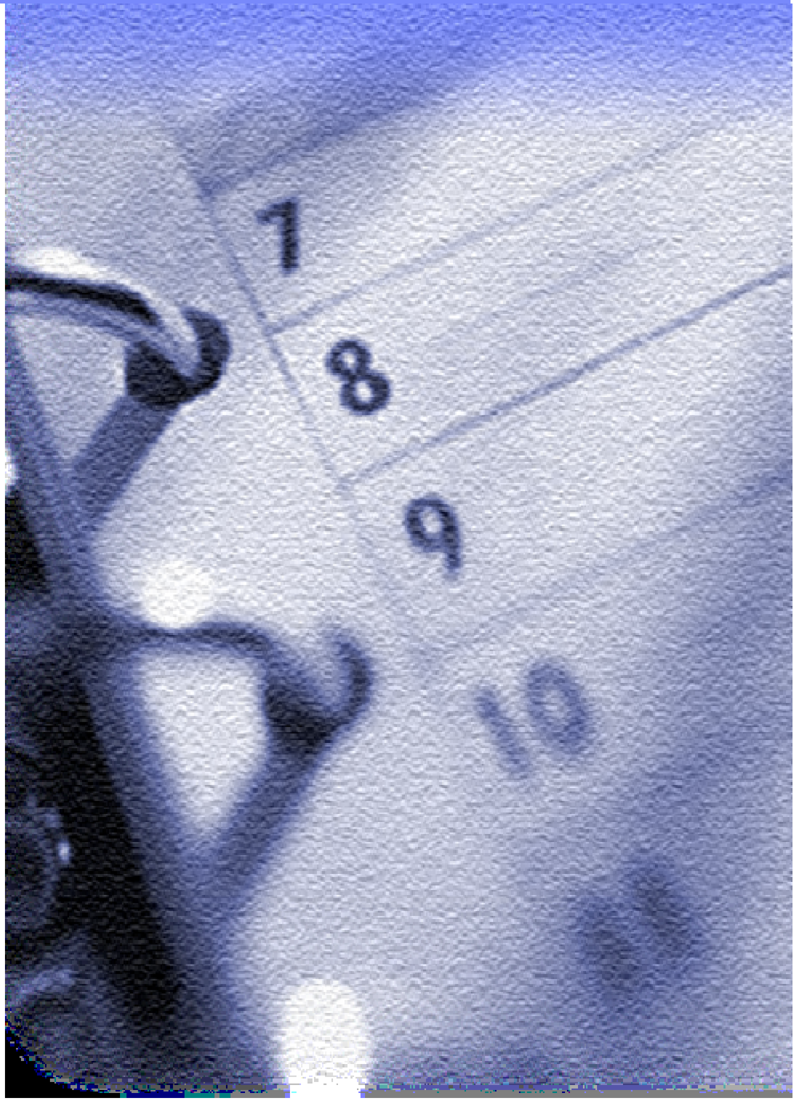
IBM Rational PurifyPlus
Developing fast, reliable code

Agenda



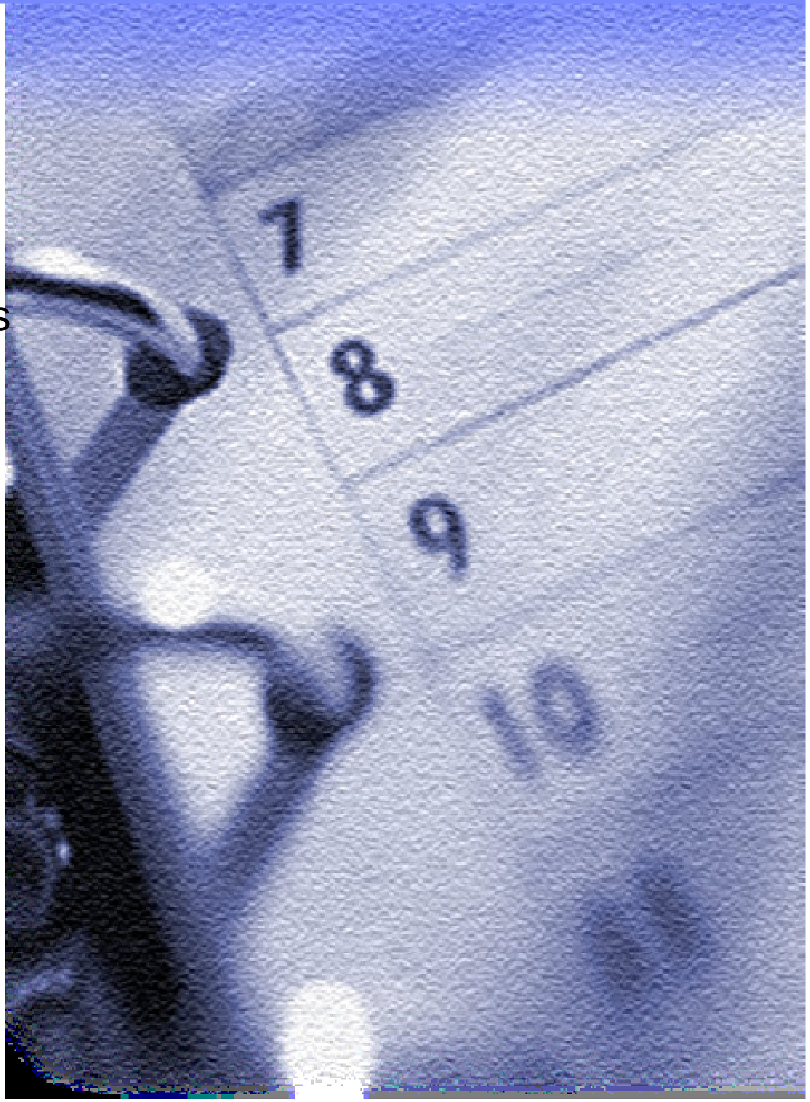
- Challenge: Reliable Software is Critical to Success
- Business Driven Development
- What makes bugs difficult to find?
- Developing fast, reliable code
  - Detect errors at run time
  - Improve software performance
  - Avoid shipping untested code
- Next Steps
- Closing, Q&A and Thanks

