NISTIR 6114

Experience Report: Comparing an Automated Conformance Test Development Approach With a Traditional Development Approach

Alan Goldfine
Gary Fisher
Lynne Rosenthal

Software Diagnostics and Conformance Testing Division
Information Technology Laboratory
National Institute of Standards and Technology

April 1998

United States Department of Commerce
Technology Administration
National Institute of Standards and Technology

NISTIR 6114

Experience Report: Comparing an Automated Conformance Test Development Approach With a Traditional Development Approach

Alan Goldfine
Gary Fisher
Lynne Rosenthal

Software Diagnostics and Conformance Testing Division
Information Technology Laboratory
National Institute of Standards and Technology

April 1998

U.S. DEPARTMENT OF COMMERCE
William M. Daley, Secretary

TECHNOLOGY ADMINISTRATION
Gary R. Bachula, Acting Under Secretary for Technology

NATIONAL INSTITUTE OF STANDARDS AND TECHNOLGY
Raymond G. Kammer, Director

NISTIR 6114

Experience Report: Comparing an Automated Conformance Test Development Approach With a Traditional Development Approach

Alan Goldfine
Gary Fisher
Lynne Rosenthal

Information Technology Laboratory
National Institute of Standards and Technology

Abstract

This paper describes a project at the National Institute of Standards and Technology (NIST) that investigated the effectiveness of using automated software test generation methods to help develop conformance tests for implementations of specifications of software standards. Traditionally, such conformance tests have been developed by manually coding and testing the test source code. Recently, several technologies that automate parts of the software test development process have appeared. This paper describes a case study that compared the use of a particular automated method (the Assertion Definition Language (ADL), developed by Sun Microsystems, Inc.) to develop conformance tests, compared with the use of traditional methods, when both methods were applied to the same standard software specifications.

Keywords: ADL; Assertion Definition Language; automated testing; conformance testing; software testing.

Table of Contents

1. Background

The Information Technology Laboratory (ITL) of the National Institute of Standards and Technology (NIST) has a major responsibility to provide technical leadership for the development of conformance tests for implementations of specifications of software standards. Conformance testing is normally done through falsification testing, where an implementation that claims conformance to a specification is tested with various combinations of legal and illegal inputs, and the resulting output is compared with the "expected results." Traditionally, the test developer, after a detailed examination and analysis of the specification, manually constructs the test requirements, the test code, the inputs, and the expected results. This approach, in particular the coding of the tests and input combinations, is extremely labor-intensive and expensive. NIST has been searching for ways to improve the process.

Recently, several technologies have appeared that attempt to automate parts of the software test development process [1], [2], [3]. In particular, the Assertion Definition Language (ADL), developed by Sun Microsystems, includes a formal assertion language that can describe the behavior of program interfaces, a supporting test data description language, and a translation system that generates C language source code from specifications written in the assertion and test data languages [4]. If the assertions and data descriptions are written appropriately, the generated source code can be viewed as a suite of conformance tests for the given program interface.

2. Project Strategy and Design

Although ADL has begun to be used in several production projects, we know of no earlier studies that have investigated whether or not the use of such an automated technique really does improve the test development process. The project at NIST described in this paper is such a study. We developed, for the same software specification, equivalent sets of conformance tests using a) an automated approach and b) the traditional manual approach. We hoped that by measuring how quickly the conformance tests were developed using each approach, we would shed some light on the effectiveness of the automated approach.

The design goal was to keep the project simple by concentrating only on comparing the effectiveness (that is, speed of development) of the two approaches. Consequently, we limited the scope of the project by choosing as the application a small subset of a standard software specification to which the two approaches could be applied. Additionally, it was clear that some development tasks would be the same regardless of the approach. Since these tasks (for example, determining the requirements or assertions on which to base the test cases) would not affect the desired comparison, we factored them out.

The project can be summarized as follows:

- We investigated existing automated test generation methods, selected one (ADL version 1.1) for use, acquired it, and installed it on existing hardware. We felt that ADL was appropriate for the task, was compatible with available hardware, and was freely available.

- We selected an appropriate subset of a software standard (the C language interface to POSIX [5]) as the application. Of the various specifications that we considered, this collection of POSIX functions best satisfied the selection criteria. In particular, the POSIX standard includes an official list of assertions that, for each function, defines "conformance" [6]. (Appendix A contains our application selection criteria and the list of applications that we considered.)

- We selected the people who would perform the programming. Essentially, the first two authors of this paper split the task, with one doing ADL work and the other doing traditional coding during the first half of the study, then switching roles for the second half. Neither of us knew ADL before the start of the project; our respective learning curves are reflected in the Results. We both knew how to program in C before the project, although neither of us was a professional C programmer.

- We then developed conformance tests for four POSIX functions (`chdir`, `umask`, `rmdir`, and `chmod`, in that order). Part of this development was the validation of the tests by applying them to two candidate implementations, one of which was certifiably POSIX compliant.

- For each appropriate step we recorded the time required to complete that step.

- At the conclusion of the study we compared, according to our measure, the effectiveness of the two approaches.

2.1 Test Development

To help ensure that the strategy and design for our project was sound, we selected a typical function in the application collection, `getcwd`, and, using the two respective approaches, developed trial run conformance tests for that function. The results of the trial run underscored the importance of focusing on the basic objective of the study, which was to compare the use of two different approaches—essentially the use of two different languages—*to accomplish the **same** programming task.* We were quickly reminded that the specific characteristics of the individual experimenters (as opposed to the characteristics of the automated vs. traditional approaches), would be extraneous to the study and would serve only to skew the results. We therefore strove to design the study to factor out the following characteristics:

- different levels of programming skill, in both ADL and C
- different initial levels of knowledge of the application specification and possible different

interpretations of the application specification details
- the potential to develop different design strategies.

In particular, the programmers deliberately worked closely together in all areas other than the actual programming. We carefully selected the sequence in which we would process the functions. We chose, for each function, an explicit subset of the official assertions to test — we excluded from our scope any assertion that corresponded to POSIX functionality that appeared to be unusually complex or tricky (the study was not supposed to be a test of an individual programmer's POSIX knowledge or design ingenuity). We agreed in advance on the precise collection of tests and test data that needed to be developed to validate the assertions for each function. (Appendix B contains the planned tests and test data for one of the POSIX functions.) Only after all these steps were complete did we begin to implement, using our respective approaches, the jointly developed design. Even during the study proper, we continued to discuss design issues, if these issues were applicable to both the automated and hand-coding approaches and were not directly related to details of the use of the respective languages. (Appendix C contains both the ADL and C versions of the conformance code written for one of the POSIX functions.)

