Testing and Validation of Computerized Decision Support Systems

R. Matthew Sailors, ME[†], Thomas D. East, Ph.D. [†],
C. Jane Wallace, RN, MS, Debra A. Carlson, BS CIS,
Margaret A. Franklin, RN, BSN, Laura K. Heermann, RN, BSN,
A. Tupper Kinder, BS, Richard L. Bradshaw, BS[†],
Adrienne G. Randolph, M.D.[†], Alan H. Morris, M.D.
[†]Department of Medical Informatics, University of Utah School of Medicine and
Pulmonary Division, LDS Hospital, Salt Lake City, Utah

Systematic, thorough testing of decision support systems (DSSs) prior to release to general users is a critical aspect of high quality software design. Omission of this step may lead to the dangerous, and potentially fatal, condition of relying on a system with outputs of uncertain quality. Thorough testing requires a great deal of effort and is a difficult job because tools necessary to facilitate testing are not well developed. Testing is a job ill-suited to humans because it requires tireless attention to a large number of details. For these reasons, the majority of DSSs available are probably not well tested prior to release. We have successfully implemented a software design and testing plan which has helped us meet our goal of continuously improving the quality of our DSS software prior to release. While requiring large amounts of effort, we feel that the process of documenting and standardizing our testing methods are important steps toward meeting recognized national and international quality standards.

Our testing methodology includes both functional and structural testing and requires input from all levels of development. Our system does not focus solely on meeting design requirements but also addresses the robustness of the system and the completeness of testing.

INTRODUCTION

In our experience, few decision support systems (DSSs) are thoroughly tested before they are released to general users, a conclusion which others have also observed (1). Thorough testing requires a great deal of effort and is a difficult job because tools necessary to facilitate testing are not well developed. Testing is a job ill-suited to humans because it requires tireless attention to a large number of details. For these reasons, the majority of DSSs available are probably not well tested prior to release. Lack of thorough testing may lead to low quality output from or malfunction of the DSS during clinical use.

Systematic test plans and tools that allow thorough structural and functional test scenarios are prerequisites for developing quality software. Designers and testing personnel often feel that "real world" and "everyday" scenarios are sufficient to completely test a complex DSS. In most testing scenarios which use "real world" historical data and events, the majority of the scenarios test only a small percentage of the knowledge base and may not test some areas of the knowledge base at all (2). While this may represent how the DSS will be used in daily practice, DSS developers should be wary of taking the chance that a wrong decision will come from the system because of inadequate testing prior to release.

For example, a system with only 10 Boolean inputs has 2^{10}, or 1024, different possible combination of inputs while a system with 10 graduated inputs, each with 10 different levels has 10^{10} different combinations of inputs. Given this complexity, how many systems are tested *systematically* with *all* of the potential combination of inputs to verify the quality of the system?

BACKGROUND

Decision Support System

The decision support system for which these testing methods were originally developed is designed to manage mechanical ventilation of critically ill patients . The DSS was developed around a finite state automaton inference engine. The knowledge base consists of several interdependent modules which handle the various aspects of mechanical ventilator therapy (e.g. ventilation, oxygenation assessment, and weaning) and a map file which controls the order of execution of the states (3).

Testing

The reasons for testing systems and software can be divided into three categories: to judge acceptability, to judge quality, and to discover problems. Acceptability testing verifies that the system meets design requirements. Quality testing is intended to build confidence in the system. Problem discovery looks for discrepancies between design specifications and observed behavior. Traditionally, acceptability testing and problem discovery are the most commonly used methods (4, 5).

0195—4210/96/$5.00 © 1996 AMIA, Inc.

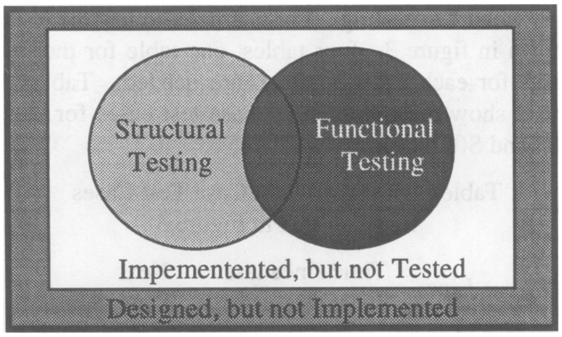

Figure 1 Venn Diagram for Realms of Design, Implementation, and Testing

By testing, we can help avoid falling victim to the Titanic Principle: 'The magnitude of a system's failure is directly proportional to the designer's (or builder's) belief that it cannot fail' (5).

