TECHNIQUES AND LESSONS FROM SOFTWARE VERIFICATION

Gerard Holzmann
Jet Propulsion Laboratory, California Institute of Technology

Software Verification Methods we used on MSL

- Redefined Code Review Process, based the Scrub tool
- New JPL-wide Coding Standard for Mission Critical Software
- Required Certification Courses for Software Developers
- Integration of Static Source Code Analysis with every build and release & with code reviews (Covellite, CodeSonar, Soname, Unit)
- Thorough Logic Verification of various MSL software subsystems with the Spin Model Checker
  - Example 1: data management subsystem (MRF)
    - Outcome: complete redesign
  - Example 2: instrument library manager (IML)
    - Outcome: revision to prevent race conditions

© 2012 California Institute of Technology, Pasadena, CA
THE JPL CODING STANDARD FOR C

http://fsm-lab.jpl.nasa.gov

LEVELS OF COMPLIANCE

- LOC-1: language compliance
- LOC-2: predictable execution
- LOC-3: defensive coding
- LOC-4: code clarity
- LOC-5: MISRA shall compliance
- LOC-6: MISRA should compliance

© 2012 California Institute of Technology, Pasadena, CA

THE POWER OF 10 RULES

1. Restrict to simple control flow constructs
2. Do not use recursion and give all loops a fixed upper-bound
3. Do not use dynamic memory allocation after initialization
4. Limit functions to no more than ~60 lines of text
5. Use minimally two assertions per function on average
6. Declare data objects at the smallest possible level of scope
7. Check the return value of non-void functions; check the validity of parameters
8. Limit the use of the preprocessor to file inclusion and simple macros
9. Limit the use of pointers. Use no more than N level of dereferencing
10. Compile with all warnings enabled, and use source code analyzers

http://spinroot.com/p/10/

© 2012 California Institute of Technology, Pasadena, CA
GENERATION OF TOOL REPORTS

Nightly Build -> make -> Build Log (~18K lines) -> extract compiler calls (~3K lines)

Defect Detection
- gcc
- gcc strict
- coverity
- codeonar
- uno

Coding Rule Compliance
- P10
- JPL standard
- MSL rules

triaging

SCRUB database

~80% OF ALL REPORTS LED TO A CODE FIX
(BOTH FOR PEER AND TOOL GENERATED REPORTS)
TESTING

- in test-based verification one often treats all code alike
  - but it is useful to distinguish:
    - deterministic (e.g., math) routines and
    - non-deterministic (e.g., reactive) code

<table>
<thead>
<tr>
<th>Current Method</th>
<th>Better</th>
<th>Best</th>
</tr>
</thead>
<tbody>
<tr>
<td>math routines (deterministic)</td>
<td>sampling-based testing</td>
<td>randomized (fuzz) testing + static analysis</td>
</tr>
<tr>
<td>reactive code (concurrent)</td>
<td>?</td>
<td>logic model checking</td>
</tr>
</tbody>
</table>

these methods are very useful although none are “logically complete” and some are not “logically sound”

---

Swarm verification
proof of concept example

- call processing kernel in C
- Lucent PathStar® Telephone Switching system
- code
- 200 feature requirements formalized in logic (LTL)
- mechanically extracted Spin verification model
- 32-core desktop system with 64 GB of memory running Ubuntu Linux
- PC
- error traces
- <10 sec
- bugs

Method: the system schedules a swarm of small, randomly different, verification runs using sampling. Sampling is iteratively broadened until a bug is found. Coverage reaches completeness, or time runs out.

time to find error traces using swarm verification:
- 11 in 1 second
- 38 in 7 seconds

© 2012 California Institute of Technology, Pasadena, CA