



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

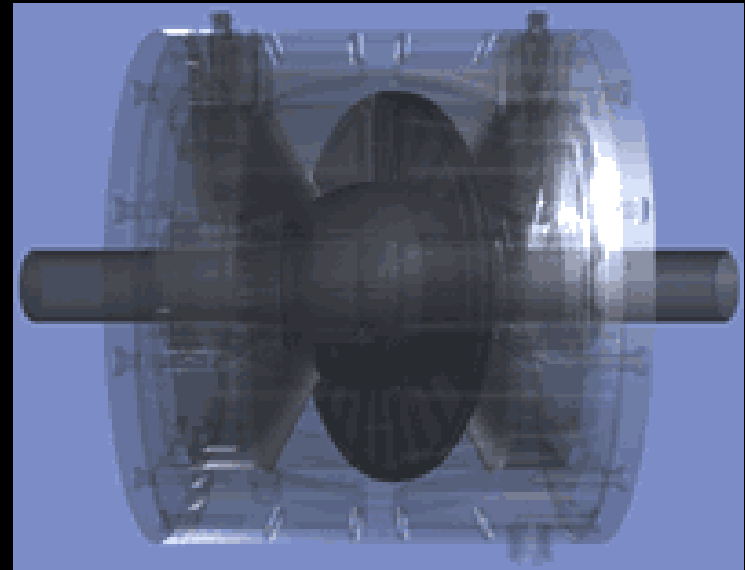
Software Design

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

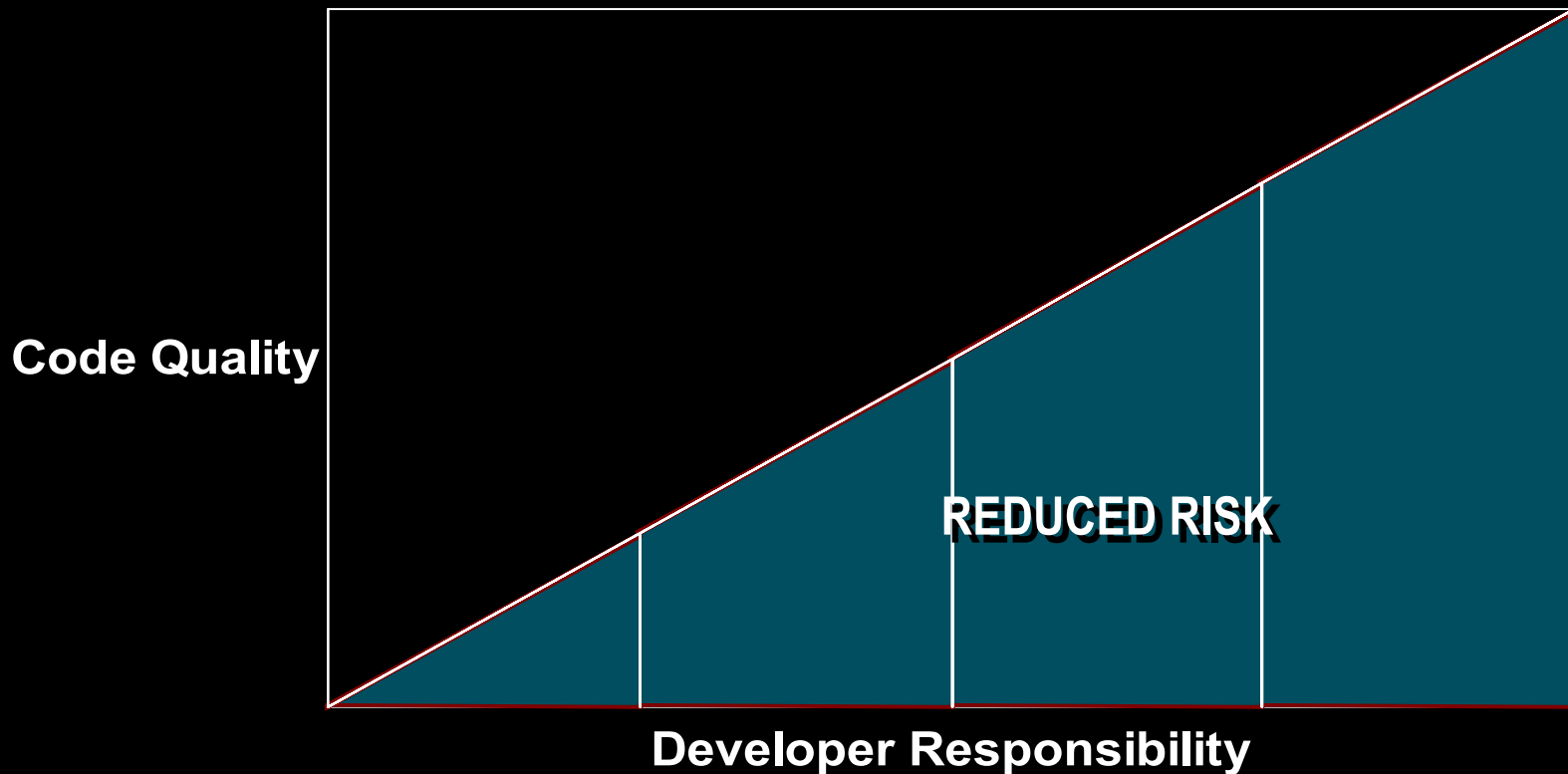
Facts

- ◆ 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 Responsibilities

- ◆ 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

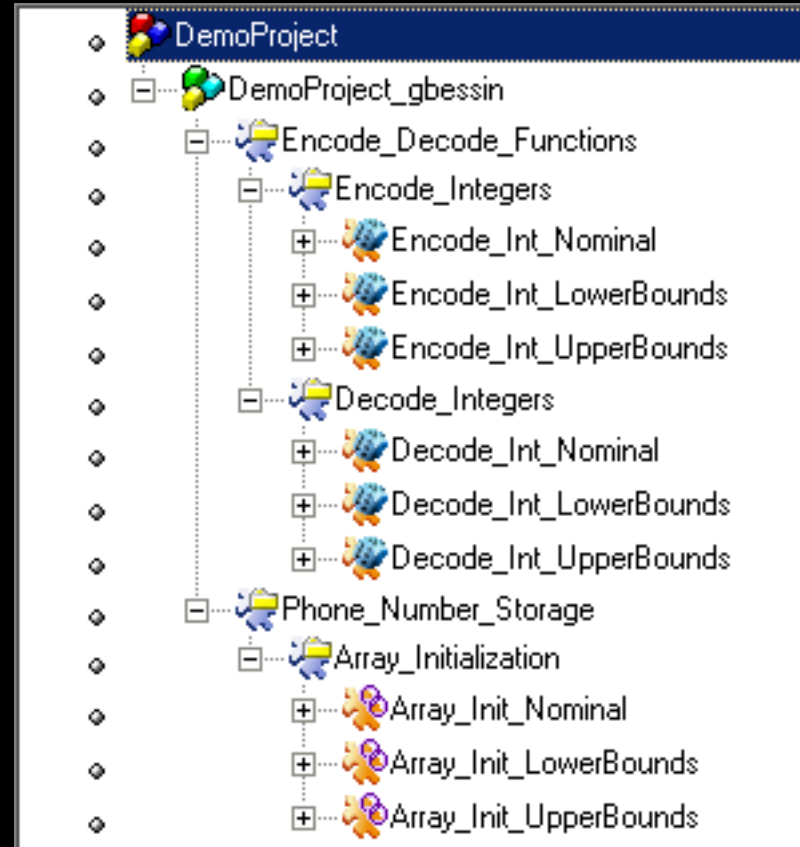
END TEST -- TEST 1

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

BaseStation - Rational Test RealTime - [Report Viewer [1]]

File Edit Project Build Report Viewer Tools Window Help

Microsoft Visual 6.0

00:00:00 200%

UmtsCode.xrd

- UmtsCode2
 - CODE_INT
 - Test 1
 - Element1
 - Test Coverage
 - Test 2
 - Element1
 - Test Coverage
 - Test 3
 - Element1
 - Variables
 - Test Coverage
 - Service Coverage
 - DECODE_INT
 - Test 1
 - Element1
 - Test Coverage
 - Test 2_1 (1/3)
 - Element1
 - Test Coverage
 - Test 2_2 (2/3)
 - Element1
 - Test Coverage
 - Test 2_3 (3/3)
 - Element1
 - Test Coverage
 - Service Coverage
 - CODE_STRING
 - Test 1
 - Element1

1.2.4 Test 3

1.2.4.1 Test Information:

Test Name	3	Test Family	nominal
Status	Failed	Execution Time	96 micro sec.
Failed Variables	1		