2.2 Comparing the Two Approaches

Early in the project we developed an initial list of measurement criteria that included various approaches to comparing conformance test development in ADL with development using a traditional approach. In the end, though, the deliberately narrow goal of comparing two approaches to building the same application led us to concentrate on ensuring that the two resulting applications would, in fact, be the same. In this way we eliminated measurements of software quality from our consideration, and simply measured the time that it took each programmer to accomplish the specified work.

We interpreted the time narrowly by only counting the time of the study itself. We didn't count the time spent planning the project, selecting the application, etc. On the other hand, the timing figures included all mistakes, "false starts," etc., in the appropriate categories.

3. Results

Programmer #1 recorded 383 preliminary hours spent learning ADL; programmer #2 recorded 107 hours.

Table 1 summarizes the "comparison" results of the study. The times are given in hours.

Table 1: Time required to write, test, revise and validate the test programs

		Times for the Automated Approach				Times for the Traditional Approach		
POSIX function	Prog-rammer	ADL coding	C coding	Testing of 2nd implementation	Total time	C coding	Testing of 2nd implementation	Total time
`chdir`	# 1	10 1/2	25	3	38 1/2			
	# 2					31	2	33
`umask`	# 1	5	9	1	15			
	# 2					6	5	11
`rmdir`	# 1					13	1	14
	# 2	5	22	1	28			
`chmod`	# 1					31	1	32
	# 2	19	31	1	51			
TOTAL		39 1/2	87	6	132 1/2	81	9	90

(Appendix D contains the complete filled-in form, which provided the basis for Table 1.)

4. Observations

Our basic observation is that the use of ADL did not reduce the time needed to develop conformance tests. We can identify several possible reasons for this.

1. A somewhat complicated and non-intuitive tool such as ADL has a significant learning curve. Although we attempted to remove learning time from the direct comparison, even the times recorded for the test itself inevitably included a learning component. If we had continued the study to include additional POSIX functions, a) the ADL coding might well have gotten easier, and b) the included learning time would have been amortized over a longer total time frame.

2. On a related note, one of the advantages claimed for ADL is the ability to build re-usable libraries of symbolically specified and manipulated test data. Our study was limited to four POSIX functions, so we had little opportunity to benefit from re-usability. Had we included more functions, and taken care along the way to build an explicit, consistent library of test data, the ADL development might have been more effective.

3. However, reasons 1 and 2 may be of little overall importance. It turns out that the current

ADL system automates only a relatively small part of the total job of developing conformance tests. As can be seen from Table 1, the proportion of work that we did in ADL itself (writing the assertions and the symbolic specifications of the test data points) was small in comparison to the coding, in traditional C, of the necessary support functions. These support functions included:

- the specification of the actual test data points,
- initializations, file opens and closes, and other housekeeping details, and
- the other auxiliary and utility functions (e.g., the parsing and manipulation of filenames) that were needed to support the evaluation of the assertions.

While the equivalent of the support code would have to be developed anyway during the traditional approach, the magnitude of the C coding did tend to swamp whatever advantages ADL provided.

4. Thus, we come to the reason that we think actually overshadowed all others. We had consistent difficulty with the interplay between the C code generated by ADL and the supporting C code that we wrote ourselves. The final mixed programs crashed frequently and were notoriously difficult to debug. This isn't a criticism of ADL, since the crashes invariably turned out to be due to the user's misunderstanding of the subtleties of ADL. However, the source code for these generated programs was either unavailable to, or unreadable by, the user. While this is part of the design of ADL, and perhaps inherent in the nature of generated code, it definitely highlighted a weakness of the current version of ADL.

5. Conclusions and Future Research

The results of this study show that a reasonable question exists regarding the assumption that automated tools provide a more effective means of developing conformance tests for program interfaces than do traditional approaches. Although our first results are largely negative, a larger scale study is needed to adequately take into account the learning curve and re-usability issues. Other automated tools and techniques need to be investigated, and perhaps more sophisticated metrics developed to better measure test development effectiveness. We look forward to the development of future, more fully automated test tools.

REFERENCES

[1] Chang, J., Richardson, D., and Sankar, S., "Structural Specification-based Testing with ADL," *Proceedings of the 1996 International Symposium on Software Testing and Analysis*, ACM Press, 1996.

[2] Leathrum, J., and Liburdy, K., "Formal Test Specifications in IEEE POSIX," *Computer Standards and Interfaces*, 17(1995), pp. 603-614.

[3] IFAD VDM-SL Toolbox web page: http://www.ifad.dk/products/toolbox.html.

[4] Sun Microsystems ADL project web page: http://www.sunlabs.com/research/adl.

[5] ISO/IEC 9945-1: 1990 (E), IEEE Std 1003.1-1990, *Information Technology — Portable Operating System Interface (POSIX) — Part 1: System Application Program Interface (API) [C Language]*, published by the Institute of Electrical and Electronic Engineers, 1990.

[6] IEEE Std 2003.1-1992, *IEEE Standard for Information Technology — Test Methods for Measuring Conformance to POSIX — Part 1: System Interfaces*, published by the Institute of Electrical and Electronic Engineers, 1992.

Appendix A — Application Selection

Selection Criteria

— The selected application must be in the form of (or easily repackaged as) a collection of C functions.
— The selected application should not be a complex, highly integrated language or system with a long learning curve.
— The selected application as a whole must be manageable, discrete, and scaleable, to allow the selection of an appropriate amount of work for Phase 1.
— The individual functions of the application should be (subjectively) at the "right" level of complexity.
— There must be available suitable implementations of the application. "Suitable" here includes the requirement that the functions of the application be available as .o and/or .c files.
— To attempt to level the playing field, the task to develop conformance tests using ADL must be done by a different development group than the task to develop the parallel test suite using the traditional, hand coding approach.
— The two development groups should start out with similar levels of background knowledge in the selected application.
— The two development groups should start out with similar levels of background knowledge in their respective approaches (i.e., developing tests in ADL vs. developing tests in the language used in the hand coding).
— The conformance tests developed for the selected application should be useful beyond the project. (?)
— The selected application should be "forward looking." (?)

