SOFTWARE COST REDUCTION

Constance L. Heitmeyer

- 4. 27. 장범석 -
WHAT IS SCR

• Set of techniques for designing software systems
  – Two formal models
    • Four variable model
    • SCR requirements model
  – Set of software tools for analyzing SCR-style requirements documents
WHAT IS SCR

• Software should be designed using “Rational Design process”

• Stages
  – Work product
    • Requirements document or a design document.
    • Each work product is associated with criteria that the work product must satisfy and a description of the information that the work product contains.
FOCUSES ON

• SCR techniques for constructing and evaluating the requirements document.
• The work product built during the requirements stage of software development.
In additional

• SCR approach to software design focusing on the design and documentation of
  – the module structure
  – the module interfaces
  – the uses hierarchy.
A-7 Requirement

• The focus on output.
• A special tabular notion for specifying each output.
• Set of criteria for evaluating a requirement document.
• document introduced Condition, event, mode, and terms.
Four Variable Model

• Generalized SCR techniques for writing requirements.

• This model represents the required behavior of a software system in terms of
  – Four sets of variables
    • Monitored, Controlled, Input, Output
  – Four relations
    • NAT, REQ, IN, OUT
Four Relation, ideal step

• NAT
  – Assumption about system behavior
    • i.e., constraints imposed on the monitored and controlled and quantities by physical laws and the system environment.

• REQ
  – Aspects of the environment that the system is expected to control.
    • i.e., how the system is required to change the controlled quantities in response to changes in the monitored quantities.
Four Relation, tolerance step

• **IN**
  – The tolerances on the monitored quantities as a mapping from the monitored quantities to the input quantities.

• **OUT**
  – The tolerances on the controlled quantities as a mapping from the controlled quantities to the output quantities.
Four-Variable Model

- Depiction of Four-Variable Model
Four Variable Model, Example

• Turns safety injection on and off in a nuclear power plant. (SIS)
  – monitors water pressure and adds coolant to the reactor core when the pressure falls below some threshold.
  – A drop in water pressure below the constant Low causes the SIS to enter mode TooLow; an increase in water pressure above a larger constant Permit causes the system to enter mode High.
  – A system operator blocks safety injection by turning a “Block” switch to On and resets the SIS after blockage by turning a “Reset” switch to On.
  – An assumption in the SIS is that water pressure ranges between 0.0 and 2000.0 psi (pounds per square inch).
Four Variable Model, Example

• Represented in SCR
  – Monitored
    • WaterPress, Block, Reset
  – Controlled
    • SafetyInjection
  – Mode
    • Pressure
      – TooLow, Permitted, and High
  – Term
    • Overridden: safety injection is blocked.
Four Variable Model, Example

- Represented in SCR
  - NAT
    - \textbf{WaterPres} is a non-negative
    - No greater than 2000.0
    - Low < Permit
  - REQ
    - Mathematical function of values of the three dependent variables, Pressure, Overridden, and SafetyInjection.
Four Variable Model, Example

• Represented in SCR
  – IN
    • If sensor is required to measure WaterPres within 1 psi, then the predicate
      • |WaterPres - w| < 1 is part of IN.
  – OUT
    • Signal to “Block” switch. {on, off}
    • Signal to “Reset” switch. {on, off}
SCR Tables

• Condition Table
  – defines a variable as a function of a mode and a condition.

• Event Table
  – defines a variable as a function of a mode and a event.

• Mode Transition Table
  – A special case of an event table.
SCR Tables

• Mode Transition Table (for Pressure)
  – Describing mode class **Pressure**.
  – A drop in water pressure below the constant **Low** causes the SIS to enter mode **TooLow**; an increase in water pressure above a larger constant **Permit** causes the system to enter mode **High**.

<table>
<thead>
<tr>
<th>Old Mode</th>
<th>Event</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>TooLow</td>
<td>( @T(\text{WaterPres} \geq \text{Low}) )</td>
<td>Permitted</td>
</tr>
<tr>
<td>Permitted</td>
<td>( @T(\text{WaterPres} \geq \text{Permit}) )</td>
<td>High</td>
</tr>
<tr>
<td>Permitted</td>
<td>( @T(\text{WaterPres} &lt; \text{Low}) )</td>
<td>TooLow</td>
</tr>
<tr>
<td>High</td>
<td>( @T(\text{WaterPres} &lt; \text{Permit}) )</td>
<td>Permitted</td>
</tr>
</tbody>
</table>

* \( @T(c) \) – c become TRUE
  \( @F(c) \) – c become FALSE
SCR Tables

• Event Table (for Overridden)
  – Describing the term **Overridden**.
  – A system operator blocks safety injection by turning a "**Block**" switch to On and resets the SIS after blockage by turning a "**Reset**" switch to On.

<table>
<thead>
<tr>
<th>Mode Class Pressure</th>
<th>Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>@F(Pressure=High)</td>
</tr>
<tr>
<td>TooLow, Permitted</td>
<td>@T(Block=On)</td>
</tr>
<tr>
<td></td>
<td>WHEN Reset=Off</td>
</tr>
<tr>
<td>Overridden</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>False</td>
</tr>
</tbody>
</table>
SCR Tables

• **Condition Table (for Safety Injection)**
  - Describing controlled variable **Safety Injection**.
  - A system operator blocks safety injection by turning a "**Block**" switch to **On** and resets the SIS after blockage by turning a "**Reset**" switch to **On**.

