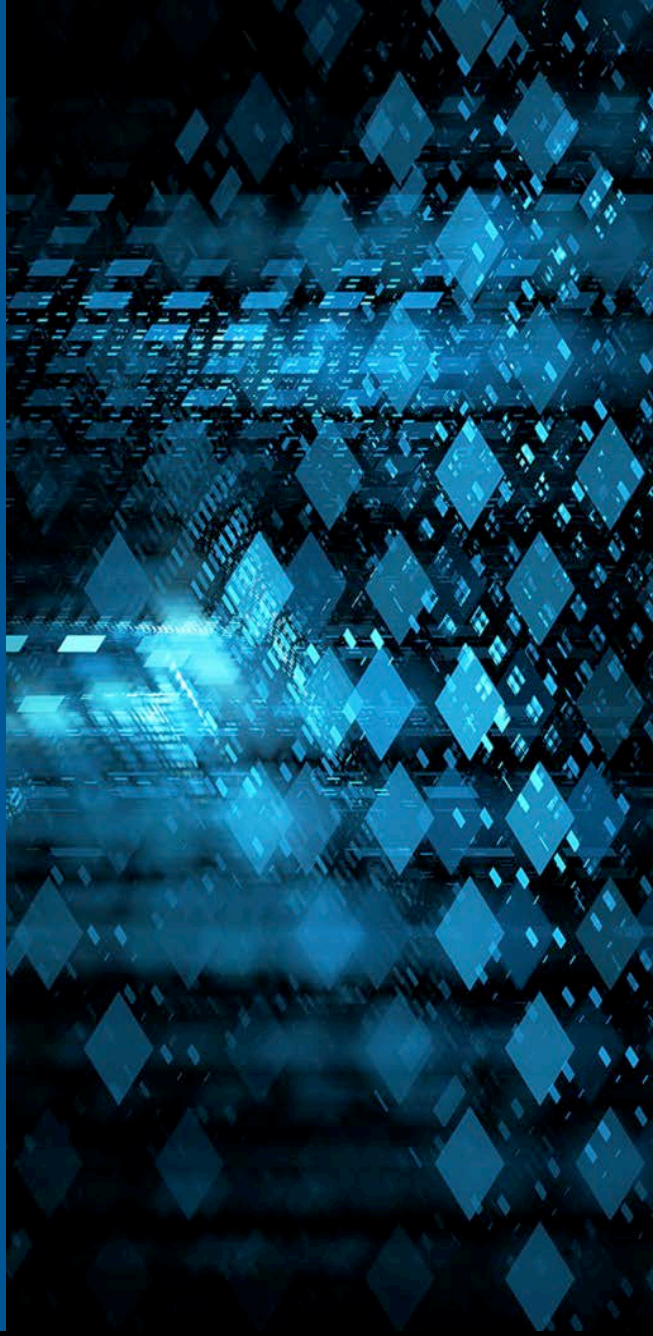
Testing in a Non-Deterministic World

Donald Firesmith

Robert V. Binder

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213



This material is based upon work funded and supported by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

DM-0003917

Topics

Testing

Determinism Assumptions

Trends

Non-Deterministic World

Testing Ramifications

Recommendations

Testing in a Non-Deterministic World

Testing

Testing – 1

Testing compares a system's actual behavior with its expected behavior so that discrepancies (bugs) can be analyzed to uncover associated defects and thereby determine quality.

Testing involves:

- Establishing known test preconditions
- Providing known test inputs
- Comparing actual with expected test outputs
- Comparing actual with expected test postconditions

