Applying the SCR Requirements Method to a Weapons Control Panel: An Experience Report

Constance Heitmeyer, James Kirby, and Bruce Labaw*
Naval Research Laboratory
Code 5546, Washington, DC 20375 USA
{heitmeyer, kirby, labaw}@itd.nrl.navy.mil

ABSTRACT
A major barrier to the use of formal methods in software practice is the difficulty software developers have understanding and applying the methods. To overcome this barrier, a requirements method called SCR (Software Cost Reduction) offers a user-friendly tabular notation to specify software requirements and a collection of easy-to-use tools that automatically detect many classes of errors in requirements specifications. This paper describes our experience in applying the SCR method and tools to a safety-critical military application—the problems encountered in translating the original contractor-produced software requirements specification into SCR and the lessons learned in applying the SCR technology to a practical system. The short time required to apply the SCR method, the serious safety violation detected, and the working system prototype produced demonstrate the utility and potential cost-effectiveness of SCR for developing safety-critical systems.

Keywords
requirements, specification, formal methods, formal specification, consistency checking, verification, validation, software tools, simulation, model checking

INTRODUCTION
During the last decade, numerous formal methods for developing computer systems have been proposed. One area in which formal methods are having a major impact is hardware design. Not only are companies such as Intel beginning to use model checking (along with simulation) as a standard technique for detecting errors in hardware designs, in addition, some companies are developing their own in-house model checkers. Further, a number of model checkers, customized for hardware design, are becoming available commercially [23].

* This work was supported by ONR and SPAWAR.
http://www.itd.nrl.navy.mil/ITD/5540/personnel/heitmeyer.html

FMSP 98 Clearwater Beach, FL USA
0-89791-954-8/98/0003

Although some limited progress has been made in applying formal methods to software (see, e.g., [14, 15, 13, 8]), the use of the methods in practical software development is still rare. A significant barrier to the use of formal methods among software developers is the widespread perception that the formal notations and formal analysis techniques provided by the methods are difficult to understand and apply. Moreover, software developers express serious doubts about the scalability and cost-effectiveness of formal methods.

A promising approach to overcoming these problems is to hide the logic-based languages associated with most formal methods and to adopt a notation, such as graphics or tables, that developers find easier to use. Specifications in the more user-friendly notation can be translated automatically to a form more amenable to formal analysis. To scale effectively, a formal method must be supported by powerful, easy-to-use tools. To the extent feasible, the tools should detect errors automatically and provide easy-to-understand feedback useful in tracing the cause of an error.

By providing a user-friendly tabular notation with demonstrated scalability, a method called SCR (Software Cost Reduction) for specifying software requirements has already achieved some success in practice. SCR was originally formulated to document the requirements of the Operational Flight Program (OFP) of the U.S. Navy's A-7 aircraft [20, 1]. Subsequently, a number of industrial organizations, such as Grumman, Bell Laboratories, and Ontario Hydro, have used SCR to specify the requirements of practical systems, including avionics systems (see, e.g., [24]), a telephone switching network [21], and the shutdown software for a nuclear power plant [27]. More recently, the requirements of the OFP of Lockheed's C-130J aircraft were specified in SCR [12]. The OFP consists of more than 230K lines of Ada, thus demonstrating SCR's scalability.

To support the SCR method, we have developed a complete formal semantics for the SCR notation [18] and a collection of software tools for specifying and analyzing software requirements [19, 16]. The SCR tools include a *specification editor* for creating and modifying

a requirements specification in the SCR tabular notation, an automated *consistency checker* for checking the specification for well-formedness errors (e.g., syntax and type errors, missing cases, circular definitions, and unwanted nondeterminism), a *simulator* for symbolically executing the specification to ensure that it captures the customer's intent, and a *model checker* for analyzing the specification for critical application properties.

Our overall goal in designing the SCR tools is to make them useful and cost-effective in practical software development. To achieve this goal, we have established the following objectives:

- The tools should be tightly integrated, and each should be based on the same formal semantics [18].
- The tools should automatically detect many classes of errors and, upon detecting an error, should provide detailed, user-oriented feedback about the location and cause of the error.
- To validate the specifications, a simulator must be supported. To facilitate testing of the specification by domain experts, the construction of customized front-ends for the simulator must also be supported.
- Because analysis of a complete requirements specifications is often infeasible (e.g., the specification may be incomplete), extracting and analyzing individual parts of the specification must be supported.

Recently, the SCR technology has been evaluated in three pilot projects. In the first project, researchers at NASA's IV&V Facility used the SCR consistency checker to detect a number of missing cases and several instances of nondeterminism in the prose requirements specification of the software for the NASA space station [10, 11]. In the second project, engineers at Rockwell-Collins used the specification editor, the consistency checker, and the simulator to detect 24 errors, many of them serious, in the requirements specification of an example flight guidance system [25].

This paper describes a third pilot project in which the SCR tools, including a newly integrated model checker [5, 4], were applied to a safety-critical military system. After introducing the system of interest, the Weapons Control Panel (WCP), this paper describes how we translated a draft software requirements specification (SRS) prepared by a military contractor into the SCR notation, the problems encountered in the translation, and the major lessons learned in applying the SCR method to the WCP.

