Testing Safety Critical Applications

Geoffrey Bessin

Product Manager – Test RealTime

Safety Critical Applications

If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone inside.

- Robert X. Cringely

Agenda

- Defining Our Terms
- Design Responsibilities
- Test Responsibilities
- Conclusion

What is a Safety Critical Application?

- Safety Critical Application:

An application where human safety is dependent upon the correct operation of the system

- Examples
  - Aircraft fly-by-wire system
  - Railway signaling systems
  - Medical devices
  - Traffic light
  - CAD Tools

Minimization of Risk

- Risk = magnitude of danger * probability of exposure
- IEC61508 Safety Integrity Levels
  - Level 4 – PFD 1:10,000 to 1:100,000
    Catastrophic Community Impact
  - Level 3 – PFD 1:1000 to 1:10,000
    Employee/Community Impact
  - Level 2 – PFD 1:100 to 1:1000
    Major Property/Production Protection – Possible Employee Injury
  - Level 1 – PFD 1:10 to 1:100
    Minor Property/Production Protection

Obligatory Safety Critical Example

- Ariane 5, 1996
- Exploded 40 seconds after launch
- Had reused Ariane 4 code
- Code was felt to be adequate
- Ariane 5 was too fast!
- 64-bit number was stuffed into 16-bit variable – overflow error

Safety Critical Standards

- ISO9001 – Recommended minimum standard of quality
- IEC1508 – General standard
- EN50128 – Railway Industry
- IEC880 – Nuclear Industry
- RTCA/DO178B – Avionics and Airborne Systems
- MISRA – Motor Industry
- Defence Standard 00-55/00-56

RIGOR

ACCOUNTABILITY

Strategies for Avoiding Critical Software Failure

- Design Diversity
- Fault Avoidance
- Extensive Testing

Agenda

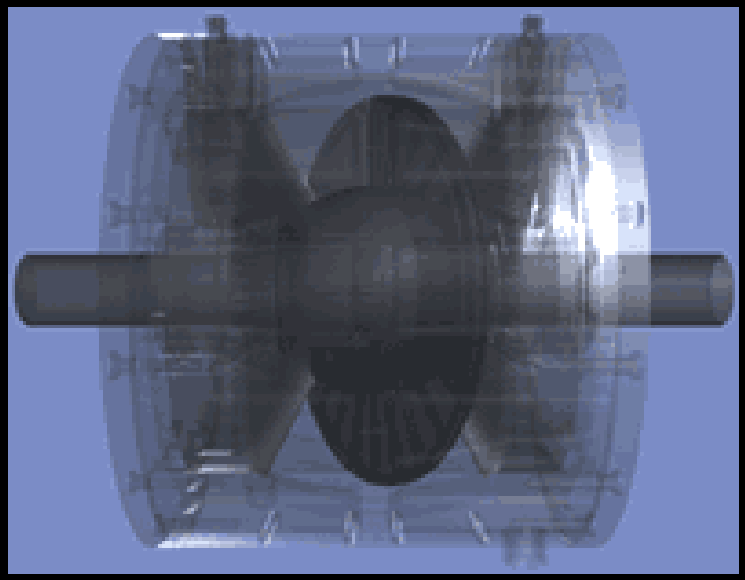
- Defining Our Terms
- Design Responsibilities
- Test Responsibilities
- Conclusion

Software Design

One way is to make it so simple that there are obviously no deficiencies and the other is to make it so complicated that there are no obvious deficiencies.

C.A.R. Hoare

Simplicity



Preventing Bugs – Not Just Testing

- Mature Development Processes
  - Capability Maturity Model
- Inspection Methods
  - Walkthroughs, formal inspections, code reading
- Design Styles
  - Testability
  - Clarity
- Languages
  - Strong typing
  - Runtime constraint checking
  - Parameter checking

Testability

The degree to which a system or component facilitates the establishment of test criteria and the performance of tests to determine whether those criteria have been met.

The effort required to apply a given testing strategy to a system.