Developing fast, reliable code using IBM Rational PurifyPlus

The Challenge: Reliable Software Is Critical To Success

- Your customer expects you to deliver a reliable application
  - Application quality can be a key differentiator
- Building reliable applications is hard!
  - How much time do you spend tracking down memory corruption problems?
  - Does integration always take longer than planned?
  - How much time do your developers spend tracking down “irreproducible” errors?
- Tracking down just one problem could take a developer days or even weeks!

Agenda



- Challenge: Reliable Software is Critical to Success
- Business Driven Development
- What makes bugs difficult to find?
- Developing fast, reliable code
  - Detect errors at run time
  - Improve software performance
  - Avoid shipping untested code
- Next Steps
- Closing, Q&A and Thanks

Imperatives in Today’s World

Innovating
the business to capture
new value.

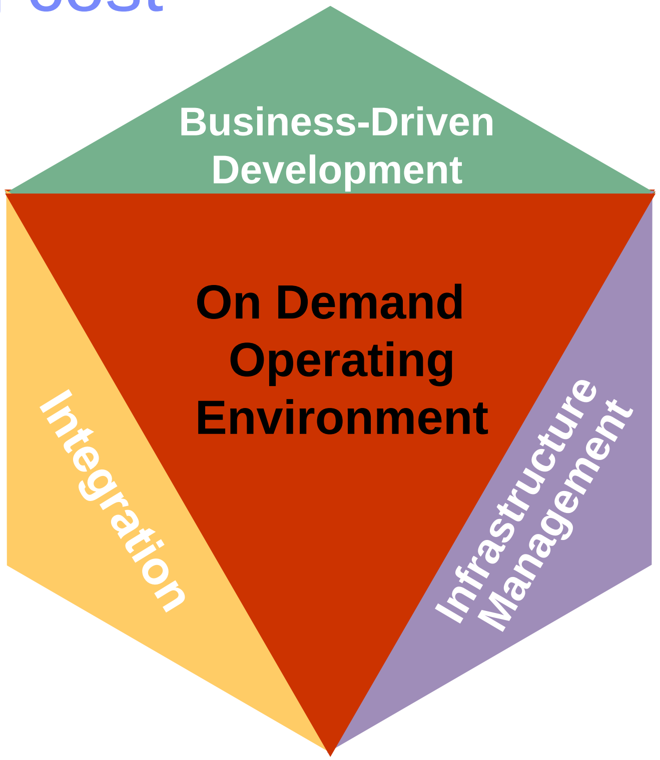
Business

IT

Improving
the productivity
of resources deployed.

IT is the lever to maximize flexibility and responsiveness while containing cost

An on demand operating environment is an integrated infrastructure aligned to business goals and processes in a resilient and secure manner


Business-Driven Development
On Demand Operating Environment
Integration
Infrastructure Management

All designed with an architecture that allows you to manage services as components

Results: Simplification and Optimization of IT to meet the needs of the business responsively

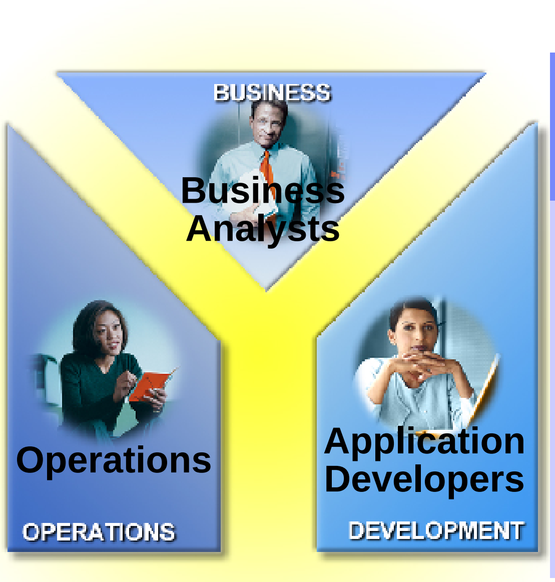
The challenge: Poor visibility, lack of cohesion across business and technology domains

Business View

- Poor visibility and governance over IT investments
- Lack of actionable information
- Blind decision-making

Operations View

- Inadequate service levels
- Inability to rapidly deploy applications
- Complex, multi-tier operating environments



Application Development View

- Overwhelming complexity
- Relentless time-to-market pressure
- Uncontrolled change