OVERVIEW OF THE WCP AND THE SRS

Operators of a U.S. military system use the WCP to monitor the status and set up the launch of one or more weapons. The WCP is linked to numerous subsystems and I/O devices in support of the launch operation; in particular, it interacts with the Launch Control System (LCS), which oversees launch preparation and determines when one or more weapons are to be launched. Operators use the control panel to open and close valves and doors and to monitor the doors, valves, and other system devices for faults. The panel consists of lights, numeric displays, and switches. The lights display information from sensors (e.g., a door is open, a subsystem has failed) and commands from the LCS (e.g., Make Launcher Ready). Numeric displays present numeric information, such as hydraulic pressure, read from sensors. Switches energize and de-energize *solenoids*, electromechanical devices that open and close doors and valves.

The contractor-developed SRS of the required behavior of the WCP is a combination of prose, diagrams, and formal descriptions. In the SRS, the names, types, and source or destination (device or subsystem) of all system inputs and outputs are presented in tables. These tables describe a total of 198 input and output variables. Of these, 99 are Boolean inputs, nine are analog inputs, 83 are Boolean outputs, and seven are real-valued outputs. In addition to the 198 input and output variables, the SRS includes 60 internal variables. Although most of the internal variables are Boolean, a few are real-valued.

We have reviewed SRSs for many military systems. The SRS for the WCP is the highest quality requirements specification we have encountered. Whereas most SRSs for military systems consist largely of prose, the SRS for WCP formally specifies the values of most WCP outputs and internal variables, using a set of "logic equations". A logic equation describes each *internal variable* as a function of inputs and other internal variables and each *output* as a function of system inputs, the WCP internal variables, and other system outputs.

Although logic equations describe most outputs and internal variables, the SRS describes other variables more informally. For example, it uses informal prose to describe all numeric outputs and the ten Boolean variables (both outputs and internal variables) not defined by logic equations. It also provides prose descriptions of four modes of operation (Initialization, Monitor, Operate, and Test) and how the values of various outputs and internal variables depend upon the mode (e.g., relays are disabled in Monitor mode). With only a few exceptions, the logic equations do not refer to modes.

The WCP is safety-critical. That is, incorrect system behavior can cause serious accidents, such as preparing the launch of a weapon under hazardous conditions. To prevent behavior that could lead to serious accidents, the SRS contains precise prose de-

scriptions of six properties that the WCP must satisfy to operate safely. For example, the first safety property in the SRS states, "*For* cVENT_SOLENOID *to open, the differential pressure must fall within the interval* [kMinTRANS_OK, kMaxTRANS_OK]." (In this expression, kMinTRANS_OK and kMaxTRANS_OK are constants.) Our interpretation of this property is that the vent valve of the weapon launcher shall open only if the differential pressure indicated by at least one of the two transducers is within safe limits. This property is referred to below as property q.

SCR OVERVIEW

Presented below is a brief introduction to the SCR model and a summary of the characteristics that distinguish the SCR method from other formal methods. For more information about SCR, see [18, 19, 3, 16].

SCR Requirements Model

The SCR requirements model includes a set $RF = \{r_1, r_2, \ldots, r_n\}$ containing the names of all variables in a given specification and a function TY which maps each variable to the set of its legal values. In the model, a *state* s is a function that maps each variable in RF to its value.

Two important constructs in SCR specifications are conditions and events. A *condition* is a predicate defined on a system state, and an *event* is a predicate defined on two consecutive system states that indicates a change in system state. We say that "an event occurs" when the value of any variable changes. The notation "@T(c)" denotes an event and is defined by

$$@T(c) \stackrel{\text{def}}{=} \neg c \wedge c',$$

where the unprimed condition c is evaluated in the current state and the primed condition c' is evaluated in the new state. Informally, "@T(c)" means that condition c becomes true and "@F(c)" means that c becomes false.

The SCR model describes a system Σ as a state machine $\Sigma = (E^m, S, S_0, T)$, where E^m is the set of possible input events, S is the set of possible system states, S_0 is the set of initial states, and T is the transform (i.e., next-state) function. The condition tables and event tables in an SCR specification define a set of functions F_1, F_2, ..., F_n, each of which computes the updated value of a state variable. When the system receives a new input event $e \in E^m$, the next-state function T composes the F_i to map the current state $s \in S$ and the new input event e to the new state $s' \in S$.

SCR Method: Distinguishing Characteristics

The SCR method differs from other methods in a number of ways:

- The SCR technology provides a relatively seamless integration of analysis tools (e.g., consistency checking, simulation, model checking), all based on the same formal semantics.
- Unlike the model used in [8], the SCR model may be applied to an entire system (or system component) rather than to individual functions.
- Because SCR specifications are executable and because the SCR toolset provides facilities for building a customized interface, a developer can derive a working system prototype from an SCR requirements specification.
- The SCR tools provide a variety of visual representations. In particular, most functions are defined by tables; the dependencies among the monitored, controlled, and internal variables in the specification are represented as a graph; and the user interface used to drive the simulator (e.g., the operation interface in the case of the WCP) is a set of animated pictures that simulate the behavior of the actual system.
- SCR uses terms understandable to most developers rather than terminology (e.g., signatures of functions) that is more oriented to users who are mathematically sophisticated.

APPLYING SCR TO THE SRS

Described below is the process used to translate the SRS into SCR, the way in which the SCR tools were used to analyze the specification and to produce a working prototype, and the effort required to develop and analyze the SCR specification of the WCP.

Translation into SCR

The inclusion of input, internal, and output variables and of logic equations makes the WCP SRS highly compatible with the SCR requirements method. In SCR requirements specifications, system inputs are represented by *monitored variables*, system outputs by *controlled variables*, and internal variables by *terms* and *modes*. We translated each logic equation to either an SCR condition table or an SCR event table.

