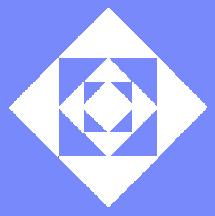


ConTest Intermittent Bugs Removal Process

Eitan Farchi

December 22, 2004

http://www.haifa.il.ibm.com/projects/verification/contest/index.html

Our Real Life Motivation

- **Project:** A file system project
- **Stage:** Component test
- **Observed behavior:** "Use-count" values corrupted during stress test
- **Time wasted:** Several calendar months, more than 0.5 PY
- **Main difficulty:** Failure was not repeatable
- **Fault description:** Incorrect use of compare-and-swap result
- **Bug should have been found during the unit test!**

Testing Concurrent and Distributed Applications
Why is Concurrent Testing Hard?

- Concurrency introduces non-determinism
  - Multiple executions of the same test may have different interleaving (and different results!)
    - an interleaving is the relative execution order of the program threads
- Re-executing a test on a single stand-alone processor is not useful
- Debugging affects the timing
- No useful coverage measures for the interleaving space
- Result: Most bugs are found in system tests, stress tests, or by the customer

Stages

- Review
  - A methodology
  - We teach a tutorial
  - Hands-on on two inspection methodologies
- Unit test
  - A tool and methodology
  - We teach how to do it
  - We supply the tools with which it is done
- Function and system test enhancements
  - More tool then methodology
  - Increases testing effectiveness
  - Helps in coverage, debugging, replay

General PR for Reviews

- I really shouldn't need to be saying it but
  - Great ROI
  - Finding bugs early
  - Save time and reduce cost
  - Help individual learning
  - Build effective teams
  - Help retentions
  - Education
  - Tester programmer communication
  - ….
- So is it done?
  - Expensive, boring, author protection, "no time"…

Traditional Review is for Sequential Code

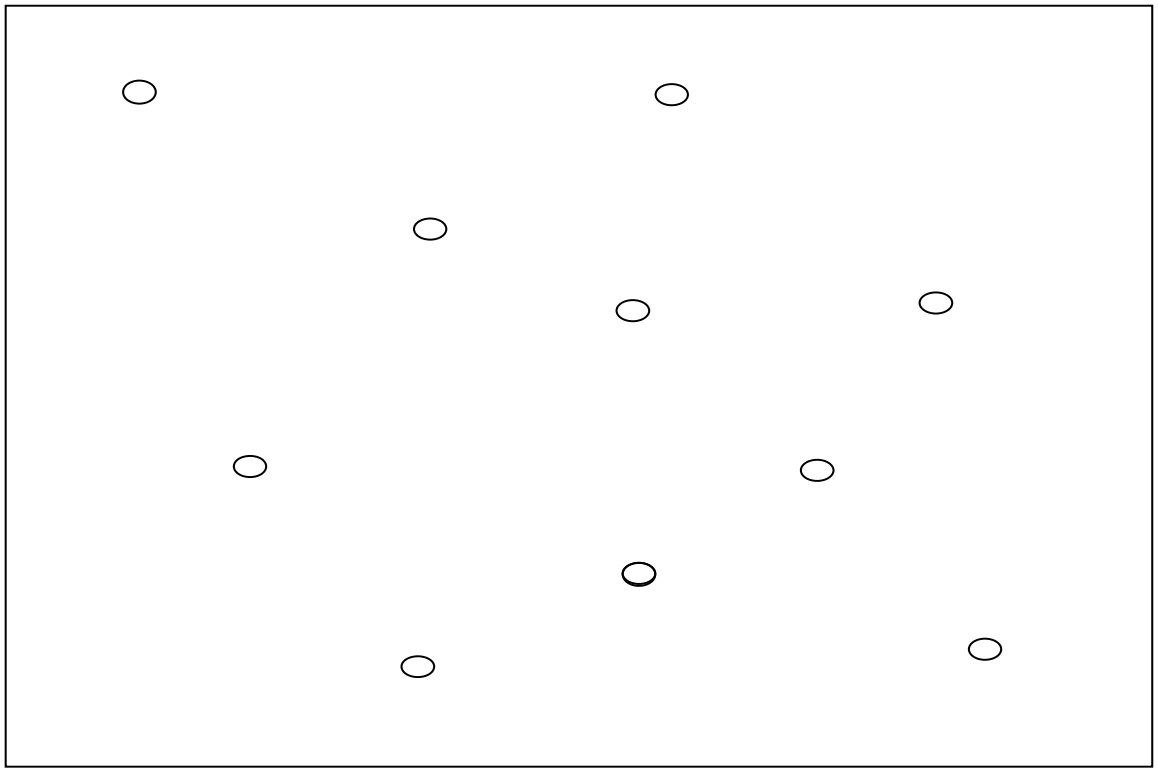
- Formal review is mainly a homework assignment
  - The assumption is that it can be done without the owner
- Walkthrough depends only on inputs not on interleaving
- Concurrent bugs tend to be distributed ☹
  - Looking at a small piece of code may miss them
  - Protocols has to be looked at as a whole
- Specific bug patterns are to be looked for

