property is also known as eventually.
Universality (P)	1) Event or condition P holds in every state of the scope of interest. 2) The *universality* property is also known as safety or invariant.
(S) *Precedes*(P)	1) S holds before P holds, where S and P are events or conditions 2) S may hold several times before P holds 3) P does not hold before S holds 4) P may hold at the same state as S holds 5) If S holds, then P may or may not hold 6) The *precedence* property represents a cause-effect relation, where S denotes a cause and P denotes an effect 7) There is no effect P without a cause T
(S) *Responds* to (P)	1) P must be followed by S, where P and S are events or conditions 2) Some S follows each time that P holds 3) The same state at which S holds may follow two or more states at which P holds 4) S may hold at the same state as P holds 5) If S holds, then P may or may not hold at a previous state 6) The *response* property represents a cause-effect relation, where P denotes a cause and T denotes an effect 7) If cause P holds, then at some future state effect S holds

- *After Q*: The scope consists of the state in which proposition Q first holds and includes all the remaining states of program execution.
- *Between Q And R*: The scope consists of all intervals of states where the start of each interval is the state in which proposition Q holds and the end of the interval is the state immediately prior to one in which proposition R holds.
- *After Q Until R*: This scope is similar to the previous one except, if there is a state in which Q holds and proposition R does not hold, then the interval of the scope will include all states from and including the state where Q last holds until the end of program execution.

Figure 1 shows the scopes defined by SPS [10].

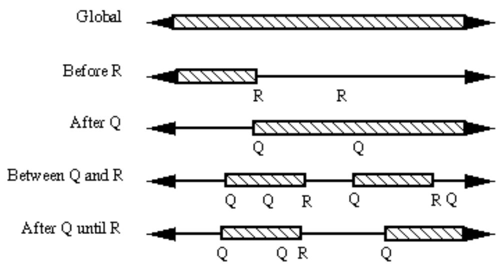

Figure 1. Scopes in SPS [10]

Tables 1 and 2 provide a detailed descriptions of the characteristics of these patterns and scopes respectively.

SPS is presented as a website [10] with links to descriptions of the patterns. The website provides a mapping of each pattern and scope combination into different formal specification languages. For example, the property "A request always triggers an acknowledgment, between the beginning

Table 2. Summary of Characteristics for SPS' Scopes

SCOPE	**CHARACTERISTICS**
$Global$	1) The scope denotes the entire computation. 2) The scope includes all the states in the computation. 3) The interval defined by the scope occurs once in a computation
$Before\ R$	1) The scope denotes a subsequence of states or events (an interval) that begins with the start of computation and ends with the state or event immediately preceding the event or state at which R holds for first time in the computation. 2) The interval does not include the state or event associated with R. 3) The interval defined by the scope occurs once in a computation. 4) One or more events (conditions) may be associated with R; a condition is a proposition and an event is a change in value of the proposition from one state to the next.
$After\ Q$	1) The scope denotes a subsequence of states or events (an interval) that begins with the first event or state at which Q holds and ends with termination of computation. 2) The interval includes the state or event associated with Q 3) The interval defined by the scope occurs once in a computation. 4) One or more events (conditions) may be associated with Q; a condition is a proposition and an event is a change in value of the proposition from one state to the next.
$Between\ Q$ and R	1) The scope denotes a subsequence of states or events (an interval) that begins when Q holds and ends with the state or event immediately preceding the event or state at which R holds. 2) Event or condition Q must hold and, at a different event or state in the future, R must hold. 3) The interval includes the state or event associated with Q 4) The interval does not include the state or event associated with R. 5) The interval defined by the scope may occur more than once in a computation. 6) Multiple intervals may be defined within an interval when Q holds more than once before R holds 7) One or more events (conditions) may be associated with Q and R
$After\ Q\ Until\ R$	1) The scope denotes a subsequence of states or events (an interval) that begins when Q holds and ends either with the state or event immediately preceding the event or state at which R holds, or begins when Q holds and ends with the termination of computation. 2) The interval includes the state or event associated with Q 3) The interval does not include the state or event associated with R 4) The interval may repeat during a computation. 5) If Q holds and R does not hold, the interval ends with termination of a computation. 6) The interval defined by the scope may occur more than once in a computation. 7) Multiple intervals may be defined within an interval when Q holds more than once before R holds 8) One or more events (conditions) may be associated with Q and R