The ease with which faults in a system can be made to reveal themselves during testing.

Since You Can't Test Safety Into An Application….

OPTION ONE

Specify, design and build the perfect system.

- Document everything
- Review everything
- Test everything

OPTION TWO

Aim for Option One, but accept that man is flawed.

- Include error detection and recovery capabilities
- Iterate

Agenda

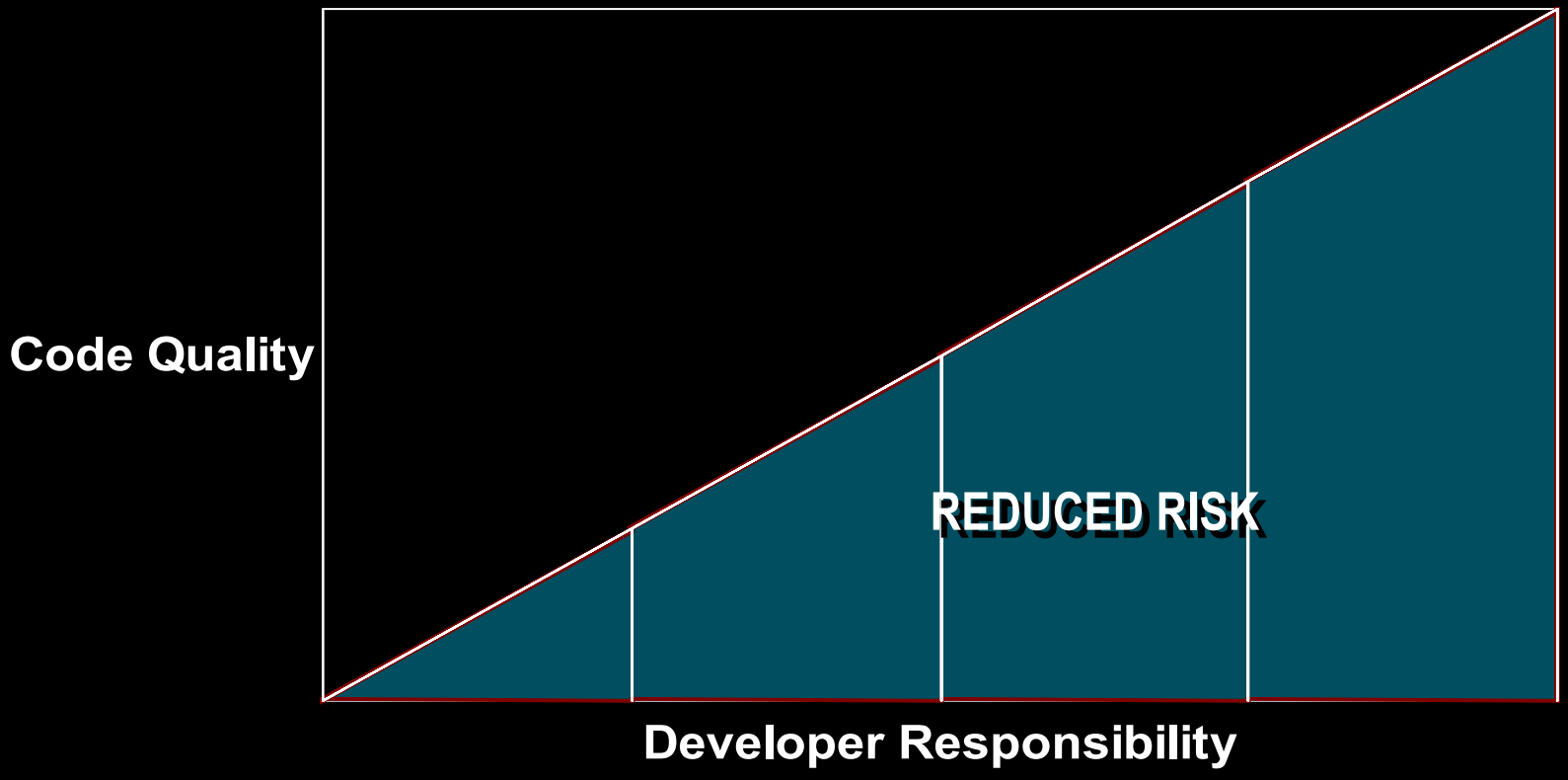
- Defining Our Terms
- Design Responsibilities
- Test Responsibilities
- Conclusion