There are two main paradigms for testing systems: structural (clear box) testing and functional (black box) testing. Structural testing requires information about the structure of the system, and subjects the individual elements of the system to independent examination. Functional testing is concerned only with the inputs and outputs of the system. While functional testing alone can effectively test a complex system, it makes it difficult to track down and fix errors. Structural testing alone cannot test a complex system because of the interactions between individual elements. Thus, an efficient testing plan will need to include both functional and structural testing.

Figure 1 shows the world in which the system designers, implementation personnel, and testers must work. The goal of testing is to minimize the operations which were designed and implemented but not tested.

Before testing can begin, a test plan must be drawn up. Ideally, this plan is based on a verification and validation document prepared during the design phase of system development. The nature of the test plan depends on the choice of performance standard. Examples of performance standards for decision support systems (DSS) include DSS - expert agreement, DSS - design agreement, quality improvement, variation reduction, and outcome improvement. Many DSSs designed for the health care field are primarily concerned with agreement between the DSS results and a "gold standard," which is often the clinician whose knowledge was supposed to be captured in the DSS knowledge base.

Traditional Testing Paradigm

The traditional method of testing a DSS involves generating scenarios which the designers feel represent the situations the DSS will encounter in real world use. The majority of these cases test only the most common pathways. In many instances, entire branches of the DSS knowledge base will not be tested by "real world" scenarios (2). This leads to the dangerous situation of relying on a system which has not been thoroughly tested, but which has been "certified" to perform as designed. Thus, when the one case in a thousand is presented to the DSS, the DSS's output is unreliable.

New" Testing Paradigms

Recently more emphasis has been placed on judging decision support aids on not only accuracy but also on the "appropriateness of its conclusions and the scope of its consideration" (6). Wasson, et.al. proposed standards for validation of systems that were followed in fewer than 50% of validation studies (1).

Investigators are now being asked to distinguish between efficacy (the promise of performance) and the effectiveness (the delivered performance) of the decision support aid and to account for the effects of potential biases in both the system itself and the methods employed to validate the system (6-8) .

METHODS

Planning

Before testing began, we created a verification and validation document, which described, in detail, methods to be used in testing and standards for developing testing plans. Our testing plan is divided into several parts designed to test both the knowledge base and its implementation. The knowledge engineer is responsible for developing the knowledge base and verifying its function on paper. The programmer is responsible for verifying that the implemented knowledge base meets design criteria. Finally, members of our quality improvement team, other than the knowledge engineers and programmers who developed the DSS, test the decision support system to verify that it performs appropriately in a variety of clinical scenarios.

Testing plans for each module of the knowledge base are created and maintained as the knowledge base grows and evolves. The testing plans include mandatory structural testing of the knowledge base module and functional testing of the integration of the module into the rest of the knowledge base. The flow diagram in Figure 2 shows the steps in our verification and validation planning procedure.

Testing Methods

In testing the ventilator management DSS, we used a combination of structural and functional testing. Each of the states in the protocol underwent individual structural testing as did each module. Func-

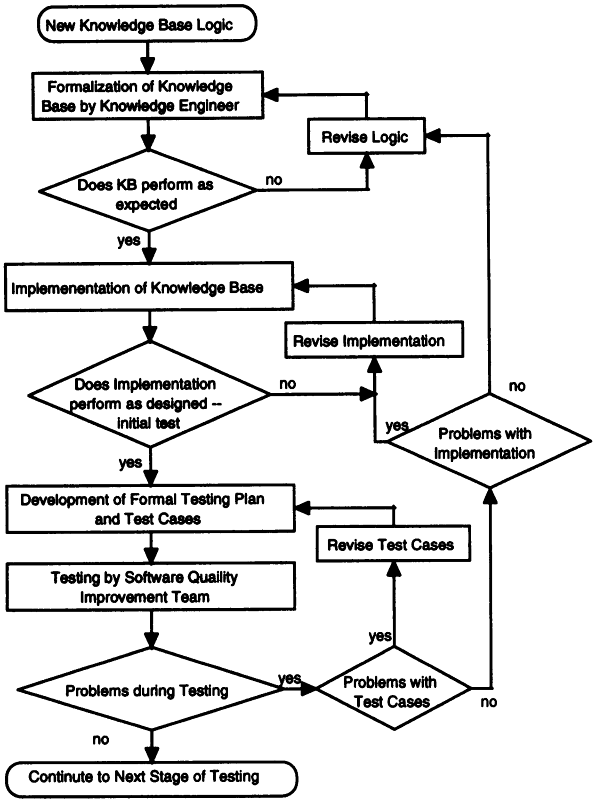
New Knowledge Base Logic
Formalization of Knowledge Base by Knowledge Engineer
Revise Logic
Does KB perform as expected
no
yes
Implemenentation of Knowledge Base
Revise Implementation
Does Implementation perform as designed -- initial test
no
no
yes
Problems with Implementation
yes
Development of Formal Testing Plan and Test Cases
Revise Test Cases
Testing by Software Quality Improvement Team
yes
Problems during Testing
yes
Problems with Test Cases
no
no
Continute to Next Stage of Testing