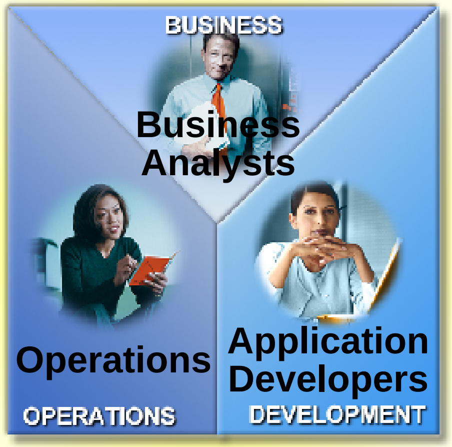
The solution: Breaking down the silos
A shared view of the development lifecycle

Business View

- Clear view of technology ROI
- Top-down and bottom-up visibility into technology projects
- Objective decision-making support

Operations View

- Improved service and quality compliance
- Predictable deployments
- Accelerated diagnosis and repair



Application Development View

- Rapid application development and deployment
- Improved collaboration
- Asset reuse

Transform and simplify software development

Business-driven development

An integrated approach to software development that aligns line-of-business, development and operations teams to improve business performance

Development as a business process

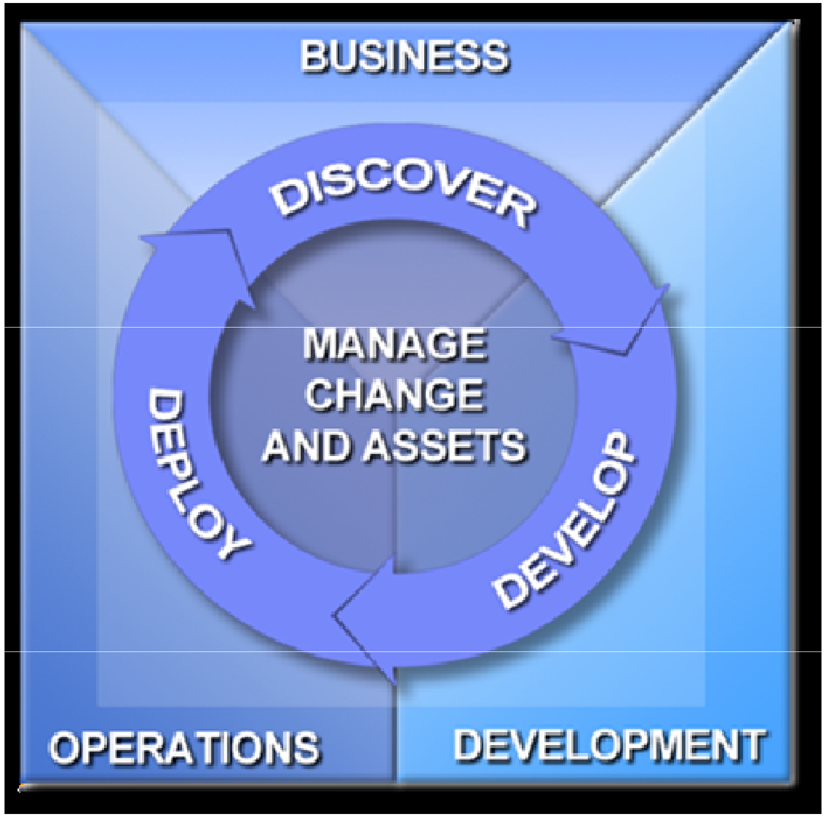
- Align Technology and Business priorities
- Improve efficiency and responsiveness
- Create innovative products

- Higher productivity - 50% + increase in developer productivity
- Improved quality - 80% fewer bugs
- Greater predictability