Facts

- On average, there are 3 bugs per 100 statements
- Testing typically consumes at least 40% of software development labor
- 50% of software projects are behind schedule
- 25% of software projects are abandoned

Hardware's Lessons

- Code ownership
- Test responsibility
- Field execution



Software Testing

- Test First
  - encourages explicit definition of implementation scope
  - helps separate logical design from physical design from implementation
  - grows confidence in the correct functioning of the system as the system grows
  - simplifies your designs
- Pair Program
  - Two-heads ARE better than one
- Use Simulation
  - Using the intended processor and I/O ports (if applicable)
- Static Metrics
  - Complexity metrics
  - Code adherence to formally defined conventions

Fundamental Resposibilities

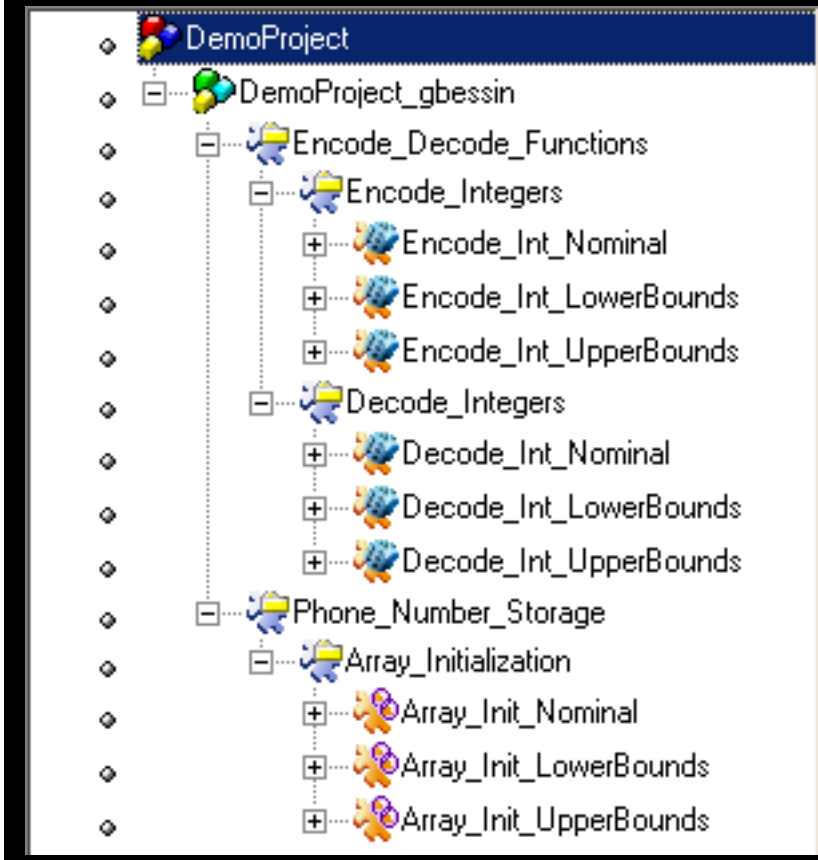
- Unit Testing
- Code Coverage Analysis
- Requirements Traceability