To obtain an online specification of the SRS, we scanned the variable tables and the logic equations from the SRS into a computer file and used optical character recognition to convert the scanned images to ASCII. To obtain an SCR specification, we edited the variable names slightly (the SRS contains variable names with embedded blanks) and translated the results (including traceability links to the SRS) into SCR tables. This translation process was easy because the system model that underlies the WCP SRS closely matches the state machine model that underlies SCR specifications.

Applying the SCR Tools

We then used our software tools to display and analyze the SCR specification of the WCP. Analysis with our consistency checker exposed a few missing cases and numerous inconsistencies in the definitions and uses of variable names. To the extent feasible, we corrected the variable name discrepancies and other minor problems in the SCR specification.

As part of this study, we also developed a simulator front-end, customized for the WCP. To build the front-end, we scanned in diagrams of the operator control panel that were included in the WCP system requirements specification. We then used an interface builder to create widgets for the switches, lights, dials, and other displayed objects and superimposed them on the control panel diagrams. Each object was displayed in the colors indicated in the WCP documentation.

Once the customized interface was connected to the SCR requirements specification, we had a working prototype of the WCP. Unlike most prototypes, which are constructed in an ad hoc manner, this prototype has both a realistic operator interface *and* a complete formal specification of the required system behavior. Such a prototype has many uses. For example, operators can validate the specification, and the interface design as well, by using the prototype to execute key scenarios. Detected problems can be corrected by changing either the user interface or the underlying SCR specification.

Finally, we used our tools to analyze the SCR specification for the safety property q. Interestingly, property q and the five other safety properties contained in the SRS are *transition* or *two-state* invariants, that is, each is a property of every pair of reachable states (s, s'), where $s, s' \in S$ and there exists an enabled input event $e \in E^m$ such that $T(e, s) = s'$.

To analyze property q, we translated q, which the SRS describes in prose, into propositional logic and then used our abstraction methods to generate a reduced model of the WCP [5, 4]. Invoking the explicit state model checker Spin [22] on the reduced model exposed a violation of the property and a counterexample, i.e., a sequence of input events that leads to two states in which the violation occurs.

To demonstrate the violation and to ensure that the detected error was not spurious, we manually translated the sequence of input events leading to the violation back into the original specification and then ran the scenario through the SCR simulator. Executing the scenario detected a violation of property q, thus showing that the violation detected by model checking was not spurious. Recently, a safety engineer, familiar with WCP, confirmed that our methods appear to have uncovered a true safety violation.

Time Required to Apply SCR

One of the most significant aspects of this project was the short time required to develop the SCR specification and to apply the SCR tools. Translating the contractor-produced specification into an SCR specification required only one person-week. Given that the contractor developed the original SRS without any knowledge of SCR, the small investment in time and effort required to produce an SCR specification from the SRS is noteworthy and, in our view, clearly demonstrates the cost-effectiveness of SCR. Building a customized interface for the SCR simulator required approximately three weeks, and use of the consistency checker and the model checker less than a day. Hence, the total time required to develop and analyze the SCR specification and to build a working prototype was approximately one person-month.

This small investment in time and effort is significant, especially when compared to the time and effort expended in two other recent projects. Both projects applied a formal method based on the mechanical prover PVS [26] to a practical system. In the first project, Dutertre and Stavridou used PVS to specify and analyze the requirements of an Air Data Computer (ADC). The specification and analysis of the ADC required approximately 18 person-months [9] and detected a single error (a case in which a minimum exceeded a maximum). In a second project, Crow and Di Vito used PVS to specify and analyze the requirements of the Global Positioning System (GPS). This project required two staff-months spread over a four-month period [8] and detected many of the same kinds of errors (e.g., ambiguity, inconsistency, and incompleteness) that the SCR tools detect.

Light-weight techniques such as SCR, e.g., techniques that apply consistency checking or model checking, can expose many of the same errors detected by more heavy-duty techniques, such as PVS. However, applying the SCR tools does not require the mathematical sophistication and theorem proving skills needed to apply many heavy-duty techniques. Applying the SCR technology also requires significantly less human effort. Further, in contrast to PVS, the SCR tools can produce both a working system prototype *and* a build-to specification that can be used by software developers to design and implement the software.

Using the SCR notation and tools does not preclude the application of more heavy-duty techniques to SCR specifications. For example, in a few cases (e.g., proving that an abstraction of a given variable is valid), we have found that light-weight tools are insufficent and that deductive reasoning and mechanical provers such as PVS are useful. Recently, we used TAME [2], a system designed to analyze automata models using PVS, to check properties of an SCR specification of a simple

Bomb Release system. We found that the SCR analysis tools performed many of the checks (e.g., checking for nonoverlapping conditions) needed for the TAME results to be valid.

PROBLEMS IN THE TRANSLATION

This section discusses five major issues that arose in developing the SCR specification from the original SRS—how to provide missing requirements information without access to domain experts, how to interpret the prose and some of the logic equations in the original SRS, how to handle the SRS modes, when to deviate from the original SRS, and how to locate information in the SRS. Although others applying formal techniques to specifications of practical systems have encountered similar problems (e.g., lack of domain expertise, questions about the interpretation of the specifications, and difficulty locating information in the SRS), the considerable progress we made despite the problems is encouraging.