Software development becomes a driver of competitive advantage

Discover, develop, and deploy assets

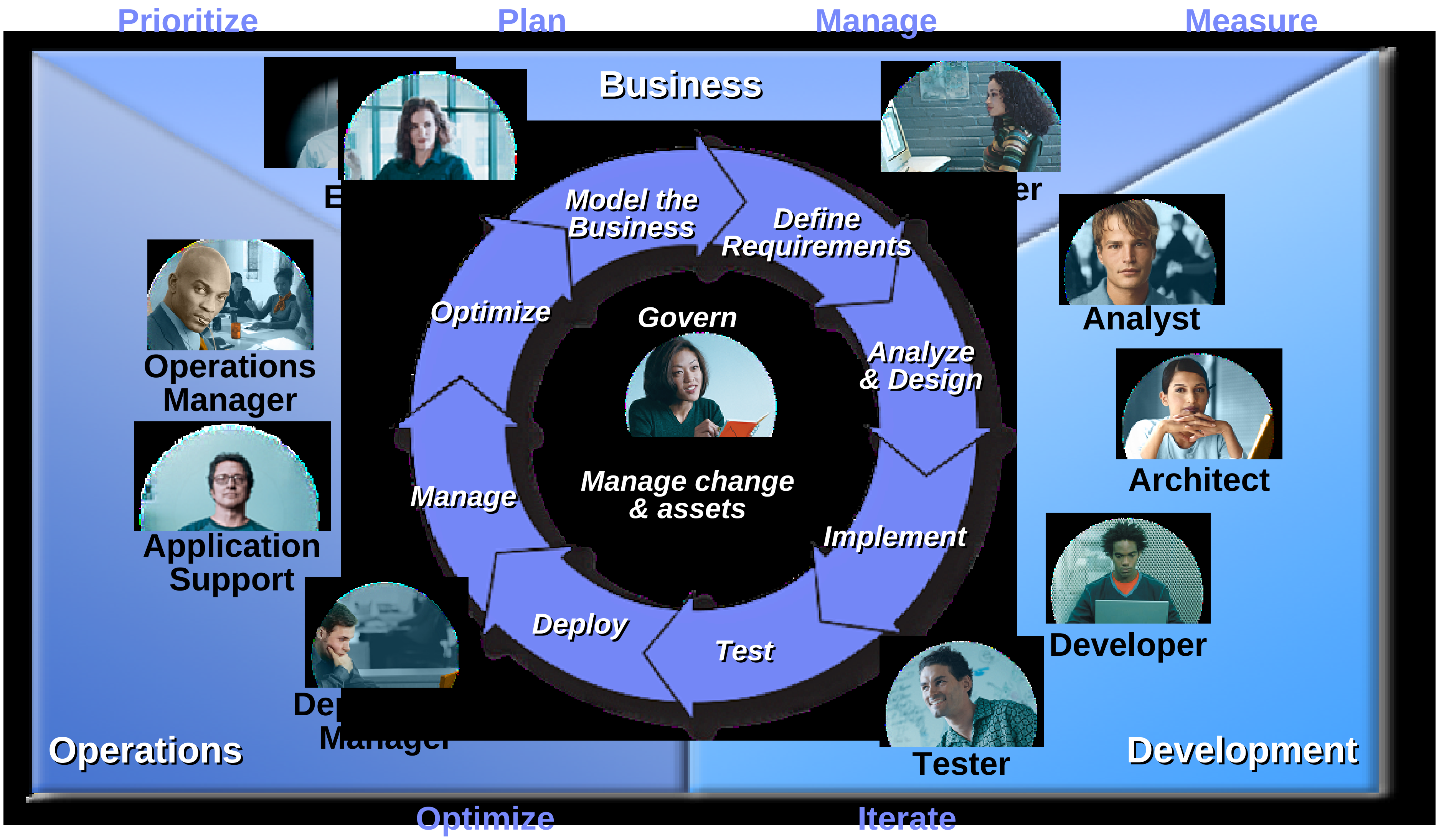
Prioritize Plan Manage Measure



Optimize Iterate

- **Discover business & technology assets**
  - Business priorities
  - Requirements
  - Middleware and software assets
- **Develop at the speed of business**
  - Rapid application development
  - Model-driven architecture
  - Asset-based development
  - Direct-to-middleware productivity
- **Deploy to closed-loop environments**
  - Automated applications deployment
  - Streamlined composite application management
  - Direct-to-operations productivity