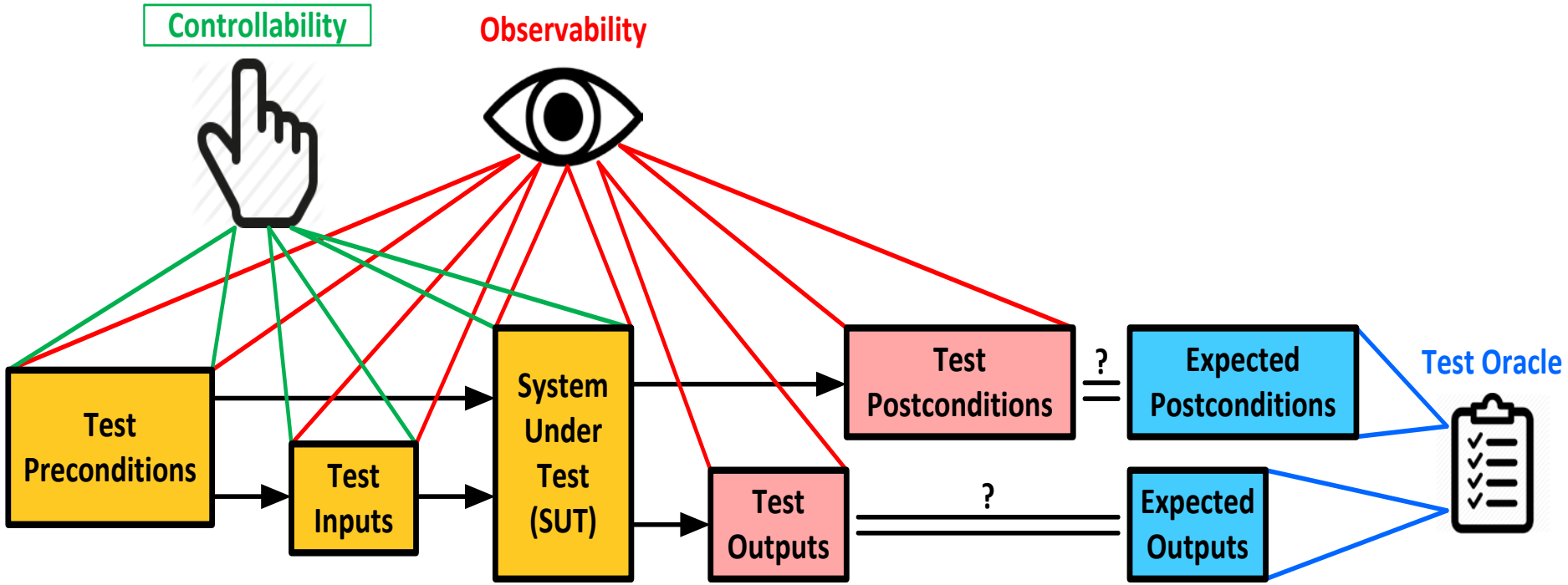
Discrepancies can be:

- Visible *failures* (incorrect visible behavior)
- Hidden *faults* (incorrect encapsulated mode, state, or data)

Testing – 2

Testing requires:

- *Controllability* to establish preconditions and create inputs
- *Observability* to verify correctness
- *Oracle* to provide expected behavior (e.g., requirements, design)



Testing in a Non-Deterministic World

Determinism Assumptions

Determinism Assumptions

Many developers assume:

- Preconditions and inputs can be controlled.
- System behavior (outputs and postconditions) are deterministic.
- Oracle provides single outcome (outputs and postconditions) given same preconditions and inputs.
- Tests are therefore repeatable
  (i.e., The same test will always yield the same result.)

These assumptions are not always true, resulting in bugs that are:

- Rare
- Intermittent
- Difficult to reproduce, localize, and diagnose

Recent technology trends increase the likelihood that these assumptions are false.

Testing in a Non-Deterministic World

Trends

Trends

Agile development relies on continuous integration achieved via repeatable regression testing. This is only feasible with automated testing, which assumes deterministic behavior.

Use of multicore processors and virtual machines increases concurrent processing and associated concurrency defects.

Increasing use of systems of systems increases concurrent system behaviors and concurrent system-system communications.

Increasing reliance on cyber-physical systems (including autonomous vehicles) increases non-deterministic environmental inputs (e.g., from sensors) and preconditions.

Testing in a Non-Deterministic World

Non-Deterministic World

Types of Non-determinism

Reality:

- Actual vs. apparent

Source:

- Natural (inherent in the nature of reality)
- Emergent behavior
- Concurrent (due to concurrency)
- Exceptional (fault and failure behavior)