<table>
<thead>
<tr>
<th>Mode Class Pressure</th>
<th>Conditions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High, Permitted</td>
<td>True</td>
</tr>
<tr>
<td>TooLow</td>
<td>Overridden</td>
</tr>
<tr>
<td>Safety Injection</td>
<td>Off</td>
</tr>
<tr>
<td></td>
<td>False</td>
</tr>
<tr>
<td></td>
<td>NOT Overridden</td>
</tr>
<tr>
<td></td>
<td>On</td>
</tr>
</tbody>
</table>
SCR Requirements Model

• The purpose
  – To assign a precise semantics to the constructs and notation in SCR requirements specifications.
  – To provide a formal foundation for mechanized analysis of the specifications

Represent a system as state machine.
SCR Requirements Model

• System state
  – function mapping each state variable to a type-correct value
    
    \[
    \begin{align*}
    TY(\text{Pressure}) & = \{\text{TooLow, Permitted, High}\} \\
    TY(\text{WaterPres}) & = \{0, 1, 2, \ldots, 2000\} \\
    TY(\text{Overridden}) & = \{\text{true, false}\} \\
    TY(\text{Block}) & = \{\text{On, Off}\}.
    \end{align*}
    \]

• Conditioned Event
  – a condition is a predicate on a single system state and an event a predicate on two system states

\[
\@T(c) \ \text{WHEN} \ d \ \overset{\text{def}}{=} \neg c \ \land \ c' \ \land \ d,
\]
SCR Requirements Model

- **Software System** $\Sigma = (S, S_0, E^m, T)$
  - $S$ is a set of states.
  - $S_0 \in S$ is the initial state set.
  - $E^m$ is the set of monitored event.
  - $T$ is state transitions.
  - Maps a monitored event $e$ in $E^m$ and a state $s$ in $S$ to new state $S'$
  - Derived from condition, event, and mode transition tables.
SCR Requirements Model

• Basic assumption
  – One input assumption
    • Exactly one monitored event occurs at each state transition.
  – Synchrony Assumption
    • Completely process each monitored event before it processes the next monitored event.
SCR Requirements Model

• Applying the definition in the model to the condition table in Table 3.

\[
\text{SafetyInjection} = \begin{cases} 
\text{Off} & \text{if } \text{Pressure} = \text{High} \lor \text{Pressure} = \text{Permitted} \lor \\
 & \quad (\text{Pressure} = \text{TooLow} \land \text{Overridden} = \text{true}) \\
\text{On} & \text{if } \text{Pressure} = \text{TooLow} \land \text{Overridden} = \text{false}. 
\end{cases}
\]
SCR Requirements Model

• To the event table in Table 2.

\[
\text{Overridden}' = \begin{cases} 
  \text{true} & \text{if} \\
  \text{false} & \text{if} \\
  \text{Overridden} & \text{otherwise} \\
\end{cases} 
\]

\[
\text{true} \text{ if } \begin{align*}
(Pressure = \text{TooLow} \land Block' = \text{On} \land \\
Block = \text{Off} \land Reset = \text{Off}) \lor \\
(Pressure = \text{Permitted} \land Block' = \text{On} \land \\
Block = \text{Off} \land Reset = \text{Off})
\end{align*}
\]

\[
\text{false} \text{ if } \begin{align*}
(Pressure = \text{TooLow} \land Reset' = \text{On} \land \\
Reset = \text{Off}) \lor \\
(Pressure = \text{Permitted} \land Reset' = \text{On} \land \\
Reset = \text{Off}) \lor \\
(Pressure' = \text{High} \land Pressure \neq \text{High})
\end{align*}
\]
SCR Requirement Model

• To define the required behavior completely and unambiguously, satisfy certain properties.
  – Disjointness Property
    • The pairwise conjunction of conditions (events) in each row of a condition (an event) table must always be false. => intersection is false.
  – Coverage Property
    • the disjunction of the conditions in each row of the table must be true. => union is true.
SCR Tool

• SCR Tools support SCR Process
  – First, a requirements specification is constructed using the SCR tabular notation.
  – Second, the specification is analyzed for violations of application independent properties, such as missing cases and unwanted ambiguity.
  – Third, the specification is validated by application experts to ensure that it captures the intended behavior.
  – Finally, the specification is analyzed for critical application properties, such as security and safety properties.
SCR Tool

- **Specification editor**
  - Construct the SCR Requirements specification

- **Consistency Checker**
  - analyze the specification for properties derived from the SCR requirements model.
  - exposes syntax and type errors, variable name discrepancies, missing cases, ambiguity, and circular definitions.
SCR Tool

• SCR Simulator
  – Run scenarios, sequences of monitored events.
  – the simulator supports the rapid construction of graphical user interfaces, customized for particular application domains.

• Model Checker
  – SPIN : analyze a finite state model of the specification.
  – TAME : theorem prover.
  – PVS : specialized interface to the general-purpose theorem prover.
  – Salsa : automatic theorem prover based on decision procedures.
Applying to practical System

- International Space Station, NASA
- Flight guidance system (FGS), Rockwell Aviation
- U.S. weapons system, NRL
- cryptographic device, NRL
SCR Approach to Software Design

• Module Structure
• Module Interface
• Uses Hierarchy
SCR Approach to Software Design

• Module Structure
  – each module is either a collection of submodules or a single work assignment, that is, a programming task that can be completed by a single programmer.
  – applying the information hiding principle to module decomposition.
  – the module guide which has a tree structure, describes the responsibility of each module by stating the design decisions that will be encapsulated by the module.
SCR Approach to Software Design

• Module Interface
  – the set of assumptions that the programmers responsible for other modules
  – the set of access programs that programs in other modules use to access the data or services provided by the module
SCR Approach to Software Design

• Uses Hierarchy
  – relation defined on the access programs in the module interface specifications.
  – Suppose A and B are access programs. Then, we say that A uses B if and only if the correctness of program A depends on the presence of a correct program B.
  – loop-free graph has advantage that it defines a number of usable subsets of the complete system.