

Abstract Semantic Analysis vs White-Box Testing

PolySpace Technologies Inc.
Chris HOTE

www.polyspace.com / *hote@polyspace.com*

Cost of Bugs (Macro-economy)

*Software failures cost up to \$60bn yearly to the US economy
(30%+ is software manufacturers addressing **bug tracking**)*

Software testing infrastructure could drastically reduce costs by:

- ✦ *Locating the source of bugs faster and with more precision;*
- ✦ *Detecting bugs earlier in the software development process;*
- ✦ *Removing more bugs before software is released.*

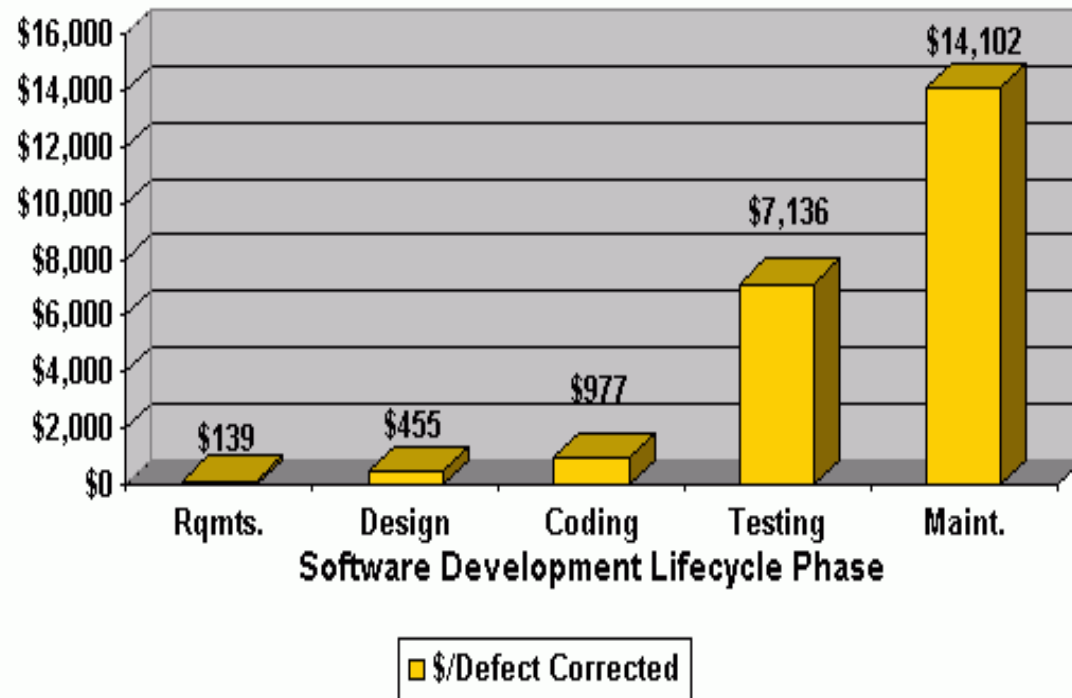
May 2002 NIST report:

www.nist.gov/director/prog-ofc/report02-3.pdf

Costs of bugs (Program level)

Costs of Correcting Defects

Source: B. Boehm and V. Basili, "Software Defect Reduction Top 10 List," *IEEE Computer*



Early detection of errors dramatically reduces testing costs

Which errors?

The IBM/Berkeley study

- Study conducted at *Berkeley* and *IBM Watson* found that *30-40% of software defects* addressed during a four-year maintenance phase on large IBM codes *are due to runtime errors*.
- M. Sullivan and R. Chillarege, Software defects and their impact on system availability, proc. 21th International Symposium On Fault-Tolerant Computing (FTCS-21), Montreal, 1991, 2-9, IEEE Press.

What are Runtime Errors?