Figure 2 Verification and Validation Planning Procedure

tional testing extended to the integration of the individual modules and the whole protocol system.

Structural Testing: The finite state automaton paradigm lends itself to structural testing which involves testing the function of the DSS by testing the method of implementation. Structural testing requires specific and detailed knowledge of the structure of the system. To test the DSS modules, we verify the operation of each state in the automaton at both the code and functional level using a combination of robust worst cast testing . For example, to test the logic shown in figure 3, four tables, one table for the test cases for each decision state, are needed. Tables 1 and 2 show the robust worst case test cases for state S08 and S09, respectively.

Table 1. Robust Worst Case Test Cases for State S08 in Figure 3

Line	Time since last PaO_2 input	Next State
1	6:31	S09
2	6:30	A07
3	6:29	A07
4	3:00	A07

Table 2. Robust Worst Case Test Cases for State S09 in Figure 3

Line	PEEP input	FiO_2 input	Next State
1	10	0.3	A07
2	14	0.59	A07
3	14	0.60	A07
4	14	0..61	A07
5	15	0.59	A07
6	15	0.60	A07
7	15	0..61	A06
8	16	0.59	A07
9	16	0.60	A07
10	16	0..61	A06

These tables begin to illustrate the problem of increasingly complex decision states. To fully specify a robust worst case testing plan for a decision with n parallel conditionals and m_1, m_2, ... m_n boundary limits (in the State S09 in Figure 3 $n = 2$ and $m_1 = m_2 = 1$), there would be $3^n \Pi(m_i) + 1$ rows and $n + 1$ columns in the logic table (thus for a single variable two-limit test, you need 7 lines, and for a 2-variable, two-limit test you need 37 lines).

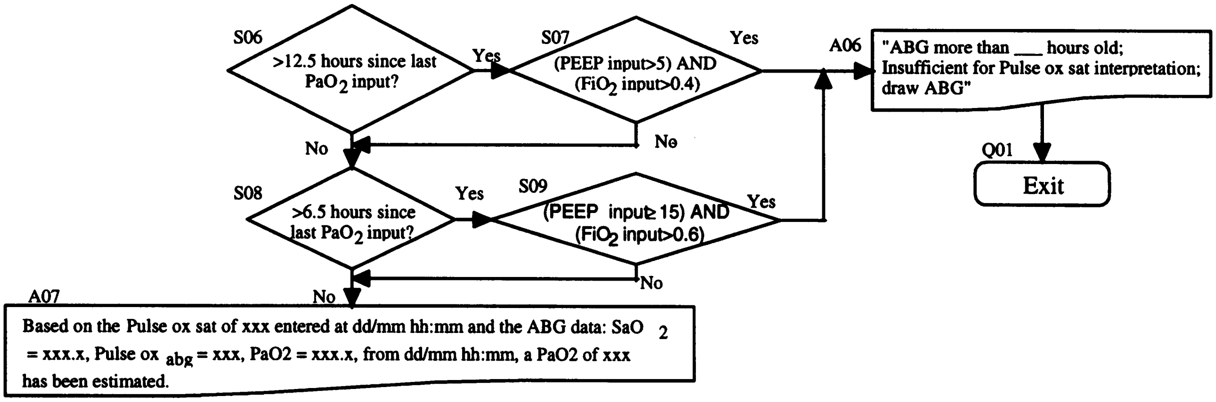
S06
>12.5 hours since last PaO_2 input?
Yes
S07
(PEEP input>5) AND (FiO_2 input>0.4)
Yes
A06
"ABG more than ___ hours old; Insufficient for Pulse ox sat interpretation; draw ABG"
No
No
Q01
Exit
S08
>6.5 hours since last PaO_2 input?
Yes
S09
(PEEP input≥15) AND (FiO_2 input>0.6)
Yes
No
A07
No
Based on the Pulse ox sat of xxx entered at dd/mm hh:mm and the ABG data: SaO_2 = xxx.x, Pulse ox_{abg} = xxx, PaO2 = xxx.x, from dd/mm hh:mm, a PaO2 of xxx has been estimated.