Specifications Considered

— Ada
— ANSI C Library Functions
— CGM
— PHIGS
— POSIX Part 1 (C Language Interface)
— POSIX Realtime Extensions
— POSIX Security Extensions
— SQL
— VRML

Appendix B — Example of Planned Tests and Test Data

What Needs to be Proven for rmdir()

For:

{ starting_dir = /
/home/goldfine
/home/goldfine/home_plus_1
}

X { path_arg = "./home/goldfine/home_plus_1/E"
"./home/goldfine/home_plus_1/N"
"./home/goldfine/home_plus_1/X"
"./home_plus_1/E"
"./home_plus_1/N"
"./home_plus_1/X"
"./E"
"./N"
"./X"
"/home/goldfine/home_plus_1/E"
"/home/goldfine/home_plus_1/N"
"/home/goldfine/home_plus_1/X"
"///home/goldfine/home_plus_1/E"
"///home/goldfine/home_plus_1/N"
"///home/goldfine/home_plus_1/X"
** "../goldfine/home_plus_1/E"
** "../goldfine/home_plus_1/N"
** "../goldfine/home_plus_1/X"
** "../home_plus_1/E"
** "../home_plus_1/N"
** "../home_plus_1/X"
"home/goldfine/home_plus_1/E/"
"home/goldfine/home_plus_1/N/"
"home/goldfine/home_plus_1/X/"
"home_plus_1/E/"
"home_plus_1/N/"
"home_plus_1/X/"
"E/"
"N/"
"X/"
"home/goldfine/home_plus_1/E//"
"home/goldfine/home_plus_1/N//"
"home/goldfine/home_plus_1/X//"

```
        "home_plus_1/E//"
        "home_plus_1/N//"
        "home_plus_1/X//"
        "E//"
        "N//"
        "X//"
        "home_plus_1/E"
        "home_plus_1/N"
        "home_plus_1/X"
        "home_plus_1/./E"
        "home_plus_1/./N"
        "home_plus_1/./X"
        "home_plus_1/../home_plus_1/E"
        "home_plus_1/../home_plus_1/N"
        "home_plus_1/../home_plus_1/X"
        "home_plus_1//E"
        "home_plus_1//N"
        "home_plus_1//X"
        "home/goldfine/home_plus_1/E"
        "home/goldfine/home_plus_1/N"
        "home/goldfine/home_plus_1/X"
        "E"
        "N"
        "X"
        "/home/goldfine/X/E"
        "/home/goldfine/X/N"
        "/home/goldfine/X/X"
        ""
        "/home/goldfine/home_plus_1/N/file
  }

 X { hp_1_search_disabled = 0
     hp_1_search_disabled = 1
  }

 X { hp_1_write_disabled = 0
     hp_1_write_disabled = 1
  },
```

invoke **rmdir** (path_arg).

Note:

E = empty directory
N = non-empty directory
X = non-existent directory
file = file
*** = skip this test when starting_dir = "/" because the combination of arguments leads to an undefined situation*

Begin each test with the directories
  /home/goldfine/home_plus_1/N
  /home/goldfine/home_plus_1/E
present (N contains the file **file**, E is empty), and with the directory
  /home/goldfine/home_plus_1/X
NOT present.

For each test,
if **rmdir()** returns 0, then
  /home/goldfine/home_plus_1/E
is removed,
  /home/goldfine/home_plus_1/N/file
remains, and
 errno == 0.
if **rmdir()** returns -1, then
 /home/goldfine/home_plus_1/E and
 /home/goldfine/home_plus_1/N/file
remain, and
 errno != 0
  If errno == EACCES, then either
    (25A) search permission is denied for some component
     (hp_1_search_disabled == 1), or
    (26A) write permission is denied for the parent of the
     directory to be removed (hp_1_write_disabled == 1).
  If errno == (EEXIST or ENOTEMPTY) (28A), then path_arg ends in

   "N" or "N/" or "N//"
  If errno == ENOENT (32A, 33A, 34A), then the test data point includes one of the following combinations:
      - starting_dir == /, and
        path_arg == { strings containing "X", or
                strings beginning with "./home_plus_1", or
                strings "./E" or "./N", or
                strings beginning with "home_plus_1", or
                strings beginning with "E" or "N", or

""
}
- starting_dir == /home/goldfine, and
path_arg == { strings containing "X", or
strings beginning with "./home/", or
strings "./E" or "./N", or
strings beginning with "../home_plus_1", or
strings beginning with "home/", or
strings beginning with "E" or "N", or
""
}
- starting_dir == /home/goldfine/home_plus_1, and
path_arg == { strings containing "X", or
strings beginning with "./home", or
strings beginning with "../goldfine", or
strings beginning with "home", or
""
}

If errno == ENOTDIR (35A), then path_arg contains the string "file".

Appendix C — Example of Code

ADL Approach

rmdir.adl (assertion file)

```
/* ADL module for function rmdir */

/* Author: Gary E. Fisher    Date written: January 24, 1997 */

module rmdir {

  extern int strcmp(const char *s1, const char *s2);
  extern char *strstr(const char *s1, const char *s2);
  extern int errno;

  char start_dir[_POSIX_PATH_MAX + 2];
  char path_arg[_POSIX_PATH_MAX + 2];
  int EACCES, ENOENT, ENOTDIR, EEXIST, EBUSY, ENOTEMPTY;
  int hp_1_search_disabled, hp_1_write_disabled;

  auxiliary {
    int check_existing(const char *xdir);
    int lsearch(const char *s1, const char *s2);
    int rsearch(const char *s1, const char *s2);
  }

// Describe semantics of rmdir

  int Zrmdir(const char *start_dir, const char *path_arg,
    int hp_1_search_disabled, int hp_1_write_disabled)

  semantics {
    normal := (return == 0),
    exception := (return == -1),

// Label rmdir errors

    search_denied := @(hp_1_search_disabled == 1),
    write_denied := @(hp_1_write_disabled == 1),
    E_exists := (check_existing("/home/fisher/home_plus_1/E")
      == 1),
    N_exists := (check_existing("/home/fisher/home_plus_1/N")
      == 1),
    F_exists := (check_existing("/home/fisher/home_plus_1/N/F")
      == 1),

    err_25A := (search_denied || write_denied),
    err_28A := ((rsearch(path_arg, "N") == 1)
      || (rsearch(path_arg, "N/") == 1)
      || (rsearch(path_arg, "N//") == 1)),
    err_32A := ((strcmp(start_dir, "/") == 0)
```

```
      && ((strstr(path_arg, "X") != 0)
        || (lsearch(path_arg, "./home_plus_1") == 1)
        || (strcmp(path_arg, "./N") == 0)
        || (strcmp(path_arg, "./E") == 0)
        || (lsearch(path_arg, "home_plus_1") == 1)
        || (lsearch(path_arg, "E") == 1)
        || (lsearch(path_arg, "N") == 1)
        || (strcmp(path_arg, "") == 0))),
    err_33A := ((strcmp(start_dir, "/home/fisher") == 0)
      && ((strstr(path_arg, "X") != 0)
        || (lsearch(path_arg, "./home/") == 1)
        || (strcmp(path_arg, "./N") == 0)
        || (strcmp(path_arg, "./E") == 0)
        || (lsearch(path_arg, "../home_plus_1") == 1)
        || (lsearch(path_arg, "home/") == 1)
        || (lsearch(path_arg, "E") == 1)
        || (lsearch(path_arg, "N") == 1)
        || (strcmp(path_arg, "") == 0))),
    err_34A := ((strcmp(start_dir, "/home/fisher/home_plus_1") == 0)
      && ((strstr(path_arg, "X") != 0)
        || (lsearch(path_arg, "./home") == 1)
        || (lsearch(path_arg, "../fisher") == 1)
        || (lsearch(path_arg, "home") == 1)
        || (strcmp(path_arg, "") == 0))),
    err_35A := strstr(path_arg, "F") != 0,


// Define assertions

    exception --> E_exists && N_exists && F_exists && (errno != 0),
    errno == EACCES --> err_25A,
    errno == EEXIST --> err_28A,
    errno == ENOTEMPTY --> err_28A,
    errno == ENOENT --> (err_32A || err_33A || err_34A),
    errno == ENOTDIR --> err_35A,

    RMDIR_NORMAL:
    normally {

      RMDIR_NORM_EXEC:
      !E_exists && N_exists && F_exists && (errno == 0)

    } //end normally
  }; //end semantics
}; //end module
```