The business-driven development lifecycle



Prioritize

Plan

Manage

Measure

Business

E

er

Model the Business

Define Requirements

Analyze & Design

Implement

Test

Deploy

Manage

Optimize

Govern

Manage change & assets

Operations Manager

Application Support

De

Manager

Analyst

Architect

Developer

Tester

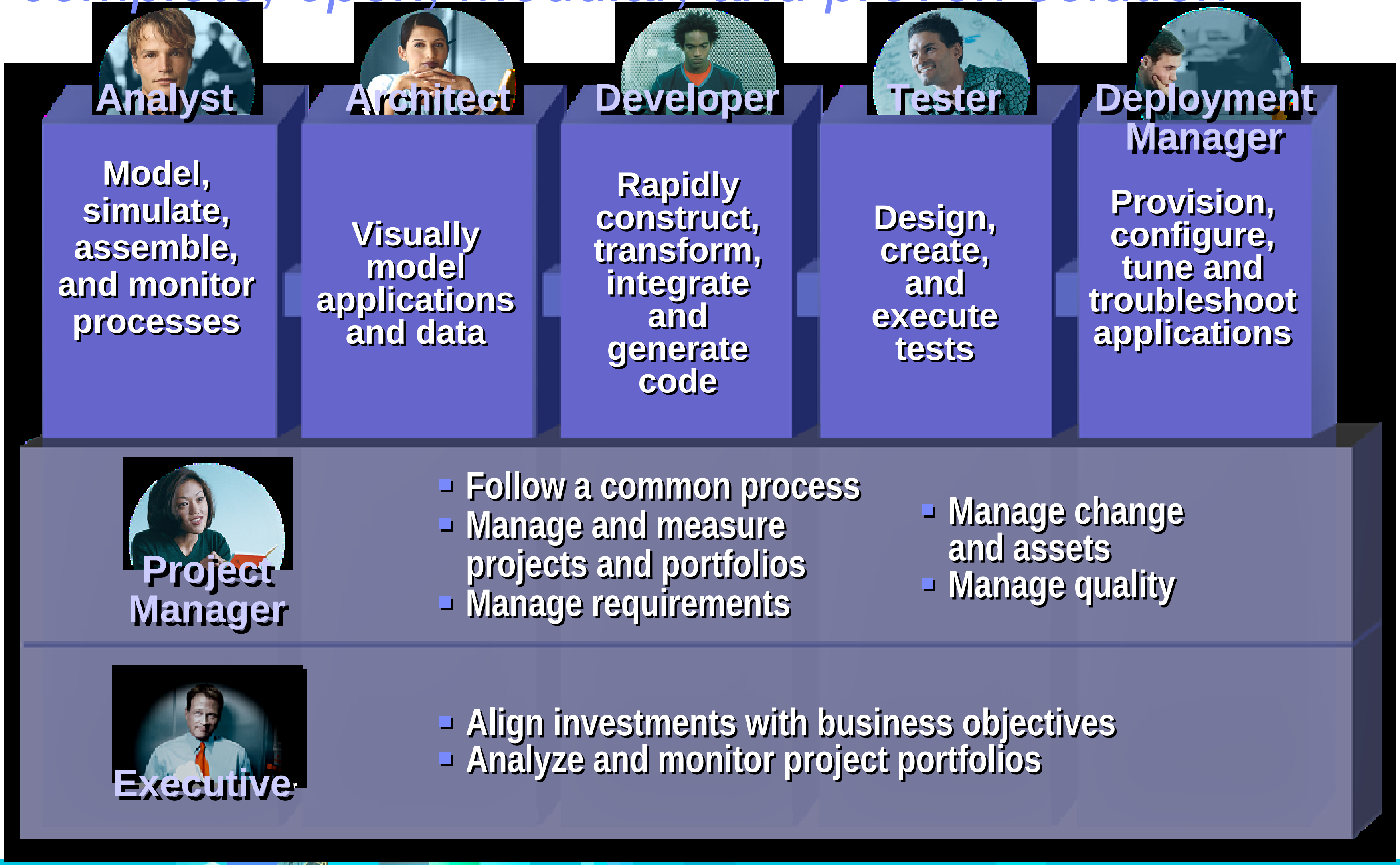
Operations

Development

Optimize

Iterate

The IBM Rational Software Development Platform
A complete, open, modular, and proven solution



Analyst

Model, simulate, assemble, and monitor processes

Architect

Visually model applications and data

Developer

Rapidly construct, transform, integrate and generate code

Tester

Design, create, and execute tests

Deployment Manager

Provision, configure, tune and troubleshoot applications

Project Manager

- Follow a common process
- Manage and measure projects and portfolios
- Manage requirements
- Manage change and assets
- Manage quality

Executive

- Align investments with business objectives
- Analyze and monitor project portfolios

IBM software quality offerings

Benefits

- Ensure reliability, functionality, scalability
- Accelerate test cycles
- Support multiple skill levels
- Share responsibility for quality across the team

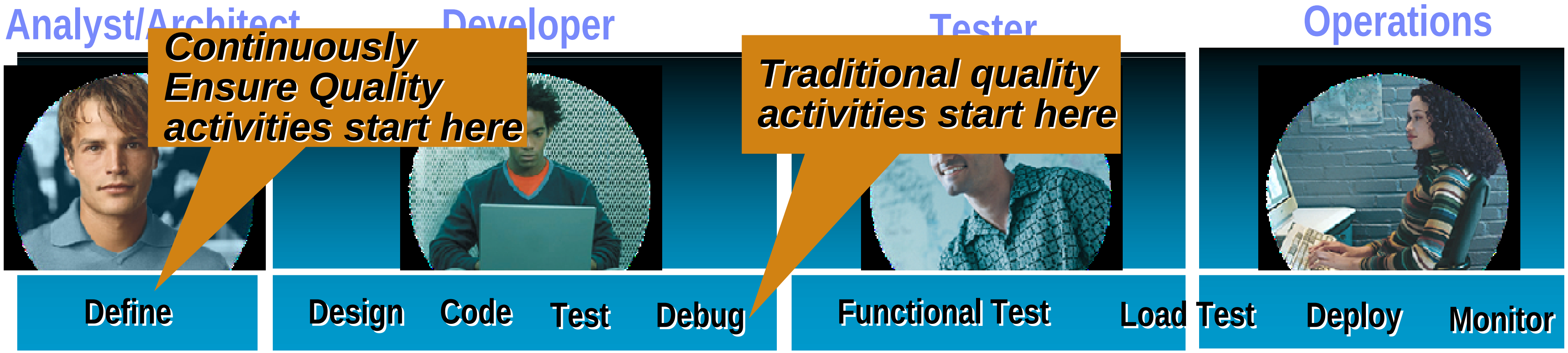
Capabilities

- Runtime analysis
- Component, system, and performance testing
- Distributed test execution
- Test planning, reporting and analysis

Key Products	**Business Analyst**	**Tester**	**Developer**
IBM Rational Manual Tester	✓	✓	✓
IBM Rational Functional Tester		✓	✓
IBM Rational Performance Tester		✓	✓
IBM Rational Robot		✓	
IBM Rational PurifyPlus		✓	✓
IBM Rational Test RealTime			✓
IBM Rational TestManager	✓	✓	✓

Continuously Ensure Quality

Uniting innovative solutions and best practices to prevent, detect, diagnose and remove defects all across the software application development and deployment lifecycle



- Achieved through quality-focused infrastructure, models, tools, processes and measurements
- Supported by all team members, not just testers
- Goes beyond reducing defect counts to improving overall software fitness

