Design Verification

Outline

- Review of PSP Levels
- Overview
- Selecting Verification Methods
- Design Standards
  - Verification Methods
    - Approaches
    - State Machines
    - Program Tracing
    - Program Correctness
  - Etc.

Review of PSP Levels

(Humphrey, 1995, p. 11)

PSP0
- Current process
- Time recording
- Defect recording
- Defect type standard

PSP1
- Size estimating
- Test report

PSP2
- Code reviews
- Design reviews

PSP3
- Cyclic development

PSP2.1
- Design templates

PSP1.1
- Task planning
- Schedule planning

PSP0.1
- Coding standard
- Size measurement
- Process improvement proposal (PIP)
- Baseline

Planning

Quality Mgt

Cyclic

Overview

(cf. Humphrey, 1995, p. 373-374)

To build high-quality software you must ensure that your designs are correct.
Thus, the question is not whether, but how, to verify your programs.

- These approaches are not foolproof.
- They are prone to human error.
- However, their structure facilitates accuracy and reliability.

This chapter discusses a number of methods for doing this.

- Formal methods can sometimes be used.
- However, this book presents “semi-formal” methods.

Selecting Verification Methods

(cf. Humphrey, 1995, p. 374-376)

- Select appropriate methods based on:
  - Your defect profile: Use verification where you have problems.
  - Effectiveness of your current methods: Use methods you know and are effective with.
  - Economics of your methods: Use the most cost-effective methods.

<table>
<thead>
<tr>
<th>Verification Methods</th>
<th>Application</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loop Verification</td>
<td>Program loops</td>
<td>Use on loop logic whenever practical.</td>
</tr>
<tr>
<td>Proper State Machines</td>
<td>State Machines</td>
<td>Use during design and in reviews and inspections on every state machine.</td>
</tr>
<tr>
<td>Symbolic Execution</td>
<td>Algorithmic Logic</td>
<td>Use whenever it applies.</td>
</tr>
<tr>
<td>Proof by Induction</td>
<td>Loops &amp; Recursion</td>
<td>Use in conjunction with trace tables.</td>
</tr>
<tr>
<td>Trace Tables</td>
<td>Simple Logic</td>
<td>Use for small program elements and with proof by induction and/or symbolic execution whenever possible. Use if other verification methods do not apply.</td>
</tr>
<tr>
<td>Execution Tables</td>
<td>Complex Logic</td>
<td>Use for small program elements and, as a last resort, when no other methods apply.</td>
</tr>
<tr>
<td>Formal Verification</td>
<td>Entire Program</td>
<td>Use whenever you know how to apply the verification methods, they appear feasible, and they are cost effective.</td>
</tr>
</tbody>
</table>

Verification Methods: Design Standards

(cf. Humphrey, 1995, p. 376-378)

- Design standards do not seem like a verification method.
- However, they provide criteria against which to evaluate a design.
- Some standards you should use are:
  - Product conventions
    - “Conceptual integrity”
  - Product design standards
    - Coding & naming conventions
    - Header, test, and documentation standards & formats, etc.
    - May be arbitrary, but you need a standard
  - Reuse standards
    - Components must be well-documented, available, meet needs, and be reliable
    - IBM’s German lab’s “C5 components catalog” parts have never received a user defect report
    - Toshiba’s control system, which achieved 90% reuse, had the “lowest defect content of any software [that users] had ever seen.”
Verification Methods:
Symbolic Execution

- In symbolic execution, the approach is to:
  - assign algebraic symbols to the program variables
  - restate the program as one or more equations in these symbols
  - analyze the behavior of these equations

- Some questions to ask are:
  - does the program converge on a result?
  - how does the program behave for both normal and abnormal input values?
  - does the program always produce the desired results?


Verification Methods:
Proof by Induction
(cf. Humphrey, 1995, p. 379-380, and lecture notes)

- Proof by induction states that:
  1. if f(n) is true for n = k
     2. and if
        • when n = z where z > k
        • and f(z) is true
        • you can show that f(z+1) is true
     3. then
        • f(n) is true for all values of n larger than k

- Look for places where there would be problems at z+1
  (logical or hardware limits, memory, etc.)