Review Methodology

- **Overall system review**
  - Automatic identification of relevant classes (files)
- **Synchronization protocol review**
  - Concurrent bug pattern based review
    - Using a check list
  - **Interleaving Review Technique - IRT**
    - **Review interleavings**
    - **Obtain test selection as a side effects**

Overall System Review

- A design problem:



Bug Pattern Example: Lost-Notify

- Losing notify: the notify is “lost” because it occurs before the thread executes the wait() primitive
  - The gap was created because the programmer didn’t think the notify would occur before the wait

Thread 1

```
Synchronized (o){
   o.wait();
}
```

Thread 2

```
synchronized (o){
    o.notifyAll();
}
```

Interleaving Review Technique

- The use of the Cartesian product technique to select interleavings and states to review
- Definition of review roles and guideline to carrying out the roles
  - Program counter – needs to thoroughly understand the system so he can determine the control flow
  - Devil's advocate – experienced in concurrent and fault tolerance systems. His role is to make choices as to the timing of events and failures
    - To maximize the probability that a bug is found
    - IRT provides guidelines for making these choices
  - Stenographer – experienced in representation techniques (use cases, sequential diagram, time diagrams, etc) and able to strike a trade-off between accuracy and readability

Adoption

- Tried successfully by projects in IBM
  - Found additional problems in reviewed code each time used
  - Small extra effort
  - “We believe that this technique should be used wherever multi threaded code is being developed. It significantly reduces the number of concurrency related bugs, and promotes quality.”
- Adopted for use in all new code developed
  - Developers see the benefit
  - Learning curve is fast
- On average a bug per person-hour is found!!!

Unit Test

- **Test execution with ConTest**
  - **repeat until 100% synchronization coverage is obtained**
    - **Class abstraction and wrapping**
    - **Use test selection of the IRT stage and the multiplication technique**
    - **Run many times**
- **You can remove most of these bugs in unit test!**
  - **Currently they are found mostly by the customer…**

Impact on Function and System Test

- Negligible change to process
  - Installed in big IBM projects in a day
- Increase efficiency in finding bugs
- Additional benefits
  - Coverage measurements (the tool of choice in Java)
  - Aids in debugging
  - Replay feature

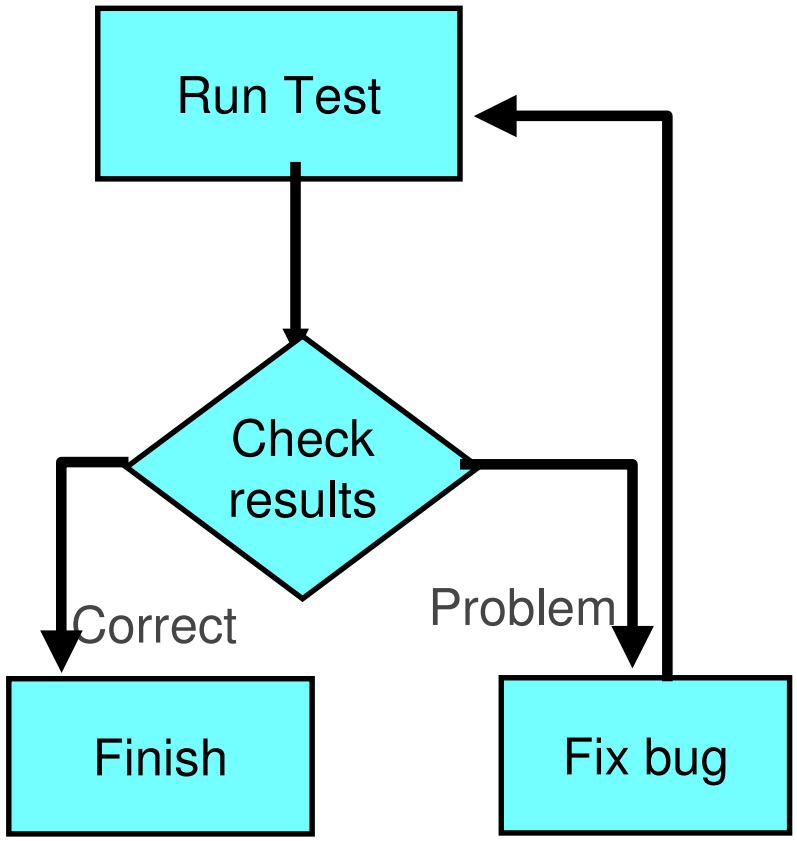
How Does ConTest Find Bugs?