Lack of Domain Expertise

Developing and analyzing the SCR specification with our tools raised numerous questions about the WCP requirements. Our questions fell into two categories: questions about the WCP environment and questions about the required behavior of the WCP. We were unable to answer many of these questions because either the information was missing from the SRS or the information was included in the SRS but, despite considerable effort, we couldn't locate it. Although a WCP project officer and the safety engineers answered a few of our questions, we had no direct access to the WCP contractor, who could answer detailed questions about the WCP and the original SRS.

Largely missing from the specification was information about the WCP environment. For example, a safety property may be violated if two sensors fail. If the simultaneous failure of two sensors has very low probability, then such a violation could be spurious. Because the SRS fails to make such assumptions explicit, we cannot tell. Further, in specifying the required behavior of the WCP in SCR, we described the behavior of each monitored variable as a state machine. Because the SRS provides little information about the system inputs, which state transitions are legal for a particular input is unknown. To avoid erroneous decisions about how the system inputs change, we allowed each state machine representing an input to transition from one state to any other state even though, in many cases, a transition is impossible. For example, if a sensor measuring hydraulic pressure is functioning normally, a reading close to the minimum pressure in one state and another reading close to the maximum pressure in the next state is probably infeasible.

Other questions arose about the required system operation. For example, our consistency checker detected some missing cases. When the consistency checker detected missing cases, how to fill in the missing information was sometimes obvious; in other cases, supplying the missing information required domain expertise.

Uncertainty about the Interpretation

As described above, the SRS combines both prose and logic equations in describing the required behavior of the WCP. Like other researchers who have applied formal methods to practical systems (see, e.g., [10, 11]), we found considerable ambiguity in the prose sections of the SRS. In the SCR specification, we used what we believed to be a reasonable interpretation of the SRS prose. In some cases, we have informal feedback from the program manager and the safety engineers that our interpretation of the prose is accurate. However, a formal review of the SCR specification by the developers of the original SRS is highly desirable.

Another question that arose was how to interpret the logic equations. Because the logic equations are formal, we expected to have little difficulty interpreting them. This was a good assumption for most of the equations, i.e., those which represent the value of an output or internal variable as a function of other variables in the same state. Such logic equations can be directly translated into SCR condition tables. For example, consider the logic equation

```
cHYDRAULIC_PRESSURE_LOW_INDICATOR := mLAMP_CHECK=up or
                not(mHYDRAULIC_OIL_PRESSURE).
```

Table 1 shows how this equation can be translated into an SCR condition table. The table states that the output `cHYDRAULIC_PRESSURE_LOW_INDICATOR` is *true* if the input `mLAMP_CHECK` is `up` or if the input `mHYDRAULIC_OIL_PRESSURE` is *false*, and *false* otherwise.

Less obvious was how to translate the remaining logic equations, which define variables called *latches*, into SCR. Latches are functions of variables in both the current state and the new state. In SCR, such functions are defined by event tables. A logic equation for a latch x has the form

$$x := (y \vee x) \wedge z,$$

where y and z are other variables. We translate all logic equations of the above form into the event table shown below. This table states that x becomes true when both y and z are true and that x becomes false when z becomes false. All latches have this pattern.

	@T($y \wedge z$)	@F(z)
$x' :=$	true	false

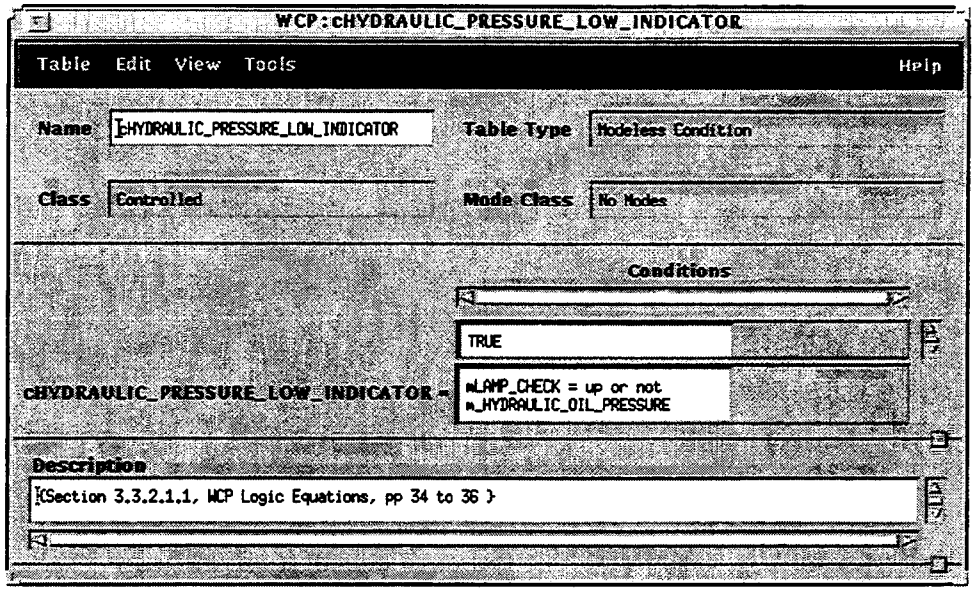

Table 1: Condition Table Defining cHYDRAULIC_PRESSURE_LOW_INDICATOR.

An example of a latch is the internal variable tPRESSURE_LATCH. The logic equation for this variable is given by

tPRESSURE_LATCH := (tPRESSURE_LATCH or mPRESSURE_HOLD) and tPRESSURE_AUTO.