*A well defined **BUG (Latent Fault)** that causes non determinism, incorrect results or processor halt*

- ✦ *De-referencing through null or out-of-bounds pointers (memory corruption)*
- ✦ *Out-of-bounds array access*
- ✦ *Read access to non-initialized data*
- ✦ *Access conflict on shared data (data corruption)*
- ✦ *Illegal type conversion*
- ✦ *Overflow / underflow on scalar operations*
- ✦ *Invalid arithmetic operations, e.g.:
division by zero, square root of a negative number*
- ✦ *Dead code*

Conventional Techniques

Static Approaches

Techniques	Limits
Manual review, Code Inspection	- <i>Effective but painful and expensive</i>
Static analyzers, Complexity Measurements, Programming Rule checking. <i>Compiler, lint-like, QAC, McCabe, AdaTest, Cantata, Logiscope, LDRA TestBed, ...</i>	<ul style="list-style-type: none"> - <i>Syntax or static semantic oriented: Weak Support for Bug Detection</i> - <i>Imprecise: Numerous warning messages to review</i> - <i>Good support for mastering S/W maintenance costs</i>

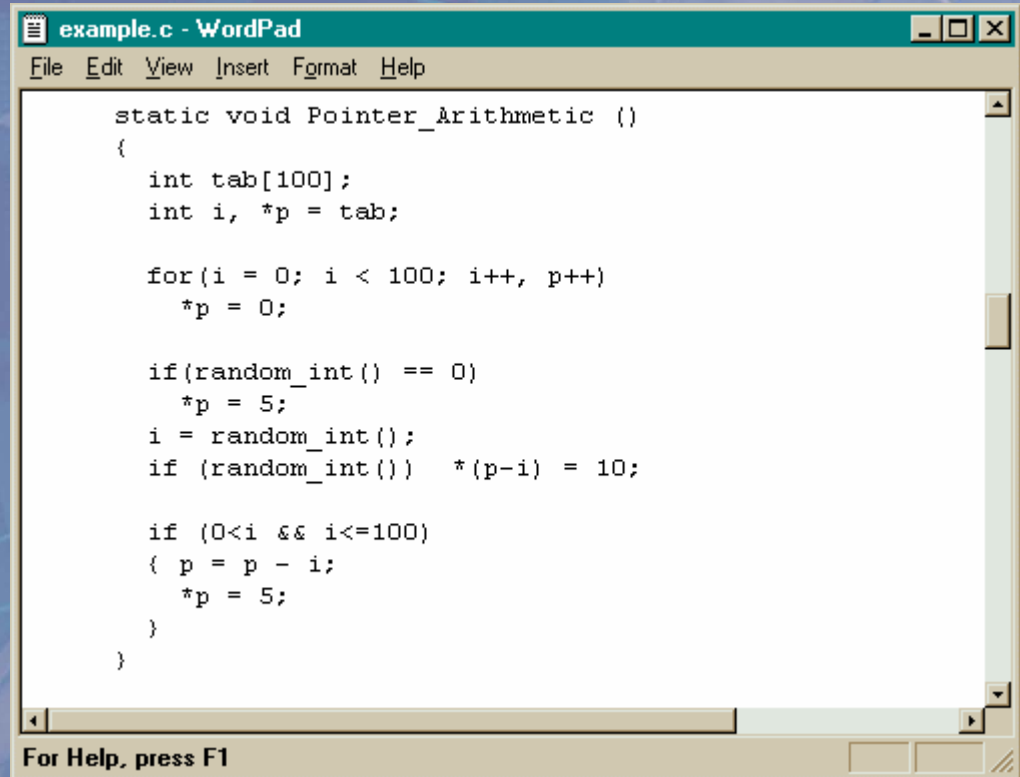
Conventional Techniques

Dynamic Approaches

Techniques	Limits
Simulation Dynamic Testing White box testing <i>RTT, AdaTest, Vector...</i>	<ul style="list-style-type: none"> - Tests cases must be developed and run - Stubs must be developed on target - Lead to Additional Debugging Time - Do not catch all errors
Code Instrumentation <i>Insure++, Purify ...</i>	<ul style="list-style-type: none"> - Intrusive: Source or Binary Code Modifications - Not always applicable to embedded S/W
Code Coverage <i>QAC, McCabe, AdaTest, Logiscope, LDRA TestBed, ...</i>	<ul style="list-style-type: none"> - Doesn't Catch All Errors

How may traditional testing techniques be improved?