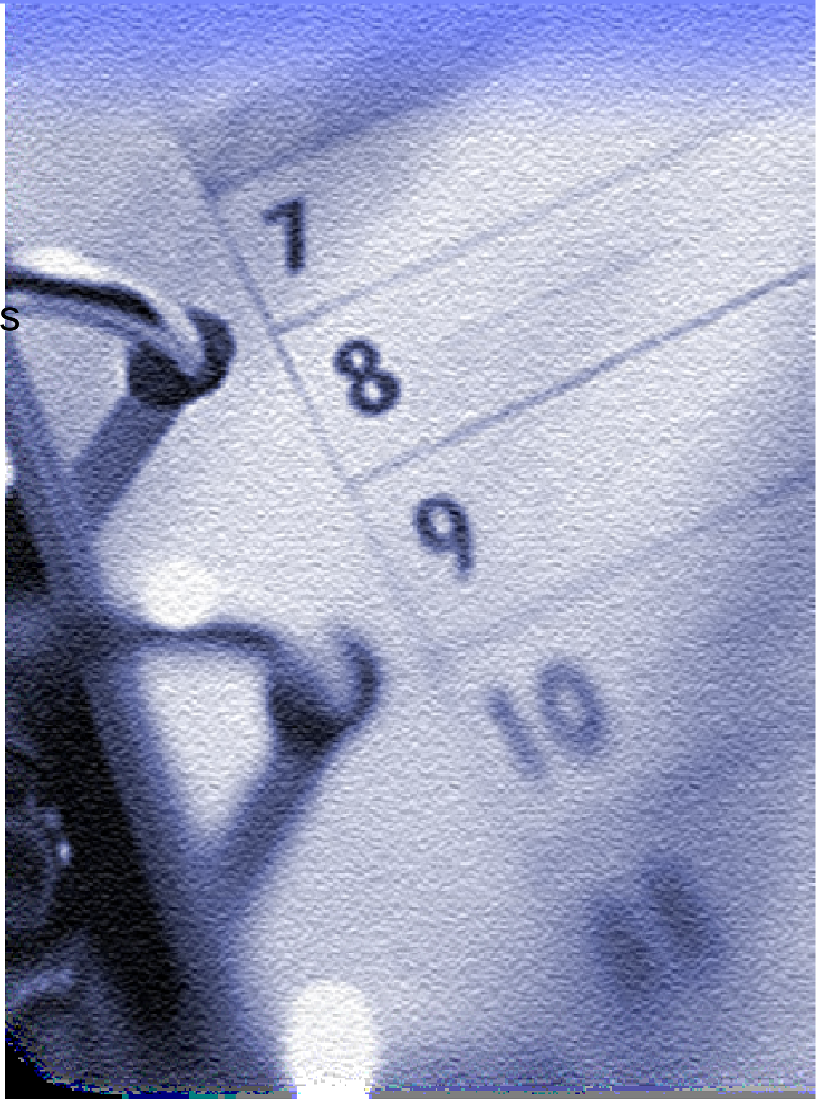
Continuously Ensure Quality: Developing Fast, Reliable Code

- To develop fast, reliable code, you need a too
  - Automatically pinpoints hard-to-find bugs
  - Highlights performance bottlenecks
  - Keeps you from shipping untested code
- You need a multi-platform and multi-language tool for Unix, Windows, Linux, Java, .NET, VB6, C/C++, and more
- That tool is IBM Rational PurifyPlus!
- IBM Rational PurifyPlus combines three industry leaders into one box:
  - IBM Rational Purify
  - IBM Rational Quantify
  - IBM Rational PureCoverage