In Table 2, the function defined by this logic equation is represented as an event table. The table states that tPRESSURE_LATCH becomes true if mPRESSURE_HOLD and tPRESSURE_AUTO become true and becomes false if tPRESSURE_AUTO becomes false (and is otherwise unchanged).

How to Deal with Modes

In SCR, each controlled variable is defined as a function of monitored variables, terms, and modes. As noted above, modes were not used systematically in the SRS: while the prose sections of the SRS describe mode-dependent behavior, only a handful of logic equations express a variable as a function of a mode and other variables. Most logic equations simply ignore the modes. Because the SRS does not handle modes systematically, we decided to follow the lead of the SRS and represent the WCP modes as Boolean terms rather than as SCR modes.

Deviating from the SRS

The decision not to use SCR modes to model the WCP modes was motivated by our strong desire to maintain close compatibility between the original SRS and the SCR specification that we derived from it. We consider compatibility to be important in demonstrating to both the project officer and the contractor that the problems exposed were bona fide problems rather than problems introduced by our translation.

Another case in which we needed to decide whether to deviate from the SRS involved implementation bias. An SCR requirements specification is designed to describe the required system behavior without making design and implementation decisions. In certain cases, the original SRS includes implementation bias. Whether to remove this implementation bias was an issue.

In this case, our decision was to remove implementation bias and thus to deviate slightly from the SRS. For example, the SRS defines the output cTEST_MODE_INDICATOR, which represents a light on the operator control panel, with a logic equation that depends on a flasher circuit. We decided that the required behavior was clearer if the SCR specification omitted the flasher circuit. Table 3 shows a condition table, which defines cTEST_MODE_INDICATOR as a function of the monitored variable mLAMP_CHECK and the internal variable tTEST_MODE. The table represents this output with three values, rather than two. In particular, it states that the light is on when the switch mLAMP_CHECK is up, off when the mLAMP_CHECK switch is down and the system is not in tTEST_MODE, and flashing when the mLAMP_CHECK switch is down and the system is in tTEST_MODE.

How to Find Information in the SRS

Using the SRS to answer questions about the WCP requirements often proved difficult because finding information quickly in the SRS depends on knowing how the WCP is organized. For example, the SRS organizes the inputs, outputs, and logic equations by subsystem rather than alphabetically. Moreover, the SRS defines some variable values only in the prose and, as

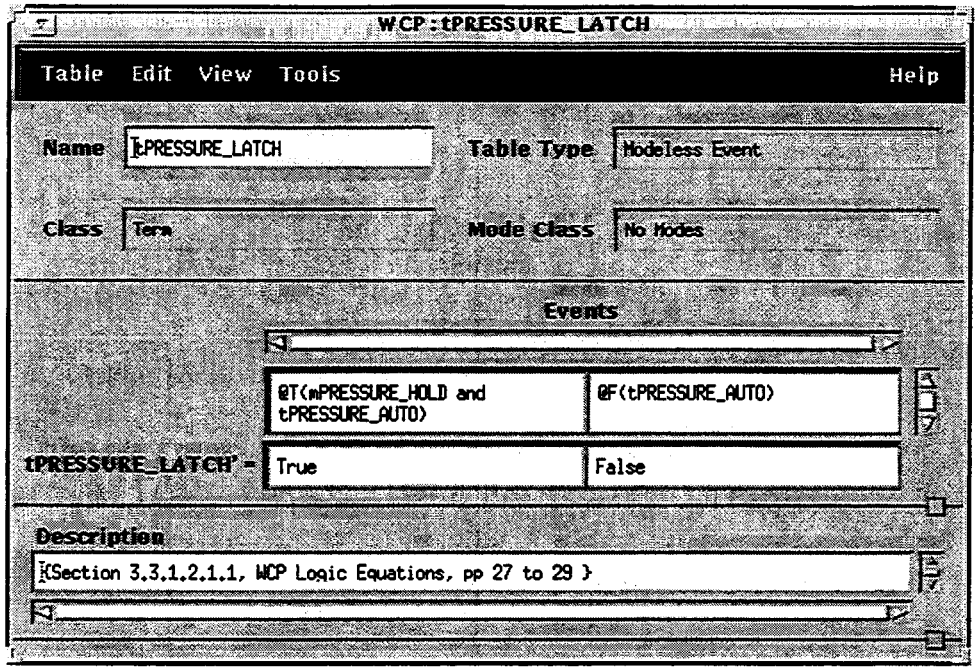

Table 2: Event Table Defining tPRESSURE_LATCH.

noted above, defines the modes of operation sometimes in prose and in a few cases in the logic equations.

Now that it is complete, we use the SCR specification, which has traceability links to the SRS, to find information in the SRS. For example, when we want to find a logic equation for some variable in the SRS, we go first to the table function in the SCR specification. Then, we use the traceability links in the SCR specification to determine the page of the SRS that contains the logic equation.

NEEDED ADDITIONS TO SCR

Analysis of the contractor-produced SRS for the WCP required us to add two new features to the SCR model checking capability, a new abstraction method and the use of simulation to validate suspected violations of properties exposed by model checking. Below, we briefly summarize the two new features as well as a minor feature that improves the readability of the safety properties. For more information about the two new features, see [17].

Using Abstraction in Model Checking

The number of reachable states in a state machine model of a practical system is usually very large, sometimes infinite. Hence, for realistic software specifications, most model checkers fail to terminate because they run out of memory. One promising approach proposed by Clarke et al. to combat state explosion in model checking is abstraction [7], which can theoretically reduce a huge (and even infinite) state space to a much smaller state space.