of execution and system shutdown." can be described by the $S\,Responds\,to\,P$ pattern within the $Between\,Q\,and\,R$ scope, where S denotes "Acknowledgement is triggered.", P denotes "Request is made.", Q denotes "Execution begins.", and R denotes "System is shut down.". SPS provides mapping of pattern and scope combinations in multiple formal presentations such as Linear Temporal Logic (LTL) or Computational Tree Logic (CTL). For example, the specification for the above property in LTL, as provided by the SPS website, is:
$\Box((Q \wedge (\neg R) \wedge \Diamond R) \rightarrow (P \rightarrow ((\neg R)U(S \wedge \neg R)))UR)$.

Traces of Computation Through the reminder of this paper, we use the notion of traces of computation to describe test cases and the behaviors accepted by pattern and scope combinations. A trace of computation is a string representing a sequence of states that depicts the propositions that hold in each state. Each character in the string represents a state, and a dash (-) implies that no proposition is true at that state. A letter symbol, e.g., P, S, Q, and R, denotes that the proposition is true in the designated state. Displaying more than one letter between parentheses implies that the propositions represented by the letters are valid at that state. For example, in the trace of computation "- - Q - - R - - (SP) - -", Q is true in the third state, R is true in the sixth state, and S and P are both true in the ninth state.

2.2 Equivalence Class Testing

In general, it is impossible to test all possible operating conditions of a software system. As such, it is necessary to define and execute representative test cases to give enough evidence that the system (or the part of interest) exhibits the desired behavior, even in those cases that have not been tested.

In Equivalence class testing, test cases are partitioned into sets from which exemplars are selected. The technique assumes that test cases from a partition are equally likely to expose an error and, hence, only a representative test case is needed. This enables developers to significantly reduce the number of tests to run. In other words, the goal of equivalence class testing is to minimize the number of possible tests while at the same time have enough coverage to provide confidence in the correctness of the code or system being tested. For example, the two traces of computation "Q - P - - - - - -" and "Q - - - - - - P - -" belong to the same equivalence class in testing the $existence(P)$ pattern within the $After\ Q$ scope. In both tests, P occurs after Q, which holds in the first state. As a result, we only need to run one of the these tests.

Using equivalence class partitioning, we can derive test sets for each pattern and scope combination. For example, the test sets for the $Absence$ (P), and $Existence$ (P) patterns can be partitioned at the upper level into two sets: 1) P does not hold in any state of the interval(s) and 2) P holds in some state of the interval(s). The test sets for the $S\ precedes\ P$ ($S\ responds$ to P) pattern can be partitioned at the upper level into two sets: 1) S does not precede P (S does not respond to P) in any state of the interval(s) and 2) S precedes P (S responds to P) in the interval(s). The partitions for the $Universality$ pattern and for $Global$ scope are slight variations of the above partitions. The next level of partitions is based on the intervals defined by the scope as follows:

- $Before\ R$ ($After\ Q$)-Partition 1:
  - R (Q) holds in the first state
  - R (Q) holds in the last state
  - R (Q) holds in other states

- the interval is not built

- $Before\ R$ ($After\ Q$)-Partition 2:
  - Q holds in the first state (not applicable for R)
  - R (Q) holds in the last state
  - R (Q) holds in other states
- $Between\ Q\ and\ R$ ($After\ Q\ Until\ R$)-Partition 1:
  - the interval is not made
  - a single interval is made and
    - Q holds in the first state
    - R holds in the last state
  - multiple intervals are made
  - nested intervals are made
- $Between\ Q\ and\ R$ ($After\ Q\ Until\ R$)-Partition 2:
  - a single interval is made and
    - Q holds in the first state
    - R holds in the last state
    - Q and R hold in other states
  - multiple intervals are made
  - nested intervals are made

2.3 Boundary Value Analysis

The boundary value analysis strategy is a method by which input values are chosen to lie on data extremes. Example values are those representing the maximum value, minimum value, and the values just before and after boundaries. The idea is that, if a the system works correctly for these special values then it will work correctly for all values in between.