rmdir.tdd (test data definition file)

```
/* rmdir tdd module */
/* Author: Gary E. Fisher  Date written: January 24, 1997 */

module rmdir;

int hp_1_search_disabled = [ 0, 1 ];
```

```
int hp_1_write_disabled = [ 0, 1 ];

char *start_dir = [ "/",
                    "/home/fisher",
                    "/home/fisher/home_plus_1" ];

char *path_arg = [ "./home/fisher/home_plus_1/E",
                   "./home/fisher/home_plus_1/N",
                   "./home/fisher/home_plus_1/X",
                   "./home_plus_1/E",
                   "./home_plus_1/N",
                   "./home_plus_1/X",
                   "./E",
                   "./N",
                   "./X",
                   "/home/fisher/home_plus_1/E",
                   "/home/fisher/home_plus_1/N",
                   "/home/fisher/home_plus_1/X",
                   "///home/fisher/home_plus_1/E",
                   "///home/fisher/home_plus_1/N",
                   "///home/fisher/home_plus_1/X",
                   "../fisher/home_plus_1/E",
                   "../fisher/home_plus_1/N",
                   "../fisher/home_plus_1/X",
                   "../home_plus_1/E",
                   "../home_plus_1/N",
                   "../home_plus_1/X",
                   "home/fisher/home_plus_1/E/",
                   "home/fisher/home_plus_1/N/",
                   "home/fisher/home_plus_1/X/",
                   "home_plus_1/E/",
                   "home_plus_1/N/",
                   "home_plus_1/X/",
                   "E/",
                   "N/",
                   "X/",
                   "home/fisher/home_plus_1/E//",
                   "home/fisher/home_plus_1/N//",
                   "home/fisher/home_plus_1/X//",
                   "home_plus_1/E//",
                   "home_plus_1/N//",
                   "home_plus_1/X//",
                   "E//",
                   "N//",
                   "X//",
                   "home_plus_1/E",
                   "home_plus_1/N",
                   "home_plus_1/X",
                   "home_plus_1/./E",
                   "home_plus_1/./N",
                   "home_plus_1/./X",
                   "home_plus_1/../home_plus_1/E",
                   "home_plus_1/../home_plus_1/N",
                   "home_plus_1/../home_plus_1/X",
```

```
                    "home_plus_1//E",
                    "home_plus_1//N",
                    "home_plus_1//X",
                    "home/fisher/home_plus_1/E",
                    "home/fisher/home_plus_1/N",
                    "home/fisher/home_plus_1/X",
                    "E",
                    "N",
                    "X",
                    "/home/fisher/X/E",
                    "/home/fisher/X/N",
                    "/home/fisher/X/X",
                    "",
                    "/home/fisher/home_plus_1/N/F" ];

test Zrmdir(start_dir, path_arg, hp_1_search_disabled,
  hp_1_write_disabled);
```

rmdir_aux.c (auxiliary function file)

```
/* rmdir_aux.c module */

/* Author: Gary E. Fisher  Date written: January 24, 1997 */

#include "rmdir_aux.h"

int check_existing(const char *dir);
void StartTest(void);
void Cleanup(void);
int lsearch(const char *s1, const char *s2);
int rsearch(const char *s1, const char *s2);
char cwd[_POSIX_PATH_MAX + 2];
int f;

static struct stat buf;

/* Start each test by executing StartTest */

void StartTest(void)
{ int rtn = 0, ret = 0;

/* Make sure normal permissions are set. */

  chmod("/home/fisher", 0755);
  chmod("/home/fisher/test", 0755);
  chmod("/home/fisher/test/rmdir", 0755);

/* Make sure the following exist. */

  chdir("/home/fisher");
  if (stat("/home/fisher/home_plus_1", &buf) != 0)
  {
    ret = mkdir("/home/fisher/home_plus_1", 0755);
    if (ret != 0) printf("COULD NOT CREATE home_plus_1\n");
```

```
  }
  else
  { ret = chmod("/home/fisher/home_plus_1", 0755);
    if (ret != 0) printf("COULD NOT CHMOD home_plus_1\n");
  }
  if (stat("/home/fisher/home_plus_1/N", &buf) != 0)
  { ret = mkdir("/home/fisher/home_plus_1/N", 0755);
    if (ret != 0) printf("COULD NOT CREATE home_plus_1/N\n");
  }
  else
  { ret = chmod("/home/fisher/home_plus_1/N", 0755);
    if (ret != 0) printf("COULD NOT CHMOD home_plus_1/N\n");
  }
  if (stat("/home/fisher/home_plus_1/N/F", &buf) != 0)
  { f = creat("/home/fisher/home_plus_1/N/F", 0755);
    close(f);
  }
  else
  { chmod("/home/fisher/home_plus_1/N/F", 0755);
    if (ret != 0) printf("COULD NOT CHMOD home_plus_1/N/F\n");
  }
  if (stat("/home/fisher/home_plus_1/E", &buf) != 0)
  { ret = mkdir("/home/fisher/home_plus_1/E", 0755);
    if (ret != 0) printf("COULD NOT CREATE home_plus_1/E\n");
  }
  else
  { ret = chmod("/home/fisher/home_plus_1/E", 0755);
    if (ret != 0) printf("COULD NOT CHMOD home_plus_1/E\n");
  }
}

/* Clean up the mess after each test. */

void Cleanup(void)
{ int ret = 0;

  chmod("/home/fisher", 0755);
  if (stat("/home/fisher/home_plus_1/N/F",&buf)==0)
  { ret = chmod("/home/fisher/home_plus_1/N/F", 0755);
    if (ret != 0) printf("COULD NOT CLEANUP /N/F\n");
  }
  if (stat("/home/fisher/home_plus_1/N",&buf)==0)
  { ret = chmod("/home/fisher/home_plus_1/N", 0755);
    if (ret != 0) printf("COULD NOT CLEANUP /N\n");
  }
  if (stat("/home/fisher/home_plus_1/E",&buf)==0)
  { ret = chmod("/home/fisher/home_plus_1/E", 0755);
    if (ret != 0) printf("COULD NOT CLEANUP /E\n");
  }
  if (stat("/home/fisher/home_plus_1",&buf)==0)
  { ret = chmod("/home/fisher/home_plus_1", 0755);
    if (ret != 0) printf("COULD NOT CLEANUP /home_plus_1\n");
  }
  errno = 0;
}
```

```
int check_existing(const char *dir)
{ if (stat(dir,&buf) == 0)
  { return 1;   /* directory does exist */
  }
  return 0;   /* directory does not exist */
}

int lsearch(const char *s1, const char *s2)
{ if (strlen(s1) < strlen(s2)) return 0;
  if (strncmp(s1, s2, strlen(s2))== 0) return 1;
  return 0;
}

int rsearch(const char *s1, const char *s2)
{ if (strlen(s1) < strlen(s2)) return 0;
  if (strncmp(s1+(strlen(s1)-strlen(s2)), s2, strlen(s2)) == 0)
    return 1;
  return 0;
}
```