We have developed methods for deriving abstractions from SCR requirements specifications, each based on the formula to be analyzed. The methods are practical: none requires ingenuity on the user's part, and each derives a smaller, more abstract model automatically. Further, each method systematizes techniques that current users of model checkers routinely apply but in ad hoc ways.

To analyze the SCR specification of the WCP for the safety property q, we needed to build a more abstract model. To do so, we used abstraction to extract from the original SCR specification only those variables (and the tables that define them) which could affect the validity of q. By applying this abstraction method, we reduced the number of variables in the specification from 258 to 55, a reduction of almost 80%.

Because the reduced specification still contains several variables with infinitely many values, we also developed a new abstraction method which replaces a detailed variable in the original specification with a more abstract variable. The WCP has seven variables which are real-valued, two input variables and five internal variables. The two input variables record the values read by the two transducers referred to in the statement of property q (see above). The other five variables are functions of one or both of the two inputs. This abstraction method uses the next-state function T as well as the property q to compute a discrete (and therefore more abstract) version of each of these variables. In each case, we reduced the set of values a variable can assume from an infinite set of real numbers belonging to

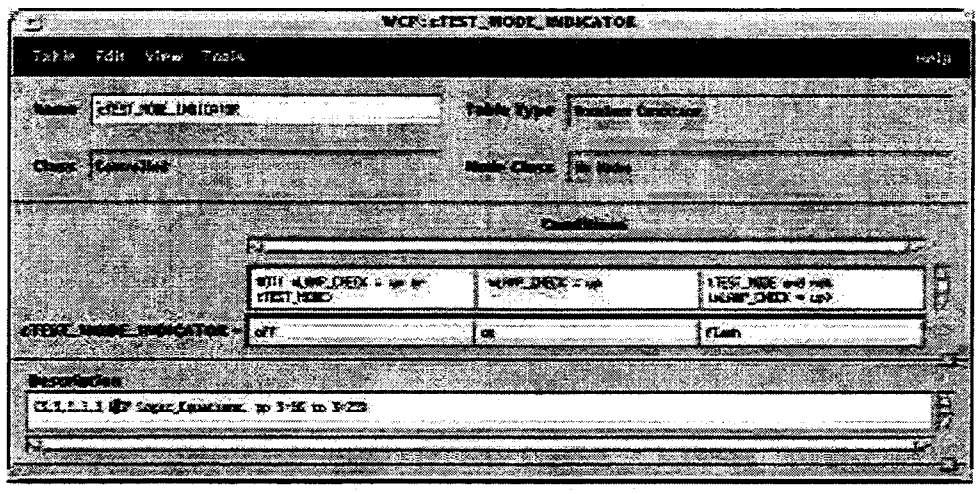

Table 3: Condition Table Defining cTEST_MODE_INDICATOR.

some interval I to a discrete set of three to seven values, each of which represents a subinterval of the original interval I. By applying this new abstraction method, we reduced the size of the state space of the WCP specification from infinite to finite.

Using Simulation for Validation

In analyzing the WCP specification for violations of the first safety property, we needed to combine model checking and simulation. Simulation was used not only to demonstrate the violation but also for validation. Because one of the abstraction methods was applied manually and because the translation of the counterexample produced by model checking into a scenario in the original SCR specification was manual, errors could have been introduced. Moreover, the abstract specification that we developed was not *complete*, that is, in some cases, a counterexample in the reduced specification had no counterpart in the complete specification. Hence, the violation detected by model checking could be spurious. Running the scenario through our simulator validated that the violation detected by model checking exposed a true safety violation in the complete specification.

Improving the Readability of the Properties

Originally, we expressed the safety property q as a logical expresssion containing unprimed and primed variables. (Recall that an unprimed variable represents the value of the variable in the old state, whereas a primed variable represents the value in the new state.) Because the variable names chosen by the contractor were very long, we found that the formal statement of the safety properties in the SCR specification was difficult to understand. To make the statement of the safety properties more concise, we substituted the SCR notation for events, "@T(c)," for expressions of the form "NOT c AND c'". This more concise formal statement of the safety properties makes them easier to understand.

REACTIONS TO THE SCR RESULTS

Our long-term goal is to transfer the SCR technology into industrial software development. The difficulty of achieving this goal was illustrated by our experience in translating the contractor-developed SRS into SCR and in discussing our SCR specification and the defects that our tools uncovered with a WCP project officer, two safety engineers, and a representative of the contractor who developed the SRS.

Reactions to the Tools

The project officer and the safety engineers were quite positive about the need for software tools to aid in developing an SRS. They were particularly interested in using our tools to construct system prototypes (see below). The WCP contractor was indifferent. Some possible reasons for this indifference are suggested below.

Reactions to the Defects

As described above, the SCR methods and tools identified many defects in the SRS: some were detected in creating the SCR specification, others by running the consistency checker, and still others by applying model checking. Neither the military personnel nor the contractor were surprised to learn that our tools uncovered many defects. All expected vagueness, inconsistency, and incorrectness in the SRS. Prior to learning about our technology, they believed that no alternative to a vague, incorrect, inconsistent SRS was feasible.

Both the project officer and the safety engineers support the integration of technology like the SCR technology into the military's software development process. Clearly, detecting errors early in software development can significantly reduce the cost of software development and improve software quality. Moreover, the safety engineers view our method for detecting safety violations in the SRS as quite promising. They are cur-

rently evaluating the utility of the SCR tools for specifying and analyzing safety-critical military systems.