The test cases defined in each of the sub-partitions in Section 2.2 are based on boundary value analysis strategies. For example, consider the following three test cases that are associated with $Absence$ - $After\ Q$ and the sub-partition "Q holds in other states" under Partition 2:

1. - - - - - - - - - P Q - - - - - - - - - -
2. - - - - - - - - - - (QP) - - - - - - - - - -
3. - - - - - - - - - - Q P - - - - - - - - -

Test case 1 is valid, and the last two test cases are not valid. Test case 2 checks that P holds in the first state in which Q becomes true. Test case 2 is also used for a $Before\ R$ test case, where Q is replaced by R. In this case, the test would be valid. Tables 3 and 4 give the complete set of test cases for the $Absence$-$Before\ R$ and $Precedence$-$Between\ Q\ and\ R$ patterns respectively. The complete set of test cases for all patterns and scopes can be found in [8].

Notice that in using the two testing techniques, test cases that check conditions that can be verified through other test(s) should be eliminated. For example, in the $S\ precedes\ P$ pattern within the $Between\ Q\ and\ R$ scope, the following test case is not needed: $---Q--(PS)---R$. There are two issues that are being checked by this test case; it checks that precedence is upheld when P and S hold at the same state, and it checks that the interval is built even when R occurs in the last state of the computation. These two conditions are covered by test cases 2(a)(vi) and 2(b)(iii) respectively as given in Table 3.

3 Motivation

The main contribution of this paper is to show how the aforementioned notions of patterns and scopes can be used in teaching black-box testing techniques. The motivation behind this use of patterns and scopes is twofold; 1) The characteristics of these patterns and scopes seem to be a perfect fit with the notions of boundary values and equivalence classes, and 2) the students will be introduced to formal specifications of software properties.

Applicability of Patterns and Scopes: Considering the scope characteristics in Table 2 and Figure 1, we can find a perfect symmetry with the conditions emphasized in testing boundary values. Scopes, formally[1], define the states of interest within system execution (i.e., define boundaries where a pattern is to hold). As a result, scopes characteristics provide sufficient examples to explain what constitutes a boundary value. For example, using the scope $Before\ R$, one can test whether the pattern of interest is upheld if that pattern holds immediately before the state where R holds, and at the same state as R, at the state immediately after R. In a similar fashion, patterns can be used to better understand equivalence class testing. For example, using the $S\ Responds$ to P pattern, we can think of multiple equivalence classes to choose tests from:

- P never holds, and S holds,
- P and S never hold,
- P holds and S never holds,
- P holds and S holds before P,
- P holds and S holds after P,
- P and S hold at the same state (this might be considered for boundary analysis),
- ...

It is important to note that scopes can also be used to explain equivalence classes and patterns can be used to explain boundary analysis. For example, the $Between\ L$ and R scope provides multiple equivalence classes to test whether the scope is built or not. Some of these classes are R holds before L, L holds at the same state as R, and L holds and is followed by multiple states where R holds. It is because these patterns and scopes are rich in characteristics that they provide ample combinations of situations that that can be used to teach and test the two black-box testing techniques. The SPS website [10], contains more than 500 properties that are defined by patterns and scopes. These properties are examples that instructor can use as real life situations.

Introducing Students to Formal Specifications: Formal specifications allow for the precise and unambiguous definition of software properties. They are also required in the use of formal verification techniques such as model checking [3] and runtime monitoring [7], which are effective approaches for improving the dependability of programs. These techniques check the correctness of the system against specifications written in a formal specification language. Additionally, formal

[1]SPS provides a formal description for each pattern and scope combination in multiple languages.