The x-ray for software

Instantly become more productive

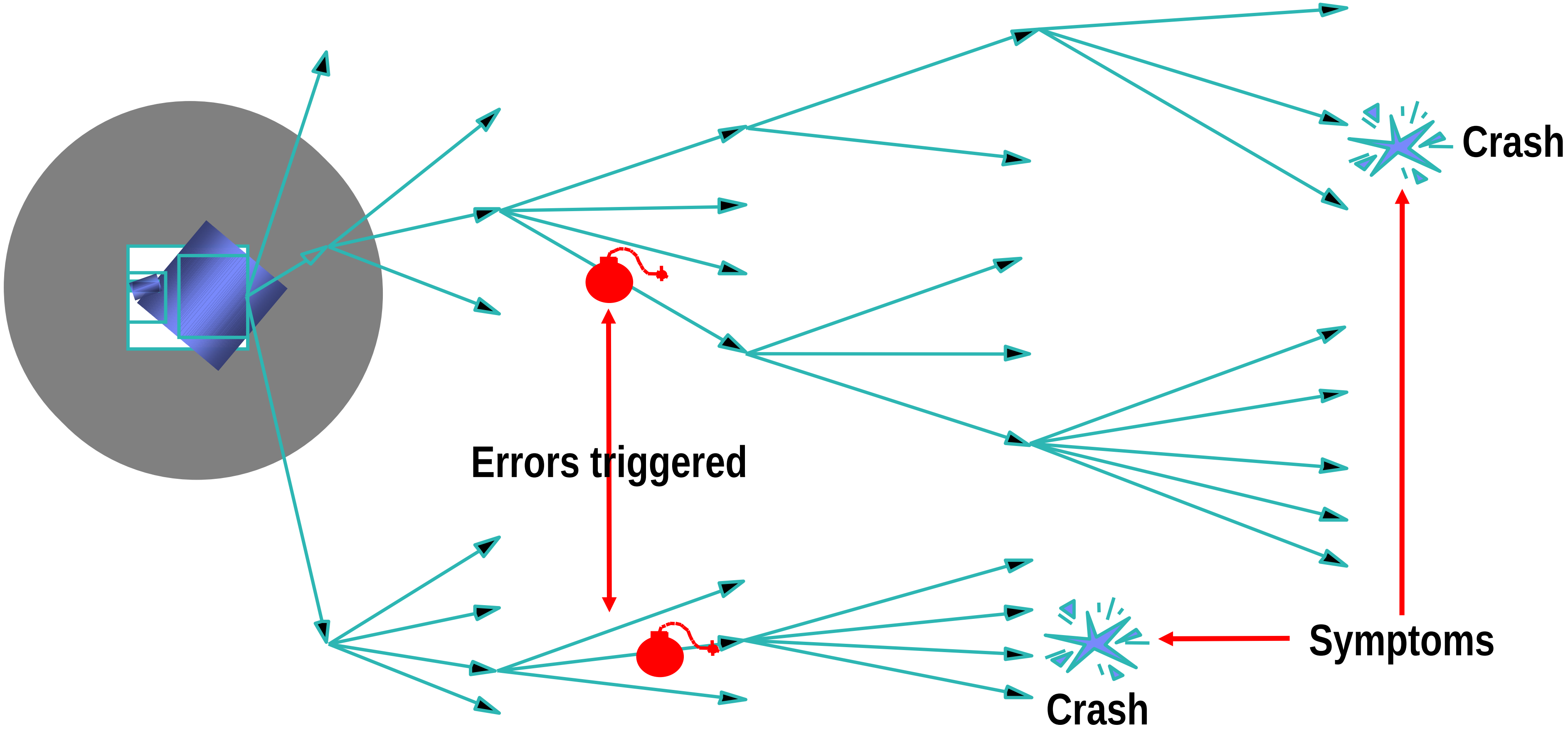
Agenda



- Challenge: Reliable Software is Critical to Success
- Business Driven Development
- What makes bugs difficult to find?
- Developing fast, reliable code
  - Detect errors at run time
  - Improve software performance
  - Avoid shipping untested code
- Next Steps
- Closing, Q&A and Thanks

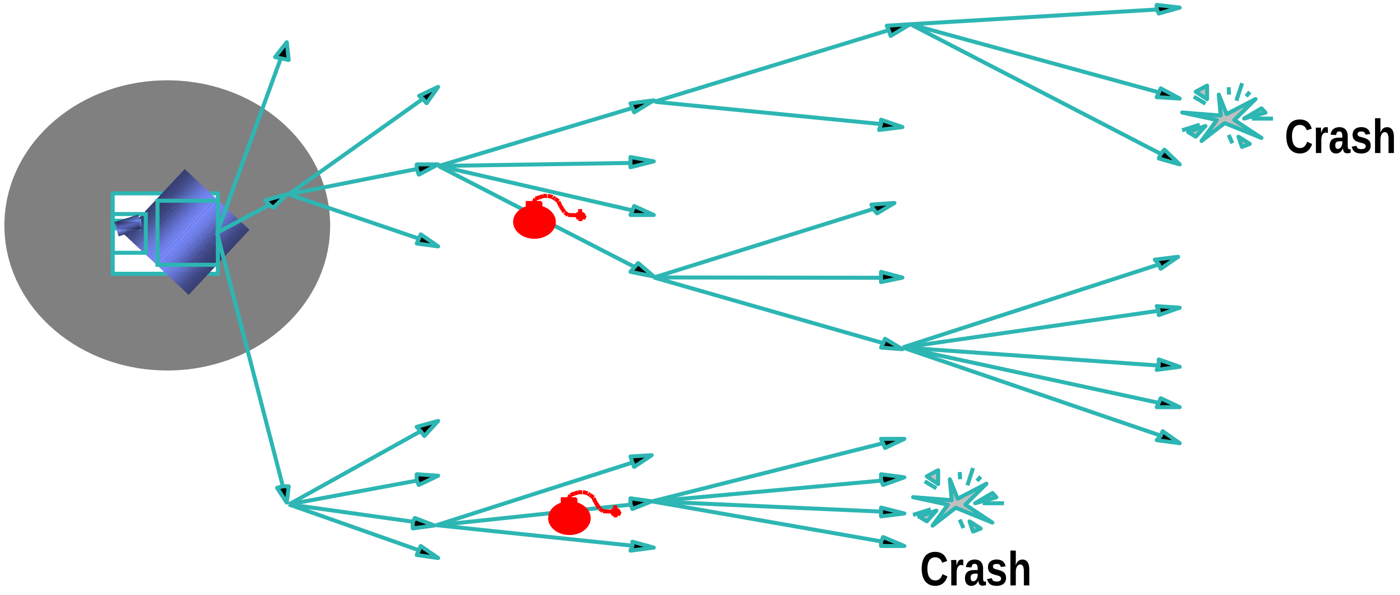
Developing fast, reliable code using IBM Rational PurifyPlus

The Problem



What Makes Bugs Difficult to Find?