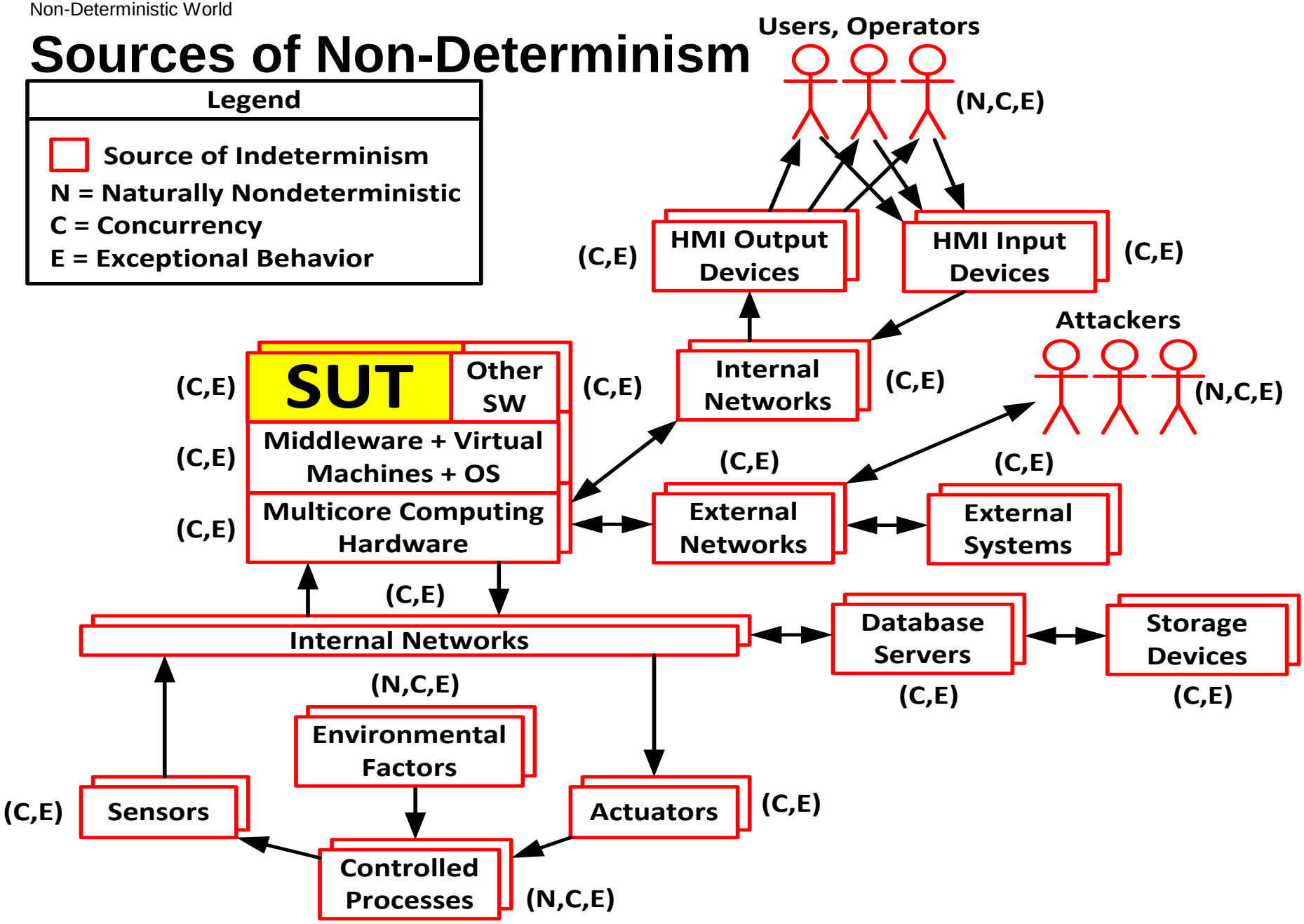
Location:

- Nondeterministic preconditions
- Nondeterministic inputs
- Nondeterministic system responses

Cause:

- Lack of controllability
- Lack of visibility (hidden variables)

Sources of Non-Determinism



Examples of Nondeterministic Systems

Cyber-physical systems:

- Process control (chemicals, petrochemicals, medicines)
- Vehicle control (manned and autonomous)
- Internet of Things (IoT)
- Power generation and distribution
- Systems of Systems (SoS), federated systems

Mobile computing

Cloud computing

Computing involving:

- Multicore processors
- Multiple processors (or computers or devices)
- Virtual Machines
- Multithreading and concurrent programming languages

Non-Deterministic Environments

Environments may not be Deterministic:

- Development Environments
- Developmental Test Environments
- Operational Test Environments
- Physical Operational Environment

Non-Deterministic Defects – 1

Developers and testers fail to adequately take the following into account when designing software and associated test suites:

Concurrency Defects:

- Classic Defects:
  Deadlock, livelock, starvation, suspension, priority inversion, (data) race conditions, order violations, atomicity violations, lock and semaphore defects
- Multicore Defects:
  Interference between cores due to sharing common resources (e.g., L2/L3 caches, RAM and disc memory, I/O, system bus), improper allocation of SW to HW, unacceptable jitter in processing times
- Virtual Machine Defects:
  VM escapes and interference between VMs, improper allocation of SW to Virtual Machines, hypervisor defects