Table 3. Equivalence Classes for Precedence Between Q and R

EQUIVALENCE CLASSES ON P	TEST CASES WITH BOUNDARY ANALYSIS AND EQUIVALENCE CLASSES ON R	EXPECTED RESULT
S Does not precede	1. The interval is not made:	
P in any state in	a) - - - - - - - - - P - - - - - - - - - - R	1a. Valid
the interval	b) - - - - (QRP) - - - - - - - - - - - - - - - -	1b. Valid
	c) - - - - Q - - - - - - - P - - - - - - - -	1c. Valid
	d) - - - - - - - - - - - - P - - - - - - - -	1d. Valid
	e) R - - - - - - - P - - - - Q - - - - - - -	1e. Valid
	2. A single interval is made:	
	a) Q holds in first state:	
	i. Q - - - P - - - - R - - - - - - - - - - -	2ai. Not valid
	ii. Q - - - S - - - - R - - - - - - - - - - -	2aii. Valid
	iii. Q - - P - - S - - R - - - - - - - - - - -	2aiii. Not valid
	iv. Q - - - - (RP) - - - - - - - - - - - - - - -	2aiv. Valid
	v. Q - - - - - R P - - - - - - - - - - - - -	2av. Valid
	vi. Q - - - (PS) - - - - R - - - - - - - - - - -	2avi. Not valid
	b) R holds in last state:	
	i. - - - - - - - - - - - P Q - - - - - - - R	2bi. Valid
	ii. - - - - - - - - - - - - - - - - Q - - - (RP)	2bii. Valid
	iii. - - - - - - - - - - - Q - - P - - - - - R	2biii. Not valid
	iv. - - - - - - - - - - - Q - - P - - S - - R	2biv. Not valid
	v. - - - - - - - - - - - Q - - S - - - - - R	2bv. Valid
	3. Multiple intervals are made:	
	a) Q - - - - R - - - - - Q - - - P - - R - -	3a. Not valid
	b) Q - - - - R - - P - - - Q - - - - R - - -	3b. Valid
	4. Nested intervals are made:	
	a) - - - - Q - - - - - - Q - - P - - R - - -	4a. Not valid
	b) - - - - Q - - P - - - Q - - - - - R - - -	4b. Not valid
	c) - - - Q - - - Q - - - R - - P - - R - - -	4c. Valid
	d) - - Q - - S - - Q - - - P - - - - R - - -	4d. Not valid
S precedes P in	5. A single interval is made:	
the interval	a) Q holds in first state:	
	i. Q S - P - - - - - R - - - - - - - - - - -	5ai. Valid
	ii. (QS) P - - - - R - - - - - - - - - - - - - -	5aii. Valid
	iii. Q - - - S - P R - - - - - - - - - - - - -	5aiii. Valid
	b) R holds in last state:	
	i. - - - - - - - - - - - - - - Q S P - - - R	5bi. Valid
	ii. - - - - - - - - - - - - - - (SQ) P - - - - R	5bii. Valid
	iii. - - - - - - - - - - - - - - Q - - S - P R	5biii.Valid
	c) Q-R hold in other states:	
	i. - - - - - - - - Q - - S - - P - - R - - -	5ci. Valid
	ii. - - - - - - - - - Q - - S - - P - R - - -	5cii. Valid
	6. Multiple intervals are made:	
	a. Q - S - P - - R - - - - Q - - R - - Q P R	6a. Not valid
	b. Q - S P - - R - Q - - S - P - R - Q - P -	6b. Valid
	7. Nested intervals are made:	
	- - - Q - - S - - Q - - - P - R - - - - -	7. Not Valid

specifications can be used to capture pre and post-conditions of methods and to generate test cases [2]. Furthermore, because formal specifications are mathematically based, it is possible to prove properties about the specifications themselves [5]. Because of the advantages of formal methods and formal verification, it is important that students learn the basics of formal specifications and are able to apply them to define system properties.

4 Educational Component for Teaching Black-Box Testing

4.1 Goals and Outcomes

The goal of the lessons described in this section are to teach students how to: 1) use SPS in support of specifying system properties, and 2) be able to develop test cases for system properties (defined by patterns and scopes) using boundary value analysis and equivalence class testing techniques. This section describes activities that support the attainment of these goals.

The educational component described in this paper can be used in any computer science or software engineering courses where the topic of testing is discussed. The component described here is approximately six hours in length (includes tutorials on SPS).

The component outcomes, given below, are separated using Bloom's taxonomy [1], where level 1 outcomes represent knowledge and comprehension outcomes (those in which the student has been exposed to the terms and concepts at a basic level and can supply basic definitions.); level 2 outcomes represent application and analysis (those in which the student can apply the material in familiar situations, e.g., can work a problem of familiar structure with minor changes in the details); and level 3 represent synthesis and evaluation (those in which the student can apply the material in new situations).