Unlike the military personnel, the contractor representative was polite but quite disinterested in the SCR technology. Several possible reasons for this apathy exist. First, because many vendors have made exaggerated claims about varied tools and techniques which have not materialized in practice, many developers are highly skeptical of the effectiveness of new software technology. Second, the SCR tools are prototypes and hence do not provide the industrial-quality support (e.g., training, installation, bug fixes, etc.) that the contractor requires.

Finally, the contractor already has a very different, more traditional software development process in place which has produced acceptable military systems. In this process, the SRS is a deliverable that is produced at low cost. Underlying the contractor's process is the assumption that most software errors will be detected by system testing *after* the code has been generated. Little incentive exists for the contractor to uncover errors during the requirements phase. As noted recently in *IEEE Software* [6], "Current government contracts typically pay contractors by the staff-hour, which provides little incentive to reduce expensive rework."

Reaction to the Customized Simulator

Both the project officer and the safety engineers were very positive about the working system prototype that the SCR technology produced. They view symbolic execution as an effective means of validating the SRS. As a result, we have plans for system operators, knowledgeable about weapons preparation and launch, to use the customized simulator to execute scenarios of interest and thus to validate that the SCR specification captures the correct system behavior. Once confidence in the specification and the interface design is high, this "prototype" can be used to train operators and to develop initial operational procedures.

Our experience is that audiences prefer examples from their problem domain. For example, military audiences dislike typical academic examples (e.g., automobile cruise control). Military personnel and contractors also want examples presented in a way that is easily comprehensible, i.e., with application interfaces and models with which they are already familiar. They usually dislike the abstract models presented by computer scientists.

SOME OBSERVATIONS

Listed below are some observations about our experience applying the SCR method and tools to the WCP SRS:

- Given a high-quality SRS, one can use the SCR tools to do considerable analysis *without* significant interaction with system experts *and* at very low cost. Clearly, applying the SCR technology requires precise, unambiguous information about the required behavior. If the SRS had been written in prose, the technology could not have uncovered so many defects nor could it have produced a working system prototype.
- Interaction with system experts is needed to validate that the interpretation of the SRS is correct and to confirm that the detected errors are real errors. As suggested above, some detected errors (e.g., the simultaneous failure of three sensors) have very low probability. The SRS should document such assumptions. Further, detailed descriptions of the system environment—the constraints imposed by physical laws and the environment in which the system operates—can rule out certain errors.
- Executable specifications are extremely valuable. The customized simulation of the WCP convinced military personnel that our specification captured the essential system behavior and thus gave our research group and our technology increased credibility.
- The only properties of interest for the WCP were invariants, in particular, two-state invariants. The many other properties that can be expressed in, e.g., temporal logic were not needed.

SUMMARY AND CONCLUSIONS

We were able to use the SCR tools to capture, analyze, and manipulate a contractor-produced specification that was not developed with the SCR method in mind. In the process of applying the SCR method, we uncovered a number of problems with the SRS, some serious. Many of these problems have been fixed, but domain expertise is needed to correct the remaining problems.

Applying the SCR method and tools to the WCP SRS and to the two other pilot projects described above demonstrated several advantages of SCR:

- Applying the SCR technology had high payoff with only a small investment in human time and effort. The primary reason for the high payoff is that the SCR technology is very well suited to control-intensive systems, such as the WCP.
- The SCR methods and tools are usable by people other than the SCR developers. In the case of both the NASA space system application and Rockwell's flight guidance system, people outside our group were able to use the SCR technology to do productive work.

- Use of the SCR notation and tools can facilitate the use of more heavy-duty techniques. Deductive reasoning, and mechanical provers that support such reasoning, can be easily applied to SCR specifications. As noted above, we recently used a system called TAME to analyze an SCR specification using deductive reasoning and the PVS prover.
- Unlike many other formal methods, the SCR tools provide feedback in terms understandable to users.

Listed below are several planned follow-on tasks involving the WCP and its software requirements:

- Translate the remaining five safety properties into logical formulas and apply our abstraction coupled with model checking and simulation to determine whether the SCR specification of the WCP satisfies the other five properties.
- Study the desirability and feasibility of using the logic equations as a notation supported by the SCR tools. Translating the logic equations into the SCR semantics should be easy. The difficulty lies in providing user feedback in terms of logic equations, rather than tables, when the tool detects an error.
- Study the feasibility of developing a more standard SCR specification from the SRS for the WCP. This SCR specification would represent the requirements in terms of SCR mode classes and eliminate some remaining redundancy in the current SRS. It would also provide a good example of an SRS that was developed directly using our tools.
- Organize the SCR specification of the WCP for ease of change. Due to the large number of variables it contains, the current SCR specification of the WCP is quite difficult to understand. A carefully thought-out reorganization of the specification should make the specification not only more understandable but also easier to modify.

An important remaining question is how to transfer formal methods technology, such as the SCR technology, into industry. Currently, contractors have little economic incentive to apply formal methods during the requirements phase. Hence, government program managers must provide that incentive. They can do so by writing contracts that require developers

- to demonstrate before the software is designed that the behavior specified by the SRS is complete and consistent and that the SRS does not require nor allow the software to violate specified safety conditions, and
- to provide a means for the government to validate the SRS before the software is designed.