rmdir.c (wrapper function file)

```
/* rmdir.c module */

/* Author: Gary E. Fisher  Date written: January 24, 1997 */

#include "rmdir.h"

char start_dir[_POSIX_PATH_MAX + 2];
char path_arg[_POSIX_PATH_MAX + 2];
char xdir[_POSIX_PATH_MAX + 2];
int errrtn;

int Zrmdir(const char *start_dir, const char *path_arg,
  int hp_1_search_disabled, int hp_1_write_disabled)
{ int check_existing(const char *xdir);
  int result;
  static int firsttime = 0;
  static mode_t no_perm, r, w, x, rw, rx, wx, rwx;
  static int c=0;

  if (firsttime == 0)
  { no_perm = 0;
    r = S_IRUSR | S_IRGRP | S_IROTH;
    w = S_IWUSR | S_IWGRP | S_IWOTH;
    x = S_IXUSR | S_IXGRP | S_IXOTH;
    rw = S_IRUSR | S_IRGRP | S_IROTH | S_IWUSR | S_IWGRP | S_IWOTH;
    rx = S_IRUSR | S_IRGRP | S_IROTH | S_IXUSR | S_IXGRP | S_IXOTH;
    wx = S_IWUSR | S_IWGRP | S_IWOTH | S_IXUSR | S_IXGRP | S_IXOTH;
    rwx = S_IRWXU | S_IRWXG | S_IRWXO;
    firsttime = 11;
  }
```

```
  chdir("/home/fisher");

/* Setup directories and files for each test. */

  StartTest();

  if (strcmp(start_dir, "/") != 0)
  { chmod(start_dir, rwx);
    if (hp_1_write_disabled == 1) chmod(start_dir, rx);
  }
  if (hp_1_search_disabled == 1) chmod(path_arg, wx);

  chdir(start_dir);

  errrtn = 0;
  errno = 0;

  result = rmdir(path_arg);

  errrtn = errno;

/* Clean up any files and directories left over from test. */

  Cleanup();

  errno = errrtn;

  return(result);
}
```