- Pointer 'p' is modified several times but few operations might cause a runtime error depending on 'for' loop and 'if' condition.



```

static void Pointer_Arithmetic ()
{
    int tab[100];
    int i, *p = tab;

    for(i = 0; i < 100; i++, p++)
        *p = 0;

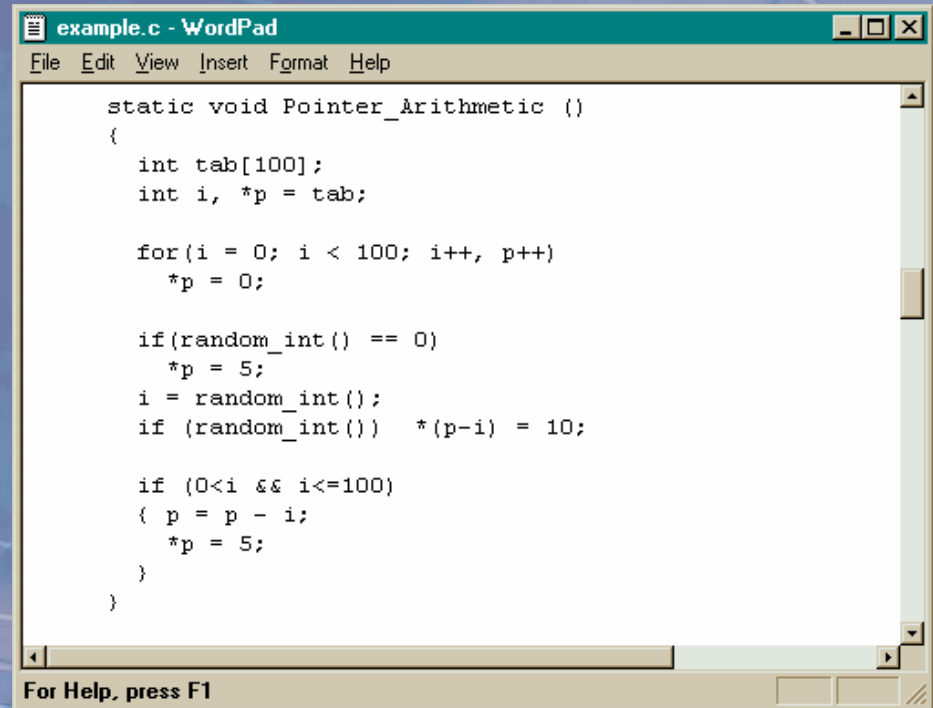
    if(random_int() == 0)
        *p = 5;
    i = random_int();
    if (random_int()) *(p-i) = 10;

    if (0<i && i<=100)
    { p = p - i;
      *p = 5;
    }
}
    
```

- Static analyzers do not take dynamic of control structures (loops, if, switch, function calls) and relationship between variables into account

How may traditional testing techniques be improved?

- Line “*p = 5” would cause memory corruption that is not detectable if random_int is unknown or if code is not instrumented



```

static void Pointer_Arithmetic ()
{
    int tab[100];
    int i, *p = tab;

    for(i = 0; i < 100; i++, p++)
        *p = 0;

    if(random_int() == 0)
        *p = 5;
    i = random_int();
    if (random_int()) *(p-i) = 10;

    if (0<i && i<=100)
    { p = p - i;
      *p = 5;
    }
}
    
```

- Dynamic testing tools do not detect errors but their consequences causing extra debugging time.
- Errors are found late in the S/W development process only if the right test scenario is part of test basis, if code was instrumented and if instrumentation libraries are available for target processor..