1.2.4.2 Element1

1.2.4.2.1 Variables

Variable	Status	Init Value	Expected Value	Obtained Value
x	Passed	0	0	0
buffer	Failed	""	"110"	"10"

1.2.4.3 -Test Coverage

File UMTSCODE.C

code_int	
Functions and exits	100.0% (2/2), +0.0 (+0)
Statement blocks	33.3% (1/3), +0.0 (+0)
Implicit blocks	none
Decisions	33.3% (1/3), +0.0 (+0)
Loops	33.3% (2/6), +16.7 (+1)

1.2.5 -Service Coverage

File UMTSCODE.C

code_int	
Functions and exits	100.0% (2/2)
Statement blocks	66.7% (2/3)
Implicit blocks	none
Decisions	66.7% (2/3)
Loops	66.7% (4/6)

1.3 -Service DECODE_INT

1.3.1 -Service Information

Service Name	DECODE_INT	Service Type	EXTERN
Status	Passed	Tests	4
Passed Tests	4	Failed Tests	0

Tests Files Report

Ready

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

The screenshot displays the Rational Test RealTime Coverage Viewer interface. The window title is "BaseStation - Rational Test RealTime - [Coverage Viewer]". The menu bar includes File, Edit, Project, Build, Coverage, Tools, Window, and Help. The toolbar shows various icons for file operations and execution. The status bar at the top indicates "Microsoft Visual 6.0" and "200%".

The left pane shows a project tree with the following structure:

- Root
 - ITEMSLIST.CPP
 - ITEMSLIST.H
 - PHONENUMBER.CPP
 - PhoneNumber & PhoneNumber::PhoneN
 - PhoneNumber & PhoneNumber::operatc
 - PhoneNumber & PhoneNumber::operatc
 - PhoneNumber & PhoneNumber::PhoneN
 - void PhoneNumber::clearNumber ()
 - void PhoneNumber::removeDigit ()
 - int PhoneNumber::addDigit (char)
 - int PhoneNumber::isFull ()
 - int PhoneNumber::isEmpty ()
 - int PhoneNumber::size ()
 - char PhoneNumber::operator[] (int)
 - void PhoneNumber::~PhoneNumber ()
 - PHONENUMBER.H
 - UMTSCONNECTION.CPP
 - UMTSCONNECTION.H
 - NETWORKNODE.H
 - UMTSSERVER.CPP
 - TCPSCK.C
 - UMTSCODE.C
 - UMTSMMSG.C
 - BASESTATION.CPP

The right pane shows the source code for "PhoneNumber & PhoneNumber::PhoneNumber (unsigned int)". The code is as follows:

```
PhoneNumber::PhoneNumber( unsigned length )
{
    // Rational Test RealTime demo
    // To fix the error found by Object Testing
    // uncomment the if statement at the end of this function
    stringLength = length;
    currentIndex = 0;
    if (length > 0)
        numberString = new char[length + 1];
    for (unsigned int index = 0; index < stringLength; index++)
        numberString[index] = '\0';
    numberString[currentIndex] = '\0';
}

void PhoneNumber::clearNumber( void )
{
    currentIndex = 0;
    for (unsigned int index = 0; index < stringLength; index++)
        numberString[index] = '\0';
}

void PhoneNumber::removeDigit( void )
{
    if (isEmpty()) throw IsEmpty();
    if (--currentIndex < 0)
        currentIndex = 0;
    numberString[currentIndex] = (char)0;
}

int PhoneNumber::addDigit( char digit )
{
    if (isFull()) throw IsFull();
    if (currentIndex < stringLength){
        numberString[currentIndex++] = digit;
        return 1;
    }
}
```

A yellow callout box highlights the loop in the first function with the text "loop branches: 0 loop, 1 loop, 2 loops or more".

The bottom of the window has buttons for "Tests", "Files", and "Report". The status bar at the very bottom says "Ready".