- cf. Example, p. 380 (Function call)

Verification Methods:
State Machines
(cf. Humphrey, 1995, p. 380-397)

- A program is likely a state machine if, with identical inputs, it behaves differently at different times.

- Example: LOC counter
  - comments
  - non-comments (program, executable)

- In a proper state machine:
  - it is possible to reach a program return state from every other state
  - all state conditions are complete and orthogonal
  - all transitions from each state are complete and orthogonal

Verification Methods:
Program Tracing
(cf. Humphrey, 1995, p. 397)

- Program tracing is performed with two general methods:
  - Execution Tables
  - Trace Tables

Rules for Checking for a Proper State Machine
(cf. Humphrey, 1995, p. 392)

- Check for hidden traps or loops.
  - It cannot get stuck in an endless loop and never reach a return state.

- See if all possible states have been identified.
  - A state is defined for every possible combinations of attributes.

- Check for state orthogonality.
  - For every set of conditions there is one and only one possible state.

- Check for transition completeness and orthogonality.
  - From every state, a unique next state is defined for every possible combination of state machine input values.

Two Examples of Checking State Machines
(cf. Humphrey, 1995, p. 393-397)

- BSet
  - cf. Fig 12.1 (state machine) and Table 12.3 (state specification), p. 382, 383
  - Do checks

- CData
  - cf. Fig 12.2 (state machine) and Table 12.5 (state specification), p. 385, 387-389
  - Do checks
Verification Methods:

**Execution Tables**
(cf. Humphrey, 1995, p. 397-405, and lecture notes)

- An execution table is an orderly way to trace program execution.
  - It is a manual check of the program flow
  - It starts with initial conditions
  - A set of variable values is selected
  - Each execution step is examined
  - Every change in variable values is entered
  - Program behavior is checked against the specification

- The advantages of execution tables are
  - They are simple
  - They give reliable proofs

- The disadvantages of execution tables are
  - They only check one case at a time
  - They are time consuming
  - They are subject to human error

An Execution Table Example
(cf. Humphrey, 1995, p. 397-405, and lecture notes)

- To use an execution table
  - Identify the key program variables and enter them at the top of the trace table
  - Enter the principal program steps
  - Determine and enter the initial conditions
  - Trace the variable values through each program step
  - For repeating loops, add additional execution table steps for each additional loop cycle
  - For long loops, group intermediate steps if their results are obvious

- Cf. ClearSpaces Example, Table 12.9, Fig 12.3, etc., p. 396-405

Verification Methods:

**Trace Tables**
(cf. Humphrey, 1995, p. 400-418, and lecture notes)

- Trace tables are similar to execution tables, but more general.
- Trace tables examine general program behavior rather than verifying individual cases.
- Trace tables use
  - Symbolic execution
  - Case checking

Example Trace Tables
(cf. Humphrey, 1995, p. 400-418, and lecture notes)

- Walk through examples from book and from lecture notes

Verification Methods:

**Program Correctness**
(cf. Humphrey, 1995, p. 418-436, and lecture notes)

- Formal mathematical proof techniques exist and are good to use when possible.
- However, we cover less formal approaches, but borrow some ideas from the formal methods.
- We apply these approaches to the testing of loops:
  - For-loop verification
  - While-loop verification
  - Repeat-until (do-while) verification
- Check:
  - Preconditions
  - Appropriate test cases
  - Loop termination conditions
  - FirstPart, SecondPart, ...

Comments on Verification Methods
(cf. Humphrey, 1995, p. 436-437)

- If you have any question about the validity of the design, perform verification.
- Test at least a single case, even when confident of the design.
- Design down, verify up.
- Verify all cases.
- Track time spent in verification and assess cost-effectiveness of approaches after you become familiar with the techniques.
- “When you verify your designs as you produce them, your design verification data can greatly accelerate your design reviews.”