Agile Verification

Gerard J. Holzmann
gh@jpl.nasa.gov
how we design & test software today
(a simplification)

require ➔ design ➔ code ➔ test ➔ fly

human-readable artifacts
machine-readable artifacts

the main quality “gates”
gaining in strength and precision as we move to the left
the standard formal methods pitch

- model-based design techniques can introduce machine readable artifacts earlier in the life-cycle – which enable more thorough tool-based analysis techniques for requirements & design
  - for instance, logic model checkers can then be used to verify requirements against high-level design models
another look at testing

• in test-based verification, we tend to treat all code alike
  – but there’s a difference between:
    • 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 but none are “logically complete” and some are not “logically sound”

needs to be fast, automatic, and scalable
example of an agile verification process

- call processing kernel in C
- 200 feature requirements formalized in logic
- mechanically extracted verification model
- 32-core desktop system with 64 GB of memory running Ubuntu Linux

How does this work?
for each property – the system schedules large swarms of small, randomly different, verification runs using sampling. sampling is iteratively broadened until a bug is found, coverage reaches completeness, or time runs out

code
<10 sec bugs

11 in 1 second
38 in 7 seconds
70 in 600 seconds

Lucent Technologies Pathstar® access server

error traces

32 - core

desktop system with
64 GB of memory
running Ubuntu Linux

Lucent Technologies Pathstar® access server

time to find error traces:
**Agile Verification** techniques can fill a gap we have in developing reliable mission-critical software

With the large multi-core systems that are now generally available, this approach has become technically feasible

<table>
<thead>
<tr>
<th></th>
<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>
<td>Pre- &amp; Post-conditions, loop invariants, theorem proving</td>
</tr>
<tr>
<td>reactive code (concurrent)</td>
<td></td>
<td>?</td>
<td>logic model checking</td>
</tr>
</tbody>
</table>