Unit Testing

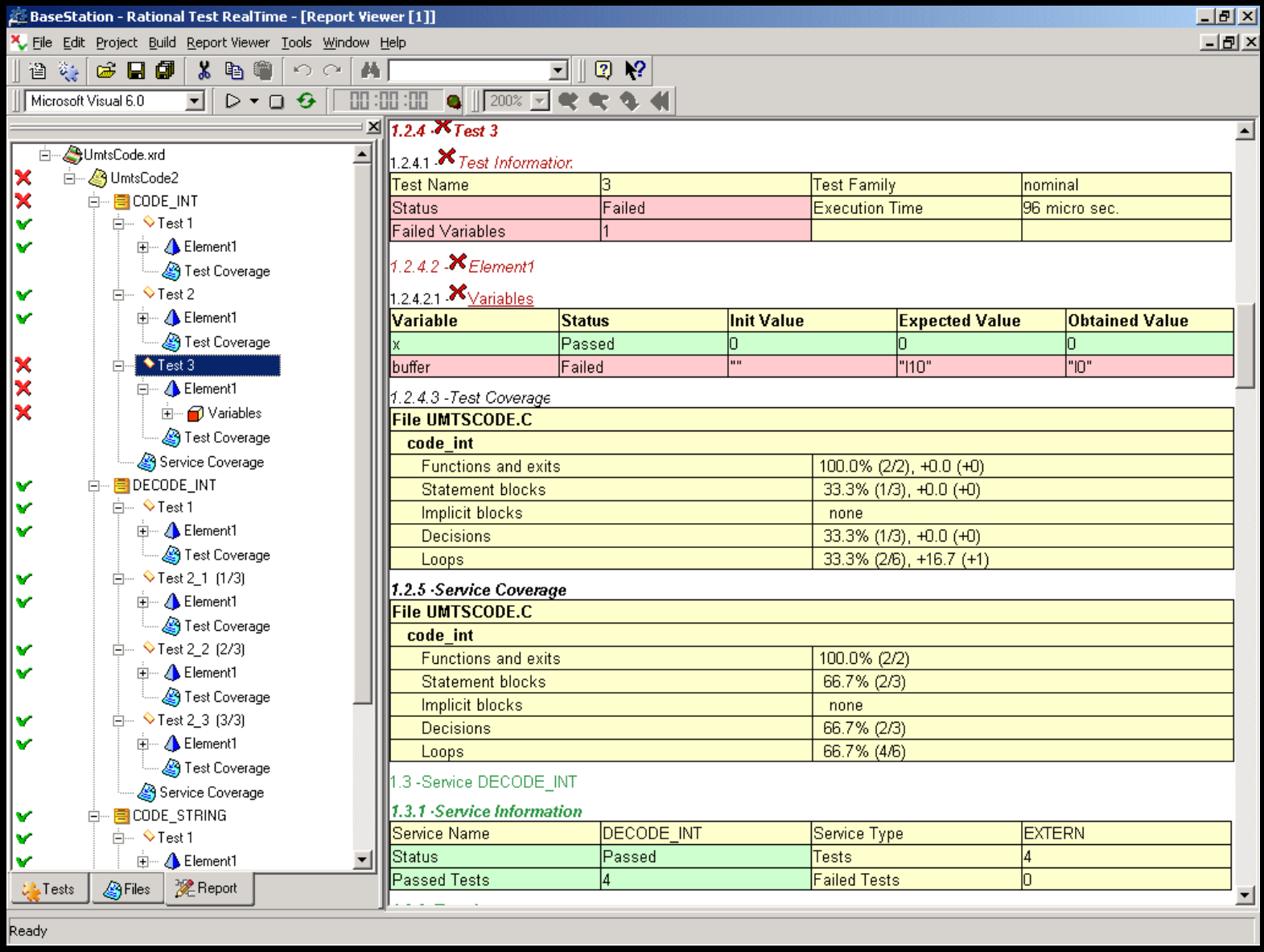
- Test harness, stub, driver creation
- Clear data definition
  - Equivalence classes
  - Boundary conditions
- Batch execution for regression testing
- Easily traceable error reporting
- Amenable to clear documentation
- Versionability