Traditional C Approach

rd.c (C conformance test program for rmdir)

```
/*  rd.c -- C conformance test program for rmdir                    */
/*  Alan Goldfine                                                   */

#include <string.h>
#include <stddef.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <errno.h>
#include <fcntl.h>

#define _POSIX_SOURCE
#define NULLSTRING "\n"
#define NULLCHAR '\0'
#define number_of_starting_dirs 3
#define number_of_path_args 62
```

```
int set_up_directory_structure (void);
int initialize_test_point_arguments (int, int, int, int, char *,
    char *, int *, int *);
int apply_assertions (char *, char *, int, int);

char *chmod_path;

mode_t rmdir_mode;

struct stat *stat_buf;

/************************* FUNCTION main ****************************/
int main ()
{ int i1, i2, i3, i4, hp_1_search_disabled, hp_1_write_disabled;
  int number_of_passes, number_of_fails, test_case_number;

  char *starting_dir, *path_arg;

  starting_dir = malloc (101 * sizeof(char));
  path_arg = malloc (101 * sizeof(char));
  chmod_path = malloc (101 * sizeof(char));
  stat_buf = malloc (sizeof (struct stat) );

  number_of_passes = 0;
  number_of_fails = 0;
  test_case_number = 0;

/*  initialize_test point arguments.                                  */
/*    for each test point:                                            */
/*     - establish standard directory structure                       */
/*     - call the assertion driver                                    */
  for (i1=1; i1<=number_of_starting_dirs; i1++)
  for (i2=1; i2<=number_of_path_args; i2++)
  for (i3=1; i3<=2; i3++)
  for (i4=1; i4<=2; i4++)
  { initialize_test_point_arguments (i1, i2, i3, i4, starting_dir,
              path_arg, &hp_1_search_disabled, &hp_1_write_disabled);
    if ( (strcmp (starting_dir, "/") == 0) &&
         (strncmp (path_arg, "..", 2) == 0) )
      continue;
    test_case_number++;
    printf ("\n\nTest case %d:  %s, %s, %d, %d", test_case_number,
            starting_dir, path_arg, hp_1_search_disabled,
            hp_1_write_disabled);

    set_up_directory_structure();

    if ( apply_assertions (starting_dir, path_arg,
         hp_1_search_disabled, hp_1_write_disabled) == 0 )
      { number_of_passes++;
        printf ("\nTEST PASSED");
      }
    else
      { number_of_fails++;
        printf ("\nFAILED");
```

```
      }
  }  /* end the nested for loops  */

/*  now print totals and close up shop  */
  printf ("\n\n\n\nTest Results for rmdir on jeepster:");
  printf ("\n  Pass: %d\n  Fail: %d", number_of_passes,
          number_of_fails);
  return 0;

}  /*  end main  */
/*****************************************************************/

/***************** FUNCTION set_up_directory_structure *************/
int set_up_directory_structure (void)
/*  each test case begins with the directories                     */
/*    /home/goldfine/home_plus_1/N  and                            */
/*    /home/goldfine/home_plus_1/E                                 */
/*  present (N contains the file file, E is empty), and with the   */
/*  directory                                                      */
/*    /home/goldfine/home_plus_1/X                                 */
/*  specifically NOT present                                       */

{ mkdir ("/home/goldfine/home_plus_1/E", 511);
  mkdir ("/home/goldfine/home_plus_1/N", 511);
  creat ("/home/goldfine/home_plus_1/N/file", 511);
/*  it shouldn't matter if E, N, and N/file are already there      */
  return 0;
}  /*  end set_up_directory_structure  */
/*****************************************************************/

/************** FUNCTION initialize_test_point_arguments ***********/
int initialize_test_point_arguments (int i1, int i2, int i3, int i4,
   char *starting_dir, char *path_arg,
   int *hp_1_search_disabled, int *hp_1_write_disabled)

{ switch (i1)
  {case 1: strcpy (starting_dir, "/"); break;
   case 2: strcpy (starting_dir, "/home/goldfine"); break;
   case 3: strcpy (starting_dir, "/home/goldfine/home_plus_1"); break;
   default: printf ("\nBad i1 = %d", i1); break;
  }

  switch (i2)
  {case 1: strcpy (path_arg, "./home/goldfine/home_plus_1/E"); break;
   case 2: strcpy (path_arg, "./home/goldfine/home_plus_1/N"); break;
   case 3: strcpy (path_arg, "./home/goldfine/home_plus_1/X"); break;
   case 4: strcpy (path_arg, "./home_plus_1/E"); break;
   case 5: strcpy (path_arg, "./home_plus_1/N"); break;
   case 6: strcpy (path_arg, "./home_plus_1/X"); break;
   case 7: strcpy (path_arg, "./E"); break;
   case 8: strcpy (path_arg, "./N"); break;
   case 9: strcpy (path_arg, "./X"); break;
   case 10: strcpy (path_arg, "/home/goldfine/home_plus_1/E"); break;
   case 11: strcpy (path_arg, "/home/goldfine/home_plus_1/N"); break;
   case 12: strcpy (path_arg, "/home/goldfine/home_plus_1/X"); break;
```

```
case 13: strcpy (path_arg, "///home/goldfine/home_plus_1/E");
         break;
case 14: strcpy (path_arg, "///home/goldfine/home_plus_1/N");
         break;
case 15: strcpy (path_arg, "///home/goldfine/home_plus_1/X");
         break;
case 16: strcpy (path_arg, "../goldfine/home_plus_1/E"); break;
case 17: strcpy (path_arg, "../goldfine/home_plus_1/N"); break;
case 18: strcpy (path_arg, "../goldfine/home_plus_1/X"); break;
case 19: strcpy (path_arg, "../home_plus_1/E"); break;
case 20: strcpy (path_arg, "../home_plus_1/N"); break;
case 21: strcpy (path_arg, "../home_plus_1/X"); break;
case 22: strcpy (path_arg, "home/goldfine/home_plus_1/E/"); break;
case 23: strcpy (path_arg, "home/goldfine/home_plus_1/N/"); break;
case 24: strcpy (path_arg, "home/goldfine/home_plus_1/X/"); break;
case 25: strcpy (path_arg, "home_plus_1/E/"); break;
case 26: strcpy (path_arg, "home_plus_1/N/"); break;
case 27: strcpy (path_arg, "home_plus_1/X/"); break;
case 28: strcpy (path_arg, "E/"); break;
case 29: strcpy (path_arg, "N/"); break;
case 30: strcpy (path_arg, "X/"); break;
case 31: strcpy (path_arg, "home/goldfine/home_plus_1/E//"); break;
case 32: strcpy (path_arg, "home/goldfine/home_plus_1/N//"); break;
case 33: strcpy (path_arg, "home/goldfine/home_plus_1/X//"); break;
case 34: strcpy (path_arg, "home_plus_1/E//"); break;
case 35: strcpy (path_arg, "home_plus_1/N//"); break;
case 36: strcpy (path_arg, "home_plus_1/X//"); break;
case 37: strcpy (path_arg, "E//"); break;
case 38: strcpy (path_arg, "N//"); break;
case 39: strcpy (path_arg, "X//"); break;
case 40: strcpy (path_arg, "home_plus_1/E"); break;
case 41: strcpy (path_arg, "home_plus_1/N"); break;
case 42: strcpy (path_arg, "home_plus_1/X"); break;
case 43: strcpy (path_arg, "home_plus_1/./E"); break;
case 44: strcpy (path_arg, "home_plus_1/./N"); break;
case 45: strcpy (path_arg, "home_plus_1/./X"); break;
case 46: strcpy (path_arg, "home_plus_1/../home_plus_1/E"); break;
case 47: strcpy (path_arg, "home_plus_1/../home_plus_1/N"); break;
case 48: strcpy (path_arg, "home_plus_1/../home_plus_1/X"); break;
case 49: strcpy (path_arg, "home_plus_1//E"); break;
case 50: strcpy (path_arg, "home_plus_1//N"); break;