- We instrument every concurrent event
  - Concurrent events are the events whose order determines the result of the program
- At every concurrent event, a random based decision is made whether to cause a context switch
  - For example, using a sleep statement
- Philosophy:
  - Modify the program in such a way that it will be more likely to exhibit bugs (without introducing new bugs)
  - Minimize impact on the testing process (under the hood technology)
  - Re-use existing tests
  - Utilize idle computer time

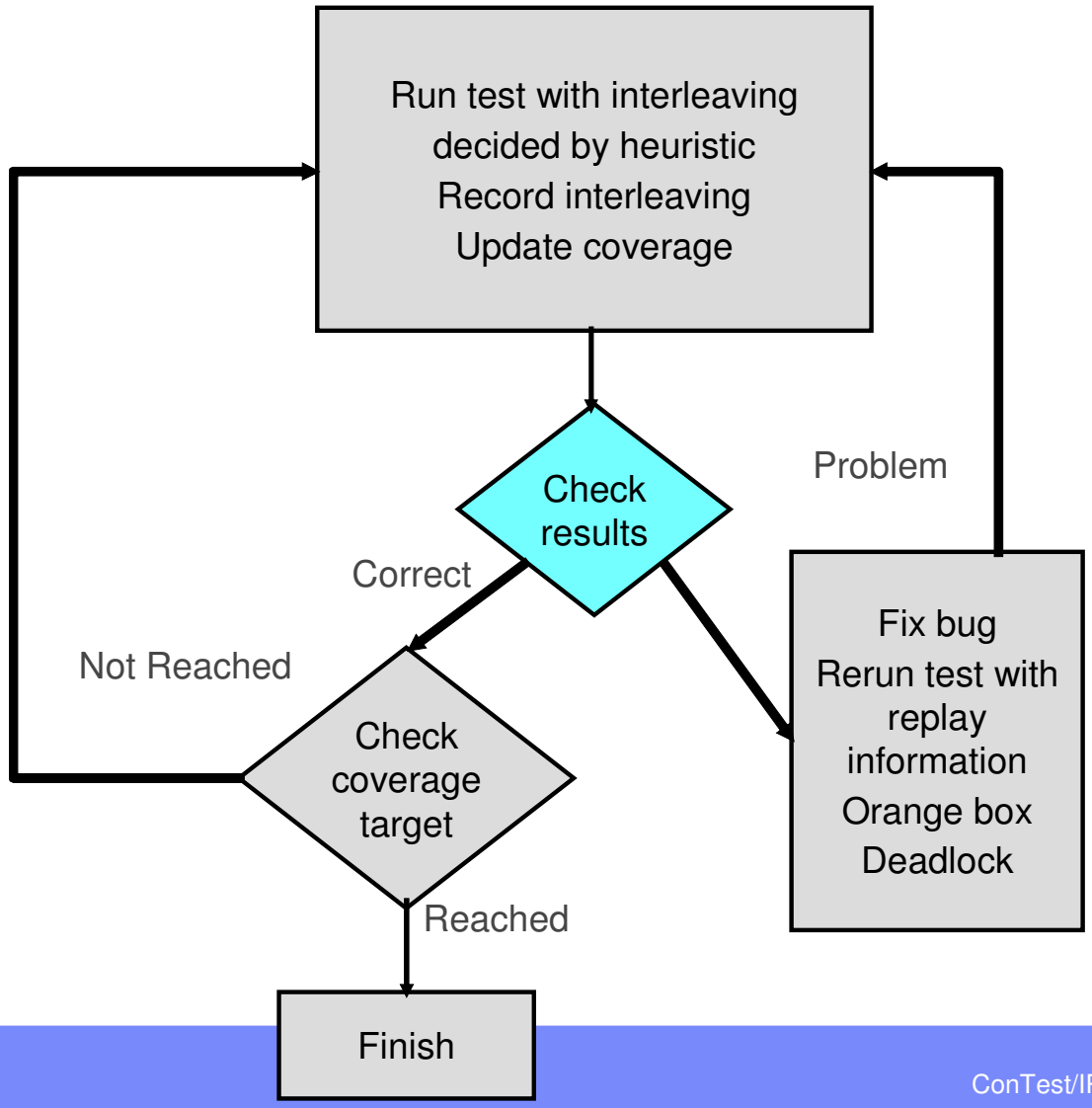
ConTest Overview

- ConTest is composed of the following components
  - An instrumentation engine that
    - Creates hooks for the irritator and for coverage printing
    - Generates coverage models
    - Instrumentation is done at the bytecode level
  - An irritator that randomly, or using heuristics, generates new interleaving on-the-fly
  - Replay component
  - Coverage component
  - Debugging aids

User Scenario Without ConTest



User Scenario With ConTest



ConTest - Status

- Status
  - Widely used internal IBM tool
  - Prototype for C/C++
  - Fault injection to simulate network traffic
  - Used both by developers and testers
  - Based on a extensive research (14 papers, 6 patents)
  - Eclipse plug-in available

ConTest/IRT Benefits

- IRT
  - Design and code review method for concurrency related defects
  - Helps in test plan design
  - Impacts RAS and general product quality
- ConTest
  - Improves testing concurrent and distributed applications for timing related bugs from *early* development stages
  - Minimum impact on the testing process (under the hood technology)
  - Re-use existing tests
- IRT in conjunction with ConTest is a complete light-weight concurrent development methodology