Solution based on Abstract Semantic Analysis

Runtime errors detection at compile time

Green
safe

Red
bug

Grey
dead

Orange
warning

```

62     static void Pointer_Arithmetic ()
63     {
64         int tab[100];
65         int i, *p = tab;
66
67         for(i = 0; i < 100; i++, p++)
68             *p = 0;
69
70         if(random_int() == 0)
71             { *p = 5; /* Out of bounds */
72               ++i;    /* Unreachable (runtime error on previous line) */
73             }
74
75         i = random_int();
76         if (random_int()) *(p-i) = 10;
77
78         if (0 < i && i <= 100)
79             { p = p - i;
80               *p = 5;      /* Safe pointer access */
81             }
82     }

```

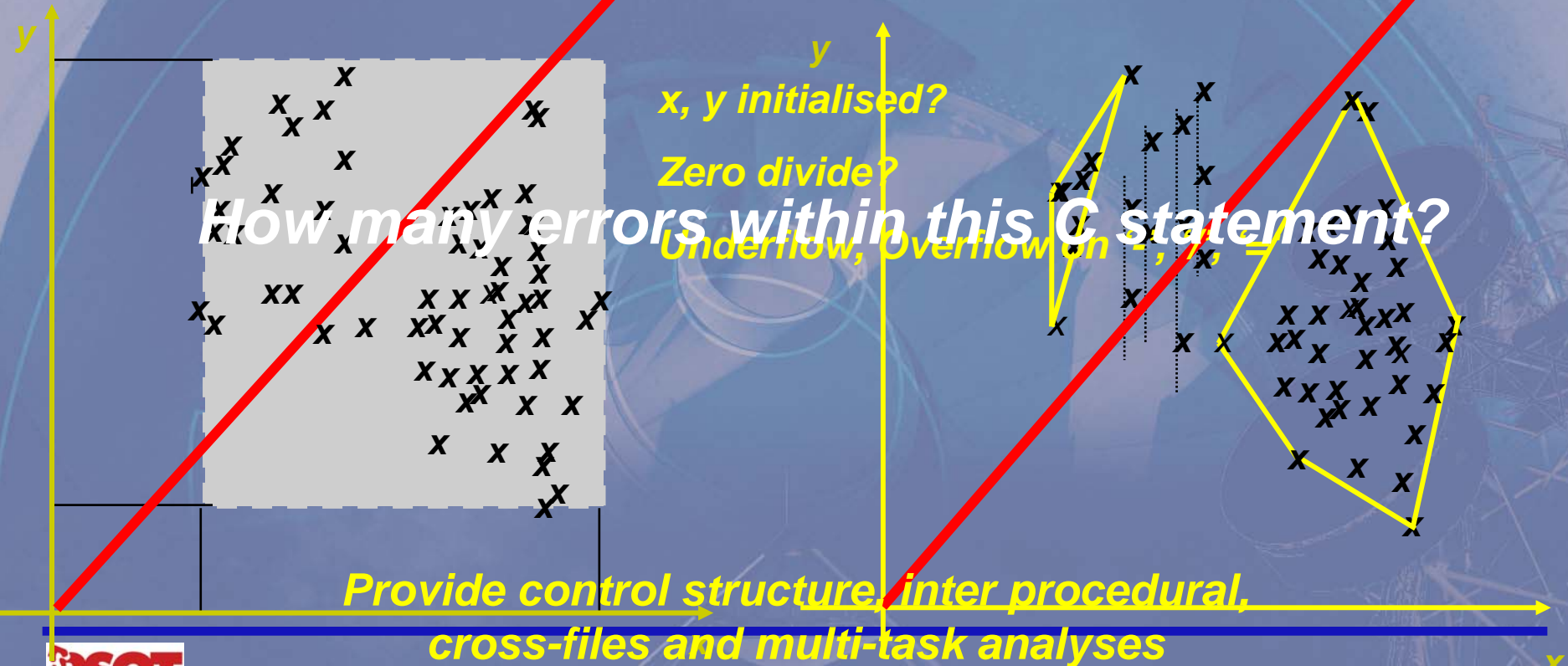
Solution based on Abstract Semantic Analysis