case 51: strcpy (path_arg, "home_plus_1//X"); break;
case 52: strcpy (path_arg, "home/goldfine/home_plus_1/E"); break;
case 53: strcpy (path_arg, "home/goldfine/home_plus_1/N"); break;
case 54: strcpy (path_arg, "home/goldfine/home_plus_1/X"); break;
case 55: strcpy (path_arg, "E"); break;
case 56: strcpy (path_arg, "N"); break;
case 57: strcpy (path_arg, "X"); break;
case 58: strcpy (path_arg, "/home/goldfine/X/E"); break;
case 59: strcpy (path_arg, "/home/goldfine/X/N"); break;
case 60: strcpy (path_arg, "/home/goldfine/X/X"); break;
case 61: strcpy (path_arg, ""); break;
case 62: strcpy (path_arg, "/home/goldfine/home_plus_1/N/file");
         break;
default: printf ("\nBad i2 = %d", i2); break;
```

```
  }

  *hp_1_search_disabled = i3 -1;

  *hp_1_write_disabled = i4 - 1;

  return 0;
}  /*  end initialize_test_point_arguments  */
/*****************************************************************/

/********************** FUNCTION apply_assertions ******************/
int apply_assertions (char *starting_dir, char *path_arg,
                      int hp_1_search_disabled,
                      int hp_1_write_disabled)
 {int rd_return, fail_code, rd_errno;

  fail_code = 0;

/*  go to starting directory  */
  chdir (starting_dir);

/*  If home_plus_1 is to be search disabled:  */
  if (hp_1_search_disabled == 1)
  { strcpy (chmod_path, "/home/goldfine/home_plus_1");
    stat (chmod_path, stat_buf);
    rmdir_mode = (stat_buf -> st_mode) & ~S_IXUSR;
    chmod (chmod_path, rmdir_mode);
  }

/*  If home_plus_1 is to be write disabled:  */
  if (hp_1_write_disabled == 1)
  { strcpy (chmod_path, "/home/goldfine/home_plus_1");
    stat (chmod_path, stat_buf);
    rmdir_mode = (stat_buf -> st_mode) & ~S_IWUSR;
    chmod (chmod_path, rmdir_mode);
  }

/*-----------------------------------------------------------------*/
/*  Attempt to remove directory                                    */
  errno = 0;
  rd_return = rmdir (path_arg);
  rd_errno = errno;
  printf ("\n errno = %d", rd_errno);

/*-----------------------------------------------------------------*/
/*  need to clean up access mode before doing assertions           */
/*                                                                 */
/*  make sure that access mode of home_plus_1 is normal            */
/*    (full access for all)                                        */

  strcpy (chmod_path, "/home/goldfine/home_plus_1");
  stat (chmod_path, stat_buf);
  rmdir_mode = (stat_buf -> st_mode)
                | ( S_IRWXU | S_IRWXG | S_IRWXO );
  chmod (chmod_path, rmdir_mode);
```

```
/*--------------------------------------------------------------------*/

/*--------------------------------------------------------------------*/
/*  now do assertions                                                 */

/*  assertions involving claim of successful completion of rmdir     */
  if (rd_return == 0)
    { if (chdir ("/home/goldfine/home_plus_1/E") == 0)
        { printf ("    Directory /home/goldfine/home_plus_1/E");
          printf ("is still there!");
          fail_code = 1;
        }
      if (chdir ("/home/goldfine/home_plus_1/N") != 0)
        { printf ("    Directory /home/goldfine/home_plus_1/N is
gone!");
          fail_code = 1;
        }
      if (rd_errno != 0)
        { printf ("    errno from rmdir is not 0!");
          fail_code = 1;
        }
    }

/*  assertions involving claim of unsuccessful completion of rmdir  */
  if (rd_return != 0)
    { if (chdir ("/home/goldfine/home_plus_1/E") != 0)
        { printf ("    Directory /home/goldfine/home_plus_1/E is
gone!");
          fail_code = 1;
        }
      if (chdir ("/home/goldfine/home_plus_1/N") != 0)
        { printf ("    Directory /home/goldfine/home_plus_1/N is
gone!");
          fail_code = 1;
        }
      if (rd_errno == 0)
        { printf ("    errno from rmdir is 0!");
          fail_code = 1;
        }
      if (rd_errno == EACCES)
        { if ( (hp_1_search_disabled == 0)
               && (hp_1_write_disabled == 0) )
            { printf ("    errno == EACCESS but there was no access
disabling!");
              fail_code = 1;
            }
        }
      if ( (rd_errno == EEXIST) || (rd_errno == ENOTEMPTY) )
        { if ( (strcmp (path_arg+(strlen(path_arg)-1), "N") != 0) &&
               (strncmp (path_arg+(strlen(path_arg)-2), "N/", 2)
                                                        != 0)  &&
               (strncmp (path_arg+(strlen(path_arg)-3), "N//", 3)
                                                              != 0)
             )
            { printf ("    errno == %d (EEXIST or ENOTEMPTY) but ",
```

```
                    rd_errno);
            printf ("path_arg did not point to N!");
            fail_code = 1;
          }
      }
    if (rd_errno == ENOENT)
      { if ( ! ( ( (strcmp (starting_dir, "/") == 0) &&
                   ( (strchr (path_arg, 'X') != NULL) ||
                     (strncmp (path_arg, "./home_plus_1", 13)
                                                          == 0) ||
                     (strcmp (path_arg, "./E") == 0) ||
                     (strcmp (path_arg, "./N") == 0) ||
                     (strncmp (path_arg, "home_plus_1", 11) == 0) ||
                     (strncmp (path_arg, "E", 1) == 0) ||
                     (strncmp (path_arg, "N", 1) == 0) ||
                     (strcmp (path_arg, "") == 0)
                   )
                 ) ||
                 ( (strcmp (starting_dir, "/home/goldfine") == 0) &&
                   ( (strchr (path_arg, 'X') != NULL) ||
                     (strncmp (path_arg, "./home/", 7) == 0) ||
                     (strcmp (path_arg, "./E") == 0) ||
                     (strcmp (path_arg, "./N") == 0) ||
                     (strncmp (path_arg,
                                    "../home_plus_1", 14) == 0) ||
                     (strncmp (path_arg, "home/", 5) == 0) ||
                     (strncmp (path_arg, "E", 1) == 0) ||
                     (strncmp (path_arg, "N", 1) == 0) ||
                     (strcmp (path_arg, "") == 0)
                   )
                 ) ||
                 ( (strcmp (starting_dir,
                            "/home/goldfine/home_plus_1") == 0) &&
                   ( (strchr (path_arg, 'X') != NULL) ||
                     (strncmp (path_arg, "./home", 6) == 0) ||
                     (strncmp (path_arg, "../goldfine", 11) == 0) ||
                     (strncmp (path_arg, "home", 4) == 0) ||
                     (strcmp (path_arg, "") == 0)
                   )
                 )
             )
           ) /* end if  */
        { printf ("    errno == ENOENT but path_arg was OK!");
          fail_code = 1;
        }
      } /*  end if(rd_errno == ENOENT  */
    if (rd_errno == ENOTDIR)
      { if ( strstr (path_arg, "file") == NULL)
        { printf ("    errno == ENOTDIR but path_arg did not contain
file!");
          fail_code = 1;
        }
      } /*  end if (rd_errno == ENOTDIR  */
  } /*  end if (rd_return != 0)  */
```

```
  return (fail_code);
}  /*  end apply_assertions  */
```