Figure 3. Section of Ventilator Management Protocol. Copyright 1996, Intermountain Health Care, Inc.

Functional Testing: Functional testing is concerned with testing the functionality of a system without regard to the method of implementation. Therefore, test cases developed solely from the DSS design specifications can be used to test a variety of different implementations of the knowledge base. The choice of test cases for functional testing of the DSS modules and their integration is more problematic, as all system inputs and outputs must be identified and specified or predicted. Test cases for functional testing are often inspired by real-world observations, but must also be derived from design specifications.

Test Case Design and Selection: Our test cases were chosen to perform both structural and functional testing. As previously noted, neither structural nor functional testing alone is a practical method of verifying the performance of a DSS. Figure 4 shows the steps used to develop specific test cases and testing plans

Error Handling and Recovery: Any errors in either the knowledge base or its specific implementation are reported to the testing supervisor, programmer, and, if appropriate, the knowledge engineer. An incident report is created which notes the observed effect, the test conditions which lead to the error and the probable location of the error. Errors in implementation are corrected by the programmer who is also responsible for competing the resolution section of the incident report noting the exact problem, its resolution, the date and time of the correction and attests to the correct operation of the module during a bench test.

Knowledge base errors are corrected by the knowledge engineer and approved by either the whole knowledge base consensus group or, for minor or emergency changes, the chair of the group. Once the knowledge base has been corrected the knowledge engineer completes the incident report and generates a work order for the programmer to adjust the implementation. Both the programmer and knowledge engineer must attest to the correct operation of the adjusted module.

Testing on all modules which may be affected by the erroneous module is stopped until corrections can be made. While this slows the overall testing process, it does reduce the probability of diagnosing "false errors" because of erroneous data propagating through the system.

RESULTS

Our testing of the computerized ventilator management protocols (approximately 350 total branch and calculation states) generated more than 2000 test cases and more than 8000 rows in the corresponding logic tables. Recent development and testing of charting screens for three ventilators required in excess of 3000 logic table rows and several hundred test cases.

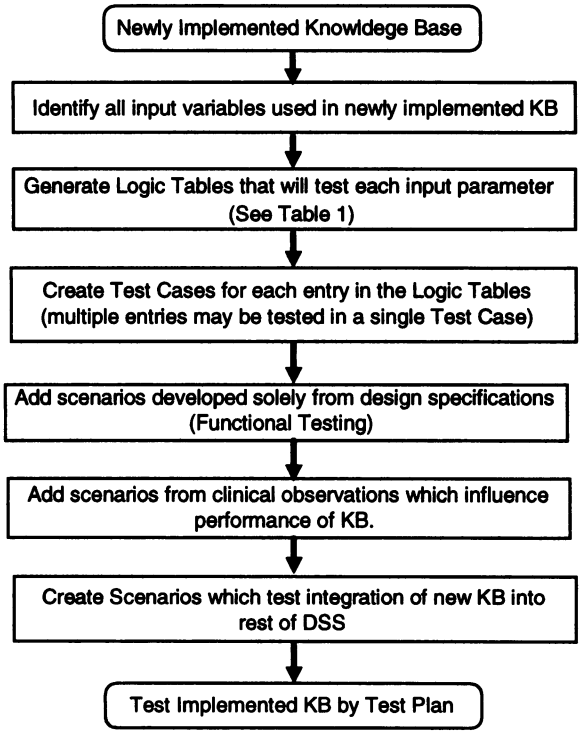
Newly Implemented Knowldege Base
Identify all input variables used in newly implemented KB
Generate Logic Tables that will test each input parameter (See Table 1)
Create Test Cases for each entry in the Logic Tables (multiple entries may be tested in a single Test Case)
Add scenarios developed solely from design specifications (Functional Testing)
Add scenarios from clinical observations which influence performance of KB.
Create Scenarios which test integration of new KB into rest of DSS
Test Implemented KB by Test Plan

Figure 4. Testing Plan Development Process

Testing of the full ventilator management DSS required more than 900 man-hours of work, but has led to a system which performs as designed and is easily maintained. Since the implementation of these systematic methods for testing our DSS were implemented, the rate of implementation and logic errors has decreased significantly.

