In *Proceedings, Second IEEE Workshop on Industrial Strength Formal Specification Techniques,* (WIFT'98), Boca Raton, FL, Oct. 19, 1998.

Using the SCR* Toolset to Specify Software Requirements

Constance Heitmeyer
Naval Research Laboratory (Code 5546)
Washington, DC 20375

1. Background

Formulated in the late 1970s to specify the requirements of the Operational Flight Program (OFP) of the A-7 aircraft [8], the SCR (Software Cost Reduction) requirements method is a method based on tables for specifying the requirements of software systems. During the 1980s and the early 1990s, many companies, including Bell Laboratories, Grumman, Ontario Hydro, and Lockheed, applied the SCR requirements method to practical systems. Each of these applications of SCR had, at most, weak tool support. To provide powerful, robust tool support customized for the SCR method, we have developed the SCR* toolset. To provide formal underpinnings for the method, we have also developed a formal model which defines the semantics of SCR requirements specifications.

2. The SCR Requirements Model

In SCR, *monitored* and *controlled variables* represent environmental quantities that the system monitors and controls [7]. The environment nondeterministically produces a sequence of input events, where an *input event* is a change in some monitored quantity. The system responds to each input event in turn by changing state and possibly changing one or more controlled quantities.

The SCR formal model, a special case of the classic state machine model, represents a system as a 4-tuple, (S, S_0, E^m, T), where S is a set of states, $S_0 \subseteq S$ is the initial state set, E^m is the set of input events, and T is a function describing the allowed state transitions [7]. T is a composition of simpler functions derived from the tables in an SCR specification. The formal model requires each table to satisfy certain properties. These properties guarantee that each table describes a total function. To specify the system requirements concisely, the SCR method uses mode classes, conditions, and events. A *mode class* organizes the system states into equivalence classes, each called a *mode*. The SCR model includes a set RF containing the names of all variables (e.g., monitored and controlled variables, mode classes) in a given specification and a type function TY mapping each variable in RF to a set of values. In the model, a *state* is a function mapping each variable r in RF to its value in TY(r), a *condition* is a predicate defined on a state, and an *event* is a predicate defined on two states when any state variable changes.

3. The SCR Tools

SCR* is an integrated suite of tools supporting the SCR requirements method. It includes a *specification editor* for creating a requirements specification, a *dependency graph browser* for displaying the variable dependencies, a *consistency checker* for detecting well-formedness errors (e.g., missing cases), a *simulator* for validating the specification, a *model checker* and *theorem prover* for checking application properties, and an *invariant generator* for automatically constructing state invariants from the SCR tables.

Specification Editor. To create and modify a requirements specification, the user invokes the specification editor [4]. Each SCR specification is organized into dictionaries and tables. The dictionaries define the static information in the specification, such as the names and values of variables and the user-defined types. The tables specify how the variables change in response to input events. Eacj

Dependency Graph Browser. Understanding the relationship between different parts of a large specification can be difficult. To address this problem, the Dependency Graph Browser (DGB) represents the dependencies among the variables in a given SCR specification as a directed graph [6]. By studying the graph, a user can detect errors such as undefined variables and circular definitions. He can also use the DGB to extract and analyze subsets of the dependency graph, e.g., the subgraph containing all variables upon which a selected controlled variable depends.

Consistency Checker. The consistency checker [7, 6] analyzes a specification for consistency with the SCR

requirements model. It exposes syntax and type errors, variable name discrepancies, missing cases, nondeterminism, and circular definitions. When an error is detected, the consistency checker provides detailed feedback to facilitate error correction. A form of static analysis, consistency checking avoids executing the specification and reachability analysis. In developing an SCR specification, the user normally invokes the consistency checker first and postpones more expensive analysis, e.g., model checking, until later.

Simulator. To validate a specification, the user can run the simulator [6] and analyze the results to ensure that the specification captures the intended behavior. Additionally, the user can define invariant properties believed to be true of the required behavior and, using simulation, execute a series of scenarios to determine if any violate the invariants. The simulator supports the rapid construction of front-ends, customized for particular application domains. For example, we have constructed a front-end that simulates a cockpit display. Pilots can use the simulated cockpit display to evaluate an attack aircraft specification. By interacting with this display, the pilot moves out of the world of requirements specification and into the world of attack aircraft, where he is the expert. Such an interface facilitates customer validation of the specification. A second customized front-end, part of the weapons system prototype mentioned below, has also been developed.