- 1-1. Students will be able to describe the simple behaviors defined by SPS patterns and scopes.
- 2-1. Students will be able to identify the appropriate pattern and scope associated with a property and to apply them to generate a specification using SPS.
- 2-2. Given a trace of computation, the students will be able to determine if the property (defined by a pattern and scope combination) holds under the trace.
- 2-3. Given a pattern and scope combination, the students will be able to identify the boundary conditions for this combination.
- 2-4. Given a set of traces of computations (test cases) for a pattern and scope combination, the students will be able to identify redundant traces (ones that belong to the same equivalence class).
- 3-1. Students will be able to specify a property using a pattern and scope combination, and will be able to define equivalence class and boundary value analysis test cases to test the property.

The lessons for Outcome 1-1 are not given in this paper. The focus will be on the rest of the outcomes.

4.2 Basic Approach and General Exercises

Following an introduction to testing and system specifications and a tutorial on SPS, the students will be given a hands-on exercise in which they use SPS to specify a series of properties. The lesson focuses on Outcome 2-1 and teaching students how to use SPS to specify patterns and scopes to generate formal specifications[2]. There are 25 possible SPS pattern and scope combinations. The concentration should be on the use of the response pattern, which is one of the most commonly used patterns in property specification [4].

Exercise 1: *For each of the properties below, use SPS to: (1) define the property pattern, (2) define the property scope, (3) map the propositions used in the pattern and scope to the appropriate phrases in the property description, and (4) generate the LTL formula for the pattern and scope.*

1. *The OK button is enabled after the user enters correct data.*
2. *A request always triggers reply between start of execution and system shutdown.*
3. *No work will be scheduled before execution.*

Exercise 2: Exercise 2 focuses on teaching students the concept of traces of computations, and how they can be used to visualize the behaviors accepted or not accepted by a system property. In addition, the subtle characteristics of patterns and scopes (Tables 1 and 2 in Section 2) are described prior to the exercise. This exercise targets Outcome 2-2. Instructions for Exercise 2 follow:

Consider the following property: When a connection is made to the SMTP server, all queued messages in the OutBox mail will be transferred to the server. This property can be described using the $Existence(P)$ scope within the $Before\, R$ scope, where P "a connection is made to the SMTP server" and R is "all queued messages in the OutBox mail are transferred to the server". For each of the traces of computations given below, state the expected result of the test (Valid or Invalid). Explain your answer.

1. - - - - - - - - - - -
2. - - - -(PR)- - - - - - -
3. - - - - P R - - - - - -
4. - - - - R P - - - - - -
5. - - - - - - - - - - - R
6. R - - - - - - - - - - -
7. - - R - - P - - - R - -
8. - - - - P - - - - - - -

Exercise 3: After completing exercises 1 and 2, the students should have sufficient understanding of the SPS patterns and scopes and are familiar with the notion of traces of computations to represent test cases. The next lessons introduce equivalence classes and boundary value analysis testing techniques. Exercise 3 aims at testing students' ability to define boundary conditions for a certain pattern and scope combination. Exercise 3 targets Outcome 2-3. Instructions for Exercise 3 follow:

Consider the following property: "The first method called will be the connect method." This property can be described using the Absence (P) pattern, within the Before R scope, where P is "Another method is called", and R is "Connect method is called". Define the boundary conditions to be tested

[2]The choice of the formal language to represent the property is irrelevant here and is up to the instructor. In this paper, we chose LTL, simply because of its common use to represent software properties.

in verifying the system behavior in regard to this property.

Note that boundary conditions should be defined in similar fashion as in Tables 3 and 4 in Section 2. Example boundary conditions for the previous property include P and R hold in the same state, P holds in the state imminently before R, P holds in the state immediately after R, R holds in the first state, and R holds in the last state.

Exercise 4: While Exercise 3 tests the students understanding of boundary value analysis, this exercise tests their understanding of the equivalence class testing techniques. Exercise 4 targets Outcome 2-4. Instructions for Exercise 4 follow:

Consider the S Responds to P pattern within the After Q scope. For each of the following traces of computation: 1) State the expected result of the test (Valid or Invalid), and 2) State which of these traces of computations are redundant (i.e., there's another trace in the set that belongs to the same class.)