Test Class/Data Definition – Regression Testing

```
SERVICE decode_int
SERVICE_TYPE extern

    -- Tested service parameters declarations
    #int x;
    #char buffer[200];
    #char *ret;

    TEST 1
    FAMILY nominal

        ELEMENT
         VAR x,          init = 0,          ev = 3
         VAR buffer,          init = "I13",          ev ==
         VAR ret,      init==,     ev=&buffer[3]
         #ret=decode_int(&x, buffer);
        END ELEMENT

    END TEST -- TEST 1

    TEST 2
    FAMILY nominal

        ELEMENT
         VAR buffer,          init in {"I243","I265","I287"},          ev ==
         VAR x,          init = 0,          ev(buffer) in {34,56,78}
         VAR ret,      init==,     ev=&buffer[4]
         #ret=decode_int(&x, buffer);
        END ELEMENT

    END TEST -- TEST 2

END SERVICE -- decode_int
```



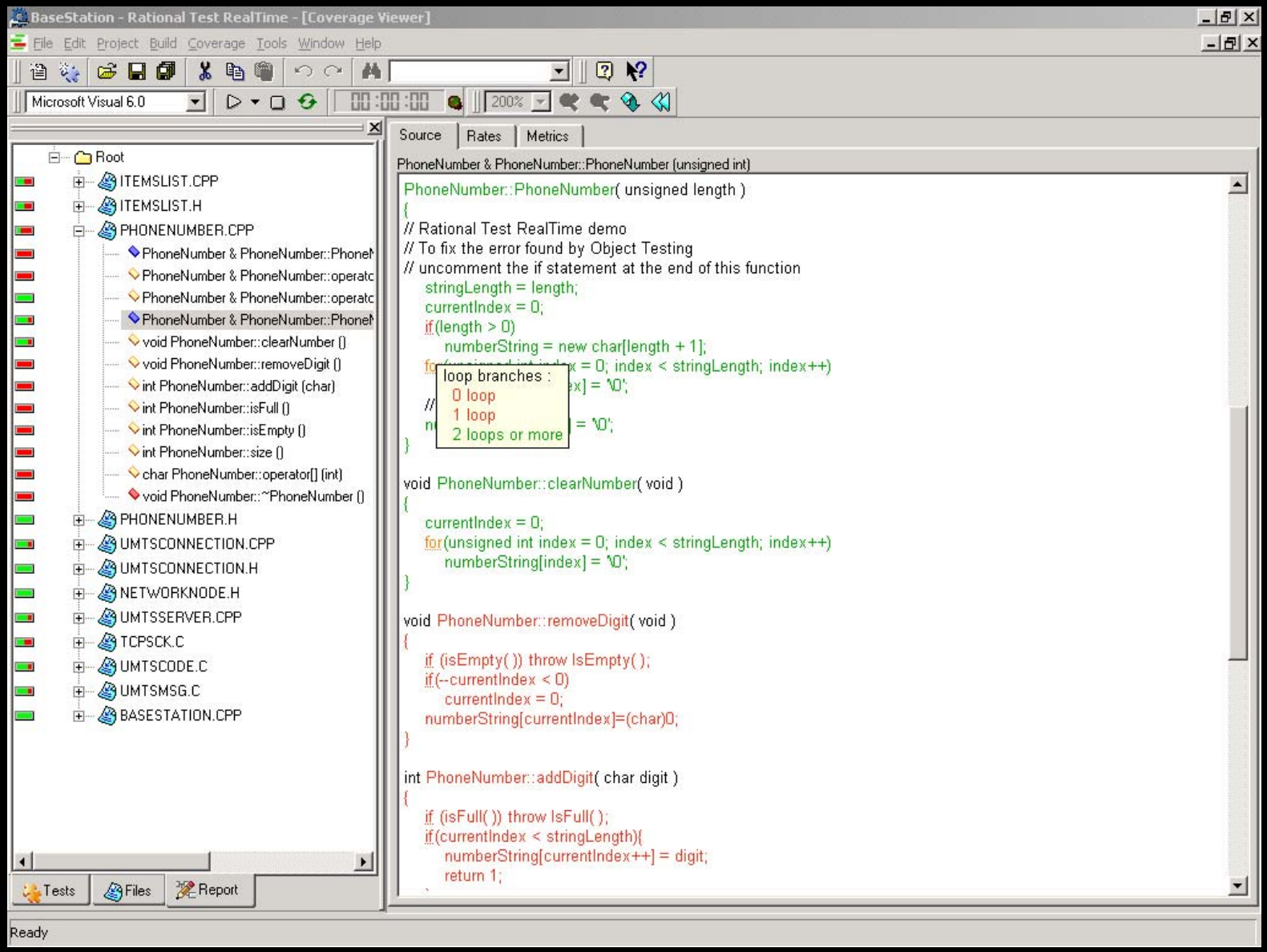
Test Reporting



Code Coverage

- Coverage Levels
  - Procedure Entries and Exits
  - Calls
  - Statements, Decisions, Loops
  - Basic, Forced and MC/DC Conditions
- Clear linkage to test cases
  - Eliminate test redundancy
- Support safety critical coverage levels
- Easily documented