Bridging the gap between static and dynamic testing approaches

*Dynamic Testing,
Interval Analysis*

*Static Analysis based
on Abstract Semantic*

Verifying: $x = x / (x - y); ?$



Main Characteristics of Semantic Analysis

- Semantically based:
Errors are directly detected not thru their consequences as it happens during testing (no debugging)
- Exhaustive:
All possible input parameter combinations are scrutinized (far beyond what testing can offer)
- Applicable at once: Source code only
 - No test cases to write,
 - No code instrumentation needed
 - No execution

Comparison with Conventional Testing Approaches

- Semantic analysis helps in three different areas:
 - Find the bugs other testing tools might have found, but much faster (no debugging, no tests executed)
 - Find the bugs that are not found during tests, because the corresponding test sequence was not executed (e.g.: overflow)
 - Find the bugs that are not found, because when they happened during tests, no anomaly was noticeable (e.g.: non-initialized data, memory corruption)

Main Benefits of Semantic Analysis

- Non-intrusive and scalable: applicable as soon as during coding phase (earliest detection of errors)
- Automatic, repeatable (cpu-based) and quantitative results (reproducible quality guidelines)
- Cost-effective:
Who can afford fixing syntax errors today without compiler log file?
The solution exists now for runtime errors.

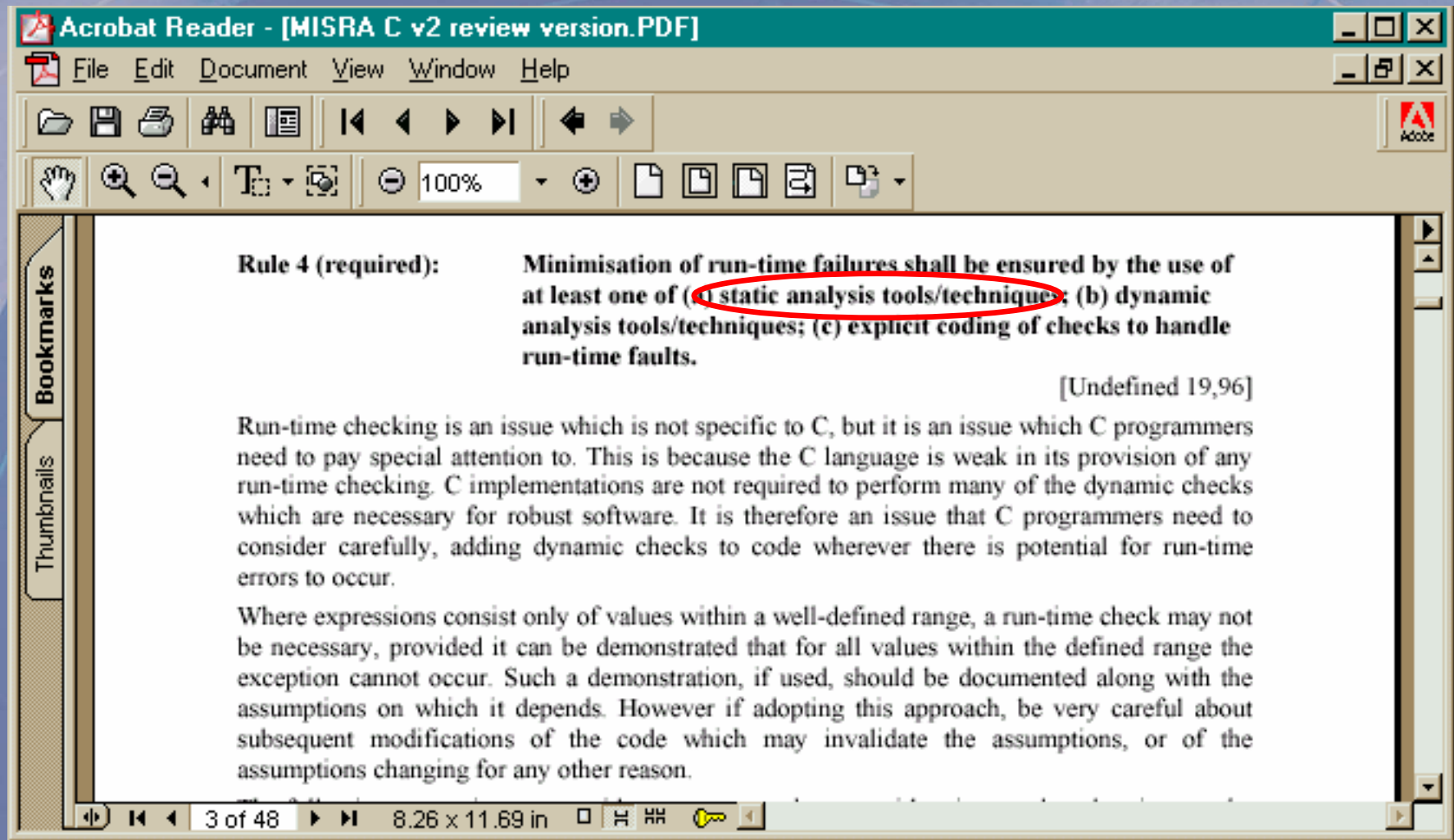
Semantic Analysis and Market Standards

- Runtime errors cause major issues in the embedded systems industries as they are difficult to reproduce and may cause substantial damages including program delay, product recall, corporate image alteration
- Thanks to static analysis technology improvement, more standards adopt those techniques as a viable alternative to white-box tests and code inspection

Semantic Analysis/MISRA

- Standard for Automotive Industry
 - Control and Data Flow Analysis (concurrent access to critical resources, documentation)
 - Code Inspection (guidelines)
 - Suppressing White-box tests at Unit Level
 - Suitable for checking S/W robustness against calibration data versatility

Semantic Analysis/MISRA



➤ Standard for Medical Device Industry

- *The FDA's analysis of 3140 medical device recalls conducted between 1992 and 1998 reveals that 242 of them (7.7%) are attributable to software failures.*