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

Integrated_With_TestRT - Rational TestManager

File Edit View Reports Tools Window Help

GUI VU

Test Case Reports

- Test Case Distribution
 - Test Plan Development Coverage
 - Test Input Planning Coverage
 - Test Input Development Coverage
 - Test Plan Suspicion Coverage
 - Test Plan Suspicion Coverage with Status
- Test Case Results Distribution
- Test Case Results Trend
- Performance Testing Reports
 - Compare Performance
 - Performance
 - Response vs Time
- Command Status
- Command Usage
- Command Data
- Command Trace
- Listing Reports
 - Build Listing
 - Computer List Listing
 - Computer Listing
 - Configuration Listing
 - Iteration Listing
 - Log Listing
 - Script Listing
 - Session Listing
 - Suite Listing
 - Test Plan Listing
 - User Listing

Test Plan - 3rd Gen BaseStation

- 3rd Gen BaseStation
 - Andreas
 - Brian
 - Christophe
 - Dawn
 - Geoff
 - EncodeDecode Functions
 - Decode
 - Lower Bounds
 - Nominal
 - Upper Bounds
 - Encode
 - Lower Bounds
 - Nominal
 - Upper Bounds
 - Phone Number Storage
 - Phone Number Array Initialization
 - Lower Bounds
 - Nominal
 - Upper Bounds

Test Plan

	Planned Test Cases	Implem
Brian	6	
Christophe	6	
Dawn	6	
Geoff	9	
EncodeDecode Functions	6	
Decode	3	
Encode	3	
Phone Number Storage	3	
Phone Number Array Initialization	3	
Gerd	6	
Karim	6	
Sandy	6	
Scott	6	
2nd Gen BaseStation	0	
1st Gen BaseStation	0	

Test Log - TestRT_Group_Node

Suite: Temporary Suite 1

Build: Build 1

Log Folder: Default

Iteration:

Start Date/Time: 02/04/2002 15:15:28

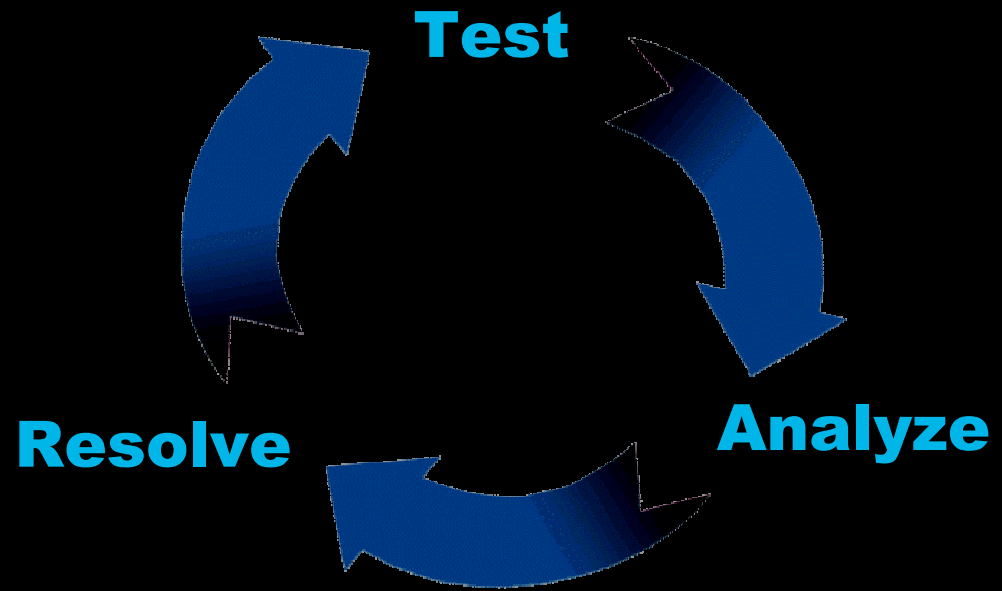
End Date/Time: 02/04/2002 15:16:11

Event Type	Result	Date & Time	Failure R...
Suite Start (Temporary Suite 1)	Fail	02/04/2002 15:15:28	Executable...
Computer Start (Temporary S...	Fail	02/04/2002 15:15:30	
TestCase Start	Fail	02/04/2002 15:15:30	
Script Start (Group...	Fail	02/04/2002 15:15:30	
User Defined	Fail	02/04/2002 15:16:07	See Descri...
User Defined	Pass	02/04/2002 15:16:07	
User Defined	Pass	02/04/2002 15:16:07	
Script End (Grou...	Fail	02/04/2002 15:16:08	
TestCase End	Fail	02/04/2002 15:16:08	
Computer End	Fail	02/04/2002 15:16:08	
Suite End (Temporary Suite 1)	Fail	02/04/2002 15:16:11	Executable...

Test Case Results Details

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 definition 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