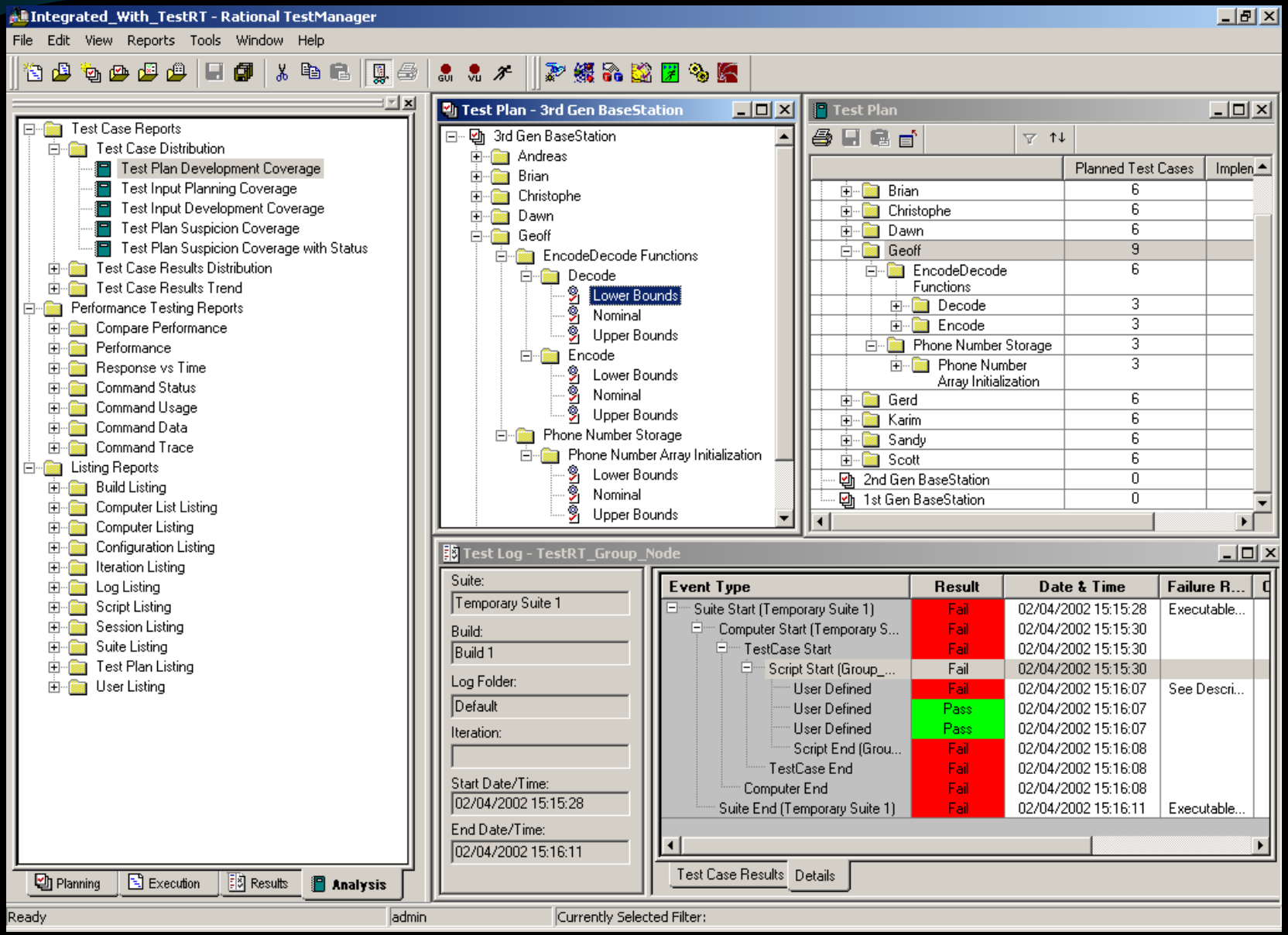
Multi-Level Code Coverage



Requirements Traceability

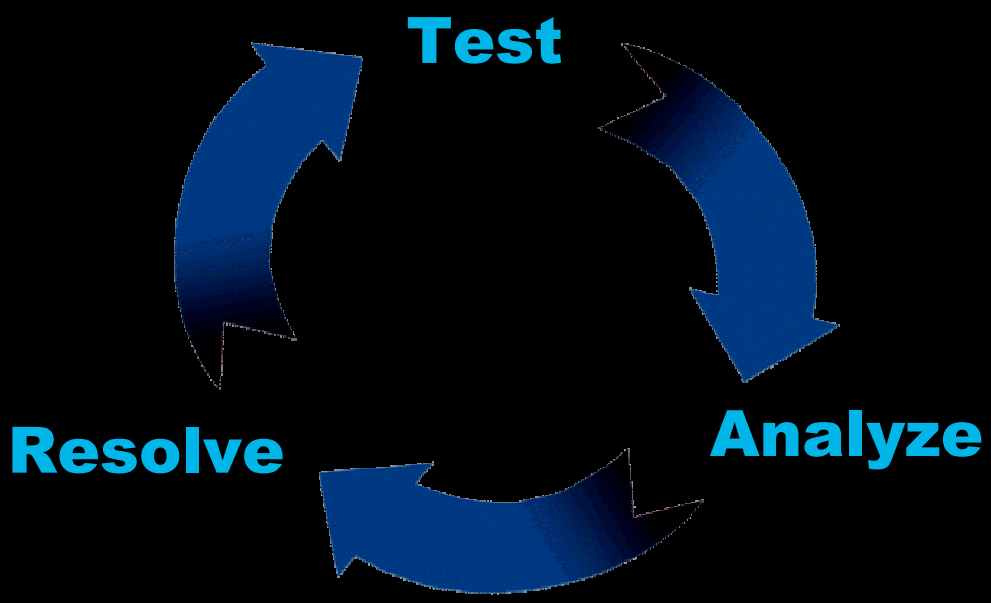
- Cornerstone to safety-critical development standards
- Properly implemented traceability
  - Ensures all requirements have tests
  - Ensures tests are updated when requirements change
- Enables independent verification
- Be Careful!
  - Requirements coverage is necessary but not sufficient
  - Does all code come from a requirement?

Test Management



Automation Options

- Automatic, target-independent test deployment
- Linkage between test results and source code
- Runtime Analysis
  - Memory leak detection
  - Performance Profiling
  - Runtime Tracing



Agenda

- Defining Our Terms
- Design Responsibilities
- Test Responsibilities
- Conclusion

The Old Days

Testing Safety Critical Applications - Conclusion

- Present
  - Responsibility lies with developer
  - Design and test are equally crucial
  - Developers need proper test training
- Future
  - Formal requirements defintion is enabled by automated toolsets
  - Formal requirements are translated into UML-like models
  - Models serve to generate both tests and code
  - Traceability is “built” into the model


Thank
You