Appendix D — Final Filled-In Form

Stage 1: Select and Learn the Application Specification

1. Identification of the application specification

"POSIX--Part 1" (the C-language interface specification), ISO/IEC 9945-1: 1990 (E), IEEE Std 1003.1-1990, supplemented by "Test Methods for Measuring Conformance to POSIX," IEEE Std 2003.1-1992.

2. Number of pages in the specification

356 pages in the interface specification document and 442 pages in the Test Methods document.

3. Number of functions contained in the specification

The POSIX C-language interface contains 99 defined functions.

4. Functions in the specification that were selected for Phase 1

Four: chdir, umask, rmdir, and chmod.

5. Time required to learn the application specification (including the joint development of strategy for the individual functions)

(Alan Goldfine), by nonzero week, 9/96 - 2/97:

9/30: 16 hrs.
10/7: 30 hrs.
10/14: 3 hrs.
10/21: 12 hrs.
10/28: 14 hrs.
11/4: 8 hrs.
11/11: 3 hrs.
11/18: 8 hrs.
12/2: 13 hrs.
12/30: 8 hrs.
1/6: 6 hrs.
1/13: 6 hrs.
2/3: 1 hr.
2/10: 2 hrs.

(Gary Fisher), by nonzero week, 9/96 - 2/97:

9/30: 32 hrs.
10/7: 15 hrs.
10/14: 1 hr.
10/28: 2 hrs.
11/18: 1 hr.

1/6: 1 hr.
1/20: 2 hrs.
2/10: 1 hr.

Stage 2: Develop the Test Suite

(This includes the specification of the assertions and test data. The use of a first candidate implementation as a reference to help test the test suite that was produced was integral to this stage, and was included in this part of the analysis)

6. List of tools and methods used to specify the assertions and the test data

ADLT (Assertion Definition Language Translator) from Sun Microsystems, supported by C language auxiliary and wrapper functions.

7. Description of the hardware and software used for the test suite development

Sun SPARCstation running Solaris 2.4 (POSIX compliant).

8. Time required to acquire/install the assertion specification software

SunOS 4.1.3 SPARCstation: 11 hrs.

Solaris 2.4 SPARCstation: 12 hrs.

9. Personnel time required to learn the assertion and test data language(s)

(Alan Goldfine), by nonzero week, 4/96 - 12/96 (includes time spent at ADL training course and on getcwd dry run):

4/29: 4 hrs.
5/6: 12 hrs.
5/13: 12 hrs.
5/20: 8 hrs.
5/27: 14 hrs.
6/3: 13 hrs.
6/10: 19 hrs.
6/17: 19 hrs.
6/24: 15 hrs.
7/1: 15 hrs.
7/8: 19 hrs.
7/15: 18 hrs.
7/22: 19 hrs.
7/29: 10 hrs.
8/5: 14 hrs.
8/12: 13 hrs.
8/19: 9 hrs.
8/26: 32 hrs.
9/2: 13 hrs.

9/9: 20 hrs.
9/16: 20 hrs.
9/23: 16 hrs.
9/30: 8 hrs.
10/7: 4 hrs.
10/14: 3 hrs.
10/21: 12 hrs.
10/28: 14 hrs.
11/4: 8 hrs.

(Gary Fisher), by nonzero week, 8/96 - 1/97 (includes time spent at ADL training course and on mkdir dry run):

8/26: 24 hrs.
11/25: 2 hrs.
12/2: 11 hrs.
12/9: 10 hrs.
12/16: 25 hrs.
1/13: 8 hrs.
1/20: 2 hrs.
1/27: 8 hrs.
2/3: 10 hrs.
2/10: 2 hrs.
2/24: 2 hrs.

10. Personnel time required to write, test, and revise, in ADL, the assertions and the test data

(Alan Goldfine), by application function and nonzero week, 10/96 - 12/96:

chdir:
10/14: 1 1/2 hrs.
10/21: 3 hrs.
10/28: 2 hrs.
11/4: 4 hrs.

umask:
11/11: 3 hrs.
11/18: 2 hrs.

(Gary Fisher), by application function and nonzero week, 12/96 - 2/97:

rmdir:
1/6: 1 hr.
1/20: 4 hrs.

chmod:
2/10: 13 hrs.
2/17: 6 hrs.

11. Personnel time required to write, test, and revise any necessary C routines that supported the ADL assertion and test data modules

(Alan Goldfine), by application function and nonzero week, 10/96 - 12/96:

chdir:
10/14: 2 hrs.
10/21: 6 hrs.
10/28: 8 hrs.
11/4: 9 hrs.

umask:
11/11: 7 hrs.
11/18: 2 hrs.

(Gary Fisher), by application function and nonzero week, 12/96 - 2/97:

rmdir:
1/27: 12 hrs.
2/3: 10 hrs.

chmod:
2/10: 8 hrs.
2/17: 17 hrs.
2/24: 6 hrs.

12. Personnel time required to write, test, and revise the test programs and test data in C

(Gary Fisher), by application function and nonzero week, 10/96 - 12/96:

chdir:
10/14: 15 hrs.
10/21: 16 hrs.

umask:
10/21: 2 hrs.
11/11: 4 hrs.

(Alan Goldfine), by application function and nonzero week, 12/96 - 2/97:

rmdir:
1/6: 13 hrs.

chmod:
2/10: 23 hrs.
2/17: 8 hrs.

Stage 3: Run the Generated Test Suite Against a Second Candidate Implementation

13. Description of the hardware and software of the second candidate implementation

Sun SPARCstation running SunOS release 4.1.3 (not POSIX compliant).

14. Personnel time required to perform the testing of the second candidate implementation using ADL

(Alan Goldfine), by application function and nonzero week, 10/96 - 12/96:

chdir:
11/4: 3 hrs.

umask:
11/18: 1 hr.

(Gary Fisher), by application function and nonzero week, 12/96 - 2/97:

rmdir:
2/3: 1 hr.

chmod:
2/24: 1 hr.

15. Personnel time required to perform the testing of the second candidate implementation using C

(Gary Fisher), by application function and nonzero week, 10/96 - 12/96:

chdir: 10/21:
2 hrs.

umask:
10/21: 5 hrs.

(Alan Goldfine), by application function and nonzero week, 1/97 - 2/97:

rmdir:
1/17: 1 hr.

chmod:
2/24: 1 hr.

Stage 4: Assess the Final Test Suites

16. Number of assertions tested/Total number of required/base assertions, returns, and error conditions in the application specification document

chdir: 18/23

umask: 4/4

rmdir: 19/25

chmod: 17/23.

17. Final number of correct generated tests

chdir: 928

umask: 262,144

rmdir: 720

chmod: 60,928.

18. Degree of portability of the generated test suite

We had no problems running, on the second candidate implementation, the test suite that was developed on the first candidate implementation.

19. Overall ease in using the automated test generation software to generate test code

The generated test executables crashed frequently and messily, although the crashes invariably turned out to be due to user error or the user's misunderstanding of the subtleties of the ADL specs. However, the source code for these generated programs is either unavailable to, or unreadable by, the user. While this is part of the design of ADLT, and perhaps inherent in the nature of generated code, it did continually lead to test programs that were notoriously difficult to debug.

20. Number and a listing of the problems in the application specification (if any) that were uncovered by the writing of the assertions

None were identified, other than an occasional lack of clarity in the presentation of the application specification.