Model Checker. After using SCR* to develop a requirements specification, a developer can invoke the Spin model checker [9] to check properties of the specification. Once a property violation is detected, the user can run the simulator to demonstrate and validate the violation. To make model checking practical, we have developed sound methods for deriving abstractions from SCR specifications [2, 5]. The methods are practical: none requires ingenuity on the user's part, and each derives a smaller, more abstract specification automatically. Based on the property to be analyzed, these methods eliminate unneeded detail from the specification.

Theorem Prover. When model checking fails to reveal an error in a requirements specification or produces many spurious counterexamples, the user may use mechanical theorem proving to establish the property. We have in fact done this for a small SCR specification, using the mechanical prover in PVS. For details, see [1].

Invariant Generator. Recently, a prototype tool that automatically generates state invariants from SCR tables [10] was integrated into SCR*. This tool has generated more than 20 interesting state invariants from the mode tables in a revised version of the A-7 requirements document.

4. Applying SCR* to Practical Systems.

To date, SCR* has been applied in several pilot projects. In one project, NASA researchers used SCR* to detect missing cases and nondeterminism in the prose software requirements specification of the International Space Station [3]. In a second project, Rockwell engineers used SCR* to expose 24 errors, many of them serious, in the requirements specification of a flight guidance system [11]. Of the detected errors, a third were uncovered in constructing the specification, a third by the consistency checker, and the remaining third with the simulator. In a third project, NRL applied SCR* to a sizable contractor-produced requirements specification of the Weapons Control Panel (WCP) of a safety-critical US military system [5]. The tools uncovered numerous errors in the specification, including a safety violation. Translating the specification into the SCR tabular notation, using SCR* to detect errors, and building a WCP prototype required only one person-month, thus demonstrating the utility and cost-effectiveness of the SCR method.

To date, more than 40 industry and government organizations and 30 universities in the US, the UK, Canada, and Germany are experimenting with SCR*. Moreover, a growing number of universities are incorporating SCR* into their software engineering courses.

5. Learning More About SCR*

To obtain a copy of the SCR* toolset and a draft toolset user guide, send a request to labaw@itd.nrl.navy.mil. For more information about SCR*, see our web site at

http://www.chacs.itd.nrl.navy.mil/SCR

References

[1] M. Archer, C. Heitmeyer, and S. Sims. TAME: A PVS interface to simplify proofs for automata models. In *Proc. User Interfaces for Theorem Provers*, Eindhoven, Netherlands, July 1998. Eindhoven Univ. Tech. Report.

[2] R. Bharadwaj and C. Heitmeyer. Model checking complete requirements specifications using abstraction. *Automated Software Engineering Journal*, 6(1), Jan. 1999.

[3] S. Easterbrook and others. Experiences using lightweight formal methods for requirements modeling. *IEEE Transactions on Software Engineering*, 24(1), Jan. 1998.

[4] C. Heitmeyer, A. Bull, C. Gasarch, and B. Labaw. SCR*: A toolset for specifying and analyzing requirements. In *Proc. 10th Annual Conf. on Computer Assurance*, June 1995.

[5] C. Heitmeyer, J. Kirby, B. Labaw, M. Archer, and R. Bharadwaj. Using abstraction and model checking to detect safety violations in requirements specifications. *IEEE Trans. on Softw. Eng.*, 24(11), Nov. 1998.

[6] C. Heitmeyer, J. Kirby, Jr., and B. Labaw. Tools for formal specification, verification, and validation of requirements. In *Proc. 12th Annual Conf. on Computer Assurance*, June 1997.

[7] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw. Automated consistency checking of requirements specifications. *ACM Trans. on Softw. Eng. and Methodology*, 5(3), 1996.

[8] 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.

[9] G. J. Holzmann. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[10] R. Jeffords and C. Heitmeyer. Automatic generation of state invariants from requirements specifications. In *Proc. 6th ACM Symp. on Foundations of Softw. Eng.*, Nov. 1998.

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