DISCUSSION

Our testing methodology compares favorably with the principles outlined by other researchers for evaluating the efficiency of a DSS (1, 7-9). We do not address the other major concern of evaluating expert systems, namely their impact on the medical care process and medical outcomes.

There are two major drawbacks to using this methodology:

1. it relies on humans to build logic tables, develop test cases, enter test cases into computer
2. it tests only the efficacy of DSS

While the reliance on humans is an inconvenience and an area for future improvement, it is the lack of effectiveness testing which can be construed as a weakness of this system. The inherent limitations of humans are well known, but by having multiple layers of personnel entering the test cases and reviewing results, the probability of missing errors are virtually eliminated.

We have developed and continue to develop some automated tools which address the aforementioned drawbacks. An automated scenario testing system has been developed to reduce the human labor required for feeding scenarios to the DSS, although human auditing of the testing results is still required. We envision that new tools and knowledge base paradigm standards, such as Arden Syntax, will allow the intelligent authoring tools which will partially automate the process of building logic tables and test cases. These can then be passed to an automated tester for testing against the knowledge base implementation.

While this method of testing and validating computerized decision support systems has been very successful for us, it is only the first step on the road to software maturity. This process is only the beginning of the evolution into the realm of Software Process Maturity. It only addresses the first few steps of the Capability Maturity Model (CMM) (10).

Many years ago, other industries recognized the need for systematized software development and testing plans. Unfortunately the medical computing industry has only just begun to realize that some sort of action needs to be taken on this front. Medical computing is now under increasing pressure to meet ISO 9000 series requirements internationally as well as Unites States Food and Drug Administration (FDA) and U.S. Congress software and medical device guidelines (11-13).

SUMMARY

We have successfully designed and implemented a software design and testing plan which has helped up meet our continuous quality improvement goals. While it does require large amounts of effort, we feel that the process of documenting and standardizing the testing methods are important steps toward meeting recognized national and international quality standards. For computerized decision support systems to fully realize their potential, they must be not only accurate but robust. Systematic and thorough testing is the most effective tool to attain this goal.

If medical informatics and medical computing are interested in improving the quality of DSS software, their respective communities need to support and implement systematic software testing practices.

References

1. Wasson J, Sox H, Neff R, Goldman L. Clinical Prediction Rules. Applications and Methodological Standards. N Engl J Med 1985;313:793-799.
2. Sailors RM, East TD. A model-based simulator for testing rule-based decision support systems for mechanical ventilation of ARDS patients. Proc Annu Symp Comput Appl Med Care 1994;1007:1007.
3. Kinder AT, East TD, Littman WD, et al. A Computerized Decision Support System for Management of Mechanical Ventilation in Patients with ARDS: An Example of Exportation of a Knowledge Base. Proc Annu Symp Comput Appl Med Care 1994:888.
4. Jorgensen PC. Software Testing: A Craftman's Approach. Boca Raton, Florida: CRC Press, 1995.
5. DeMillo RA. Software Testing and Evaluation. The Benjamin/Cummings Publishing Compayn, Inc., 1987.
6. Diamond GA, Pollock BH, Work JW. Clinician Decsions and Computers. In: Shabot MM, Gardner RM, eds. Decision Support Systems in Critical Care. New York: Springer-Verlag New York, Inc., 1994:(Orthner HF, ed. Computers in Medicine;
7. Rossi Mori A, Pisanelli DM, Ricci FL. Evaluation stages and design steps for knowledge-based systems in medicine. Med Inf Lond 1990;15(3):191-204.
8. Miller PL, Sittig DF. The evaluation of clinical decision support systems: what is necessary versus what is interesting. Med Inf Lond 1990;15(3):185-90.
9. Ohayon MM. Validation of expert systems: examples and considerations. Medinfo 1995;8 Pt 2:1071-5.
10. Saiedian H, Kuzara R. SEI Capability Maturity Model's Impact on Contractors. Computer 1995;28(1):16-26.
11. Office of Device Evaluation, Center for Devices and Radiological Health , Food and Drug Adminstration. Review Guidance for Computer Controlled Medical Devices Undergoing 501(k) Review. Washington, D.C.: Department of Health and Human Services, 1991.
12. U.S. Food and Drug Adminstration. Good Clinical Practices. Buffalo Grove, IL: Interpharm Consulting, 1992.
13. Committee on Energy and Commerce U.S. House of Representatives. Report on the Safe Medical Devices Act of 1990. Washington D.C.: Congressional Printing Office, 1990.