Non-Deterministic Defects – 2

Performance Bugs:

- Missed deadlines (latency and response time)
- Unacceptable jitter
- Incorrect order

Reliability Defects:

- Buffer overflow, automatic garbage collection

Robustness Defects:

- Missing, inadequate, or incorrect error, fault, failure, and environmental tolerance

Security Defects:

- Vulnerabilities
- Defects in security controls (e.g., incorrect configurations)

Non-Deterministic Defects – 3

Rare alignments of cyclic behaviors that result in intermittent and non-reproducible failures

Positive feedback under stress propagated through a system's environment triggering showstoppers or dangerous loss of control

Bizarre responses that are internally consistent but externally dangerous

Related Issues – 1

Testing autonomous systems:

- Verify reasoning process
- Verify models match reality

Testing AI systems:

- Verify learning and adaptation

Fuzzy success criteria (boundary of correct behavior space):

- Pass, fail, partially pass/fail, unclear
- Stochastic pass/fail

Lack of oracle:

- Lack of verifiable requirements
- Excessively complex oracle

Related Issues – 2

False positive and false negative test results

Hard vs. software real-time

Black swan events:

- Definition:
  - Rare (outlier),
  - Impact (big & negative)
  - Explainable (after the fact)
- Types:
  - Failures and faults
  - Accidents and near-misses

Testing in a Non-Deterministic World

Testing Ramifications

Testing Ramifications

Many developers and testers are not adequately familiar with non-deterministic defects.

- Many developers and testers do not know how to test for non-deterministic defects and unwanted emergent behavior.

Limits test automation (comparison of actual behavior to oracle):

- More complex oracle (success criteria is a set, not an instance)
  - Oracle as complex as system
  - Previous system version is less precise/accurate than new system
- Manual determination of success
- Operational testing
- Unlikely at unit test level (more system or system system)

Need to use simulation/modeling rather than operational environment (test preconditions and environmental inputs) to obtain controllability and visibility.

Testing in a Non-Deterministic World

Recommendations

Recommendations – 1

Explicitly address non-determinism in test planning.

Provide training and mentoring in non-deterministic defects and how to test for them.

Augment testing with concurrency analysis of architecture and design.

- Allocation of SW to threads, VMs, and cores
- Potential interference paths between threads, VMs, and cores
- Use of thread-safe class libraries

Emphasize automation of unit and integration testing, which are more likely to be deterministic.

- Static analysis
- Code coverage and boundary values (including range checking)

Instrument non-deterministic elements.

- Scrutinize logs for rare timing and other anomalies.

Recommendations – 2

Ensure existence of verifiable quality requirements.

Perform specialty engineering testing of quality requirements:

- *Capacity testing* including time-varying load testing, stress testing, and volume testing
- *Performance testing* for latency, jitter, response time, and throughput
- *Reliability testing* including soak testing
- *Robustness testing* of rare exceptional situations
- *Safety testing* based on hazard analysis (STAMP, misuse cases, or FMECA)
- *Security testing* based on threat analysis (abuse cases, attack trees, attack surfaces)

Recommendations – 3

Use modeling and simulation:

- Use M&S to control non-deterministic hardware and environmental inputs.
- Use very large numbers of simulation runs to find rare events. Google simulates 3 million miles of autonomous driving per day.
- Use combinatorial testing to achieve reasonable coverage.
- Verify models and simulations, which can also contain defects.

Create test suites that interleave high levels of realistic, high-risk, behavioral and rate variations.

Evaluate test results using general postcondition goals as oracles rather than irrelevant intermediate-behavior differences.

- Use statistical analysis when desired behavior is stochastic. Emphasize end-to-end mission thread operational testing.

Recommendations – 4

Use “standard best practices”:

- Continuous integration and testing
- Model-based testing (MBT)
- Use a Test Asset Management System, and treat test assets as first class work products.

Unit Testing:

- Test all externally controlled and non-deterministic exceptions, failures, and aberrant behaviors with extreme points
- Add scalable run-time invariant checking, enable and check

Component Testing:

- Test every cause and every effect at least once, separately
- Test every “illegal” sequence for an appropriate response

System Testing:

- Test crash, recovery, and restart under realistic conditions

Contact Information

Presenter

Donald Firesmith

Principal Engineer

Telephone: +1 412.268.6874

Email: dgf@sei.cmu.edu