1. Q - - - - - - - P S - - - - - - - -
2. - - - - - S - - - - P - - - Q - - -
3. (QP) - - - - - - - - - - - - - - - - -
4. - - Q - - - - P - - - - - - S - - -
5. - - - - - - P Q - - - - - - - - - -
6. - - S - - P - - - - - Q - - - - - -
7. - - - - - - - (PQS) - - - - - - - - - -
8. - - - - - S (PQ) - - - - - - - - - - -
9. - - - - - - P (QS) - - - - - - - - - -
10. - - - Q - - - P - - - S - - - - - -

Exercise 5: After completing the previous exercises and other assignments determined by the instructor, the students should be able to define test cases in the form of traces of computations to validate their pattern and scope generated properties. Exercise 5 checks their ability to satisfy Outcome 3-1. Instructions for Exercise 3 follow:

For each of the properties given below, specify the appropriate pattern and scope combination. In addition, define a minimal set of traces of computations to validate your choice. The Properties are:

1. *When a connection is made to the SMTP server, all queued messages in the OutBox mail will be transferred to the server.*
2. *When the name of a mailbox is double-clicked, the mailbox will be opened.*

5 Summary and Future Work

Because testing remains the major form of verification used in industry, it is important that students are exposed to the different testing techniques. It is also important that these students are familiar with the importance of formal specifications and their uses. In this paper, we introduced an approach the links these two ideas together. We introduced an approach that assists in the understanding of system properties as well as in understanding two of the most common black-box testing techniques: equivalence classes and boundary value analysis. We also provided exercises based on the use of property patterns and scopes to enhance students' understandings of these testing techniques and of system specifications. The approach and exercises can be used in Computer Science and Software Engineering courses that introduce testing.

Evaluation of the Approach: This semester (Spring 08), we began using the introduced approach in the capstone course in Computer Science at the University of Texas at El Paso, and an introductory course in Software Engineering at Embry-Riddle Aeronautical University. The initial observation is that students are showing more interest in the topic of testing when they are given real life examples form the SPS website [10]. In each of the two courses, we have explained the ideas of black-box testing and patterns and scopes. We also gave the students each of the exercises introduced in the paper. Unfortunately, at the time of the witting of this paper, we have not yet tested their understanding of the testing techniques. In each course, there will be two exams for the reminder of the semester (second midterm and the final). Each of these will have questions on black-box testing. These questions will be the same questions given to students in previous semesters. Once we have the results of these exams, we will analyze the performance of the students and compare them against those in previous semesters. We will report the results in the conference presentation of the paper at the ASEE conference, as well as in a future publication.

Future work also includes applying this approach and defining similar exercises for other black-box testing techniques, such as the cause-effect graph testing technique. Additionally, we are in the process of developing a tool that can be used to run the actual test cases represented as traces of computation. The tool will allow the students to compare their expected results with the actual results of the tests. The tool is based on the technique of using a model checker to validate formal specifications [9].

References

[1] Bloom, B.S., "Taxonomy of Educational Objectives: The Classification of Educational Goals," Susan Fauer Company, Inc., 1956, pp. 201-207.

[2] Cheon Y., Leavens G. T., "A Simple and Practical Approach to Unit Testing: The JML and JUnit Way", European Conference on Object-Oriented Programming, ECOOP, Malaga, Spain, June 2002.

[3] Clarke, E., Grumberg, O., and D. Peled. *Model Checking*. MIT Publishers, 1999.

[4] Dwyer, M.B., Avrunin, G.S., and Corbett, J.C., "Patterns in property specifications for finite-state verification," *Int. Conf. on Software Engineering, ICSE*, Los Angeles, CA, May, 1999, pp. 411–420.

[5] Hall, A., "Seven Myths of Formal Methods," *IEEE Software*, September 1990, 11(8).

[6] Ghezzi, C., Jazayeri, M., and Mandrioli, D., *Foundamentals of Software Engineering*. Prentice Hall, 2002.

[7] Gates, A., Roach, S., et al., "DynaMICs: Comprehensive Support for Run-Time Monitoring," *Runtime Verification Workshop*, Paris, France, July 2001, pp. 61-77.

[8] Salamah, S., "Supporting Documentation for the SPS-Prospec Case Study," UTEP-CS-05-14, the University of Texas at El Paso, April 2005.

[9] Salamah, S., Gates, A., "A Technique for Using Model Checking to Teach Formal Specifications," *to appear in the proceedings of the 21st IEEE-CS Conference on Software Engineering Education and Training* April 2008.

[10] Spec Patterns, http://patterns.projects.cis.ksu.edu/, January 2008.