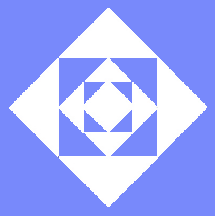

Interleaving Review Technique

Shachar Fienblit

December 18, 2003

Outline

- Motivation
- The Interleaving Review Technique
- Adaptation
- Method Details
- Example
- Guidelines for the Devil's Advocate
- Summary

Motivation – What Is The Problem?

- Concurrency and fault tolerance related problems are hard to find
  - Costly (usually found late in system testing)
  - Difficult to recreate and debug
    - Complex scenarios
    - Error and exception paths
  - Unfortunately often found at customers' site
- Finding problems as early as possible
  - During design and coding
  - In addition to the traditional reviews and testing tools
- Need a method for wide audience
  - Lightweight
  - Language and platform independent
  - Cost effective immediately

The Interleaving Review Technique

- Derivation of the walkthrough technique
  - Design review and code review oriented at
    - Concurrency
    - Fault tolerance
  - In addition to traditional review
  - Helps in test plan design
- Provides a lightweight concurrency oriented code verification technique
- In addition to conTest which provides a testing environment for concurrency problems

Adaptation

- Tried successfully by middleware projects in IBM
  - Found additional problems in already reviewed code each time used
  - Small extra effort
- Adopted for use in all new code developed
  - Developers see the benefit
  - Learning curve is fast
- Some statistics gathered

IRT Statistics from experiments with already reviewed code

Project type	Overall time spent (hours for all developers)	Number of bugs found	Comments
Cluster device drivers	2	2	Two developers
Cluster device drivers	10	15	Five developers. Was done after six hours of regular reviews
Cluster device drivers	12	4	Two developers
Cluster device drivers	1	1	Two developers
Cluster device drivers	2	1	Two developers
Cluster device drivers	3	2	Three developers
Group communication	10	1 design, 2 bugs, 17 code modifications	Two developers
Group communication	5	2 bugs, 4 code modifications	Two developers

Desk checking

- An extremely effective code review technique used for early detection of sequential program errors
  - Desk checking means manual execution of the program and writing the tests first
  - The system behavior is reviewed
- Introduced in 1974 by C.A.R. Hoare in his structured programming course
  - “The first principle of error detection is that the sooner an error is detected the less trouble it will cause”
  - Back then most written programs were sequential

A toy example of desk checking

- Program definition – sum up the positive integers that are smaller than j
- Program segment

```
int sum = -1;
int j = 0;
read(j);  //The user inputs j
for(int i = 0;i < j; i++)
   sum = sum + i;
```

- Tracing of test data j = 1
- Bug found – sum = -1 at the end

control	sum	I	j
	-1		0
read(j)			
	0		1
i = 0			
	-1	0	1
(i < j)	is true		
sum =	sum + i		
	-1	0	1
i++			
	-1	1	1
i < j	Is false		

The problems IRT (Interleaving Review Technique) addresses

- When attempting to review or test the system behavior of a concurrent/distributed and fault tolerant system several problems arise
  - Non-determinism
    - Given that the program is in some state, the next program state is depended on which process executes next
      - As a result it is not clear how to proceed with the review process
  - The space of possible program schedules, sometimes called the space of possible interleavings is exponential
  - It is hard to recreate failures

The problems IRT addresses (continued)

- The state of the concurrent or distributed program is determined by the state of all its processes and their interrelated temporal dependencies
  - 3 processes with 10 states have 1000 possible states to review
- Tests are much more expensive
  - Require the interaction of many machines and failures in predetermined sequences

IRT Consists of

- The use of the Cartesian product technique to select interleavings and states to review (FoCus)
- 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

IRT Consists of (Continued)

- The same scenarios are reviewed in more and more details as the development cycle progresses and
  - Are finally used as a base to preparing the test plan
- Functional coverage is used to determine if the test plan was carried out
- Simulator is used to facilitate the review
  - Supporting tools should be developed to support execution through user interaction when only part of the system state is known
- Note: the review is beneficial in the absence of a simulator

Why let a different process advance after a lock is obtained?

- The devil's advocate decides that another process/thread advances right after or before a synchronization operation is performed
  - He also make sure that locks are waited on
- Most of the synchronization primitives require that all processes accessing the shared resource follow the protocol
  - Thus, obtaining the lock does not guarantee protection if other processes are not attempting to obtain the same lock
- This choice method significantly decreases the number of interleaving to consider for review

A toy IRT example – who is the king?

- Code segment executed by several processes with the objective of choosing a leader processor

```
boolean chosen= false;  // global variable used for process coordination
boolean ImAKing = false; // local – indicates the current process status
if (chosen == false) {
   lock();
      chosen = true;
      ImAKing = true;
   unlock();
}
```

An IRT example – who is the king?

Time

Process one	Process two	Chosen	Process one ImAKing	Process two ImAKing
Program counter - start executing		false	false	false
If(chosen == false) Is true		false	false	false
Devil advocate – advance second				
	If(chosen == false) Is true	false	false	false
	lock()	false	false	false
	chosen = true;	true	false	false

An IRT example – who is the king?

Time

Process one	Process two	Chosen	Process one ImAKing	Process two ImAKing
	ImAKing = true	true	false	true
	unlock()			
Devil's advocate – advance first				
lock()		true	false	true
chosen = true;		true	false	true
ImAKing = true		true	true	true
unlock()		true	true	true

Some Guidelines for the Devil's Advocate

- Increase contention on shared resources
- Delay locks so that locks are obtained in different orders
- While in critical section
  - Force error paths, assume that potentially blocked operations are blocked and cause signals and interrupts to occur
- Cover all possible scenarios of waiting on event
  - Event notification is sent
    - Before and after the event is waited on
  - If waiting on event is not atomic - event notification is sent after the event is checked and before it is waited on
- Break assumption that depend on hardware and scheduler
  - Assume that delays are not long enough
  - Assume that changes are not visible due to the memory model
- Based on concurrent bug pattern paper (PADTAD2003)

Summary

- Effective lightweight method to increase quality
  - Cost effective
  - Benefits are evident from first proper use
- Support “quality culture”
  - Quality by design
- Tried successfully by two significant products