- *Software quality assurance needs to focus on preventing the introduction of defects into the software development process and not on trying to "test quality into" the software code after it is written.*
- *Software testing is one of many verification activities intended to confirm that software development output meets its input requirements. Other verification activities include various static and dynamic analyses, code and document inspections, walkthroughs, and other techniques.*

See <http://www.fda.gov/cdrh/comp/guidance/938.html>

Semantic Analysis/DO178-B

- Recommendations for Airborne Systems Industry
 - Control and Data Flow Analysis (concurrent access to critical resources, documentation)
 - Code Inspection (guidelines)
 - Unit test targeting arithmetic issues (overflow, arithmetic exceptions) and memory corruptions (illegal de-referenced pointers)

Semantic Analysis/CENELEC

- Standard for Railways Transportation Industry
 - Control and Data Flow Analysis (concurrent access to critical resources, documentation)
 - Code Inspection (guidelines)
 - Suppressing White-box tests at Unit Level: suitable for SIL-3 and 2 system certification)

Direct Benefits of early detection of errors

Activity	Cost without static detection of errors	Cost with static detection of errors
Debugging	1 day per error (How many bugs found in this file during tests? Which cost?)	5' minutes per error highlighted by semantic analysis
White-box testing	2 days/1000SLOCs	-0-
Peer review	1 day/1000SLOCs	1 hour/1000SLOCs
Design review	One month/Application	two days/Application
Savings		\$2-4 per LOC

Cost of late detection of errors, Value of comprehensive testing

Activity	Cost without static detection of errors	Cost with static detection of errors
Project planning	Unplanned expenses, Resource planning, Schedule Integrity, Multiple version control	Lessened Improved Improved Lessened, As a result of fewer bugs
On-site maintenance costs	?	Higher visibility on remaining S/W reliability breaches before code freeze and delivery
Corporate image	?	
Savings		Unparalleled quality improvement (exhaustiveness and repeatability of the semantic analysis)

Conclusions

- Semantic analysis now available for the automatic detection of runtime errors at compile time
- Goes beyond traditional testing at lower cost
- Provides immediate results
- The ultimate barrier to latent faults