Towards this end, we are working on language that government program managers can include in contracts. We are also seeking influence in organizations that program managers look to for guidance (e.g., "Project Breathalyzer" sponsored by the Software Program Managers Network).

ACKNOWLEDGMENTS

We gratefully acknowledge Cheryl Sarteschi's development of the customized front-end for the simulator. We also acknowledge Myla Archer and Ralph Jeffords for their detailed comments on a draft of this paper. We also appreciate the constructive comments of the anonymous referees.

REFERENCES

[1] T. A. Alspaugh, S. R. Faulk, K. H. Britton, R. A. Parker, D. L. Parnas, and J. E. Shore. Software requirements for the A-7E aircraft. Technical Report NRL-9194, Naval Research Lab., Wash., DC, 1992.

[2] M. Archer and C. Heitmeyer. TAME: A specialized specification and verification system for timed automata. In *Proc., Real-Time Systems Symposium Work-in-Progress Session*, 1996.

[3] R. Bharadwaj and C. Heitmeyer. Applying the SCR requirements method to a simple autopilot. In *Proc., Fourth NASA Langley Formal Methods Workshop (Lfm97)*, 1997.

[4] R. Bharadwaj and C. Heitmeyer. Model checking complete requirements specifications using abstraction. Technical Report 97-7999, Naval Research Lab., Wash., DC, 1997.

[5] R. Bharadwaj and C. Heitmeyer. Verifying SCR requirements specifications using state exploration. In *Proc., First ACM SIGPLAN Workshop on Automatic Analysis of Software*, 1997.

[6] N. Brown. Industrial-strength management strategies. *IEEE Software*, pages 94–103, July 1996.

[7] E. Clarke, O. Grumberg, and D. Long. Model checking and abstraction. In *Proc., Principles of Programming Languages (POPL)*, 1994.

[8] J. Crow and B. L. Di Vito. Formalizing space shuttle requirements. In *Proc. of FMSP'96, The 1st Workshop on Formal Methods in Software Practice*, 1996.

[9] B. Dutertre and V. Stavridou. Formal requirements analysis of an avionics control system. *IEEE Transactions on Software Engineering*, 23(5):267–278, May 1997.

[10] S. Easterbrook and J. Callahan. Formal methods for V & V of partial specifications. In *Proc. 3rd Intern. Symposium on Requirements Engineering (RE '97)*, Annapolis, MD, Jan. 1997.

[11] S. Easterbrook and J. Callahan. Formal methods for verification and validation of partial specifications: A case study. *Journal of Systems and Software*, 1997.

[12] S. R. Faulk, L. Finneran, J. Kirby, Jr., S. Shah, and J. Sutton. Experience applying the CoRE method to the Lockheed C-130J. In *Proc. 9th Annual Conf. on Computer Assurance (COMPASS '94)*, pages 3–8, Gaithersburg, MD, June 1994.

[13] A. Flora-Holmquist and M. Staskauskas. Moving formal methods into practice: The VFSM experience. In *Proc. of FMSP'96, The 1st Workshop on Formal Methods in Software Practice*, 1996.

[14] A. Hall. Using formal methods to develop an ATC information system. *IEEE Software*, pages 66–76, Mar. 1996.

[15] M. P. E. Heimdahl and N. Leveson. Completeness and consistency analysis of state-based requirements. In *Proc. of 17th Int'l Conf. on Softw. Eng. (ICSE '95)*, pages 3–14, Seattle, WA, Apr. 1995. ACM.

[16] C. Heitmeyer, J. Kirby, and B. Labaw. Tools for formal specification, verification, and validation of requirements. In *Proc. 12th Annual Conf. on Computer Assurance (COMPASS '97)*, Gaithersburg, MD, June 1997.

[17] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using model checking and simulation to detect a safety violation in a control system specification. Submitted for publication.

[18] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Tools for analyzing SCR-style requirements specifications: A formal foundation. Technical Report NRL-7499, Naval Research Lab., Wash., DC. In preparation.

[19] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Transactions on Software Engineering and Methodology*, 5(3):231–261, April–June 1996.

[20] K. L. Heninger. Specifying software requirements for complex systems: New techniques and their application. *IEEE Trans. Softw. Eng.*, SE-6(1):2–13, Jan. 1980.

[21] S. D. Hester, D. L. Parnas, and D. F. Utter. Using documentation as a software design medium. *Bell System Tech. J.*, 60(8):1941–1977, Oct. 1981.

[22] G. J. Holzmann. The model checker SPIN. *IEEE Trans. on Softw. Eng.*, 23(5):279–295, May 1997.

[23] R. Kurshan. Formal verification in a commercial setting. In *Proc., Design Automation Conference*, 1997.

[24] S. Meyer and S. White. Software requirements methodology and tool study for A6-E technology transfer. Technical report, Grumman Aerospace Corp., Bethpage, NY, July 1983.

[25] S. Miller. Specifying the mode logic of a flight guidance system in CoRE and SCR. In *Proc. of FMSP'98, The 2nd Workshop on Formal Methods in Software Practice*, 1998.

[26] S. Owre, J. Rushby, N. Shankar, and F. von Henke. Formal verification for fault-tolerant architectures: Prolegomena to the design of PVS. *IEEE Transactions on Software Engineering*, 21(2):107–125, Feb. 1995.

[27] A. J. van Schouwen, D. L. Parnas, and J. Madey. Documentation of requirements for computer systems. In *Proc. RE'93 Requirements Symp.*, pages 198–207, San Diego, CA, Jan. 1993.