- Many memory-related bugs have no immediate visible symptoms
- Often the symptoms are difficult, if not impossible, to reproduce
- Manually tracing symptoms back to problems in the source code can be a daunting task



Why Is My Program So Slow?

- Do you deliver slow code that’s “good enough?”
- Is it too difficult to find performance problems?
- Can you get repeatable, useful performance data?
- Can you quantify the effect of performance fixes?
- Does your performance profiling data help you manage your development project?

What about Predictability at runtime?

Code Coverage Challenge

- Can you guarantee the code that you deliver is reliable?
- How complete is your test suite?
- Are you sure that you have tested the entire application?
- Can you understand the summary of several tests?

How Reliable is Untested Code?

Agenda

- Challenge: Reliable Software is Critical to Success
- Business Driven Development
- What makes bugs difficult to find?
- Developing fast, reliable code
  - Detect errors at run time
  - Improve software performance
  - Avoid shipping untested code
- Next Steps
- Closing, Q&A and Thanks

Developing fast, reliable code using IBM Rational PurifyPlus

Classic Uninitialized Memory Problem With C++

- Does the following missile launching code work?

```
typedef enum LaunchCodeType {
                LAUNCH_STANDBY,
                LAUNCH_OK,
} LaunchCodeType;

LaunchCodeType launch_code =
                LAUNCH_STANDBY;

Void standby()
{
                LaunchCodeType launch;
                while (1) {
                                if (launch == LAUNCH_OK){
                                      launch_missile();
                                }
                                sleep(1000);
                }
}
```

Classic Uninitialized Memory Problem With C++

- Does the following missile launching code work?

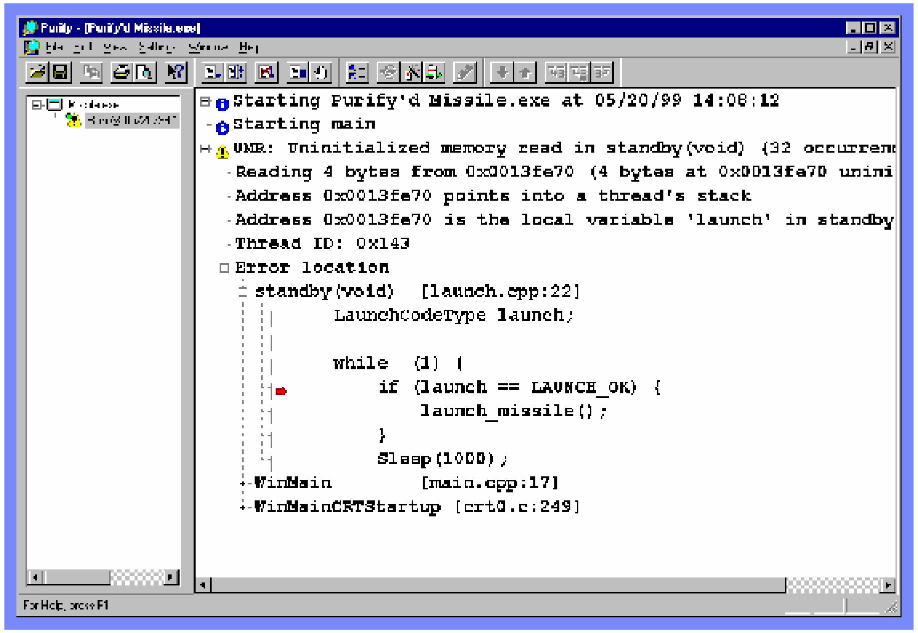
```
typedef enum LaunchCodeType {
                LAUNCH_STANDBY,
                LAUNCH_OK,
} LaunchCodeType;

LaunchCodeType launch_code =
                LAUNCH_STANDBY;

Void standby()
{
                LaunchCodeType launch;
                while (1) {
                                if (launch == LAUNCH_OK){
                                       launch_missile();
                                }
                                sleep(1000);
                }
}
```

Uninitialized Memory Detection

- Difficult to identify
- Diagnosis methods
  - The hard way – identify the problem, track down the uninitialized variable using a debugger
  - The easy way – use automated run-time error checking tool like Rational PurifyPlus



```
Starting Purify'd Missile.exe at 05/20/99 14:08:12
 Starting main
UMR: Uninitialized memory read in standby(void) (32 occurren
  Reading 4 bytes from 0x0013fe70 (4 bytes at 0x0013fe70 unini
  Address 0x0013fe70 points into a thread's stack
  Address 0x0013fe70 is the local variable 'launch' in standby
  Thread ID: 0x143
  Error location
    standby(void)   [launch.cpp:22]
            LaunchCodeType launch;

            while  (1) {
                if (launch == LAUNCH_OK) {
                    launch_missile();
                }
                Sleep(1000);
    WinMain         [main.cpp:17]
    WinMainCRTStartup [crt0.c:249]
```

Detecting Memory Corruption: Rational Purify

- ✓ Automatically identifies the problem while the application is running!
- ✓ Finds errors in third-party code, even without source
  - Bugs in libraries that you deliver are still bugs in your application!
- ✓ No recompile required
  - Much faster and easier to use
  - Incorporate into your existing project quickly and easily
- ✓ Test what you are delivering
  - No need to maintain multiple source baselines
  - QE knows that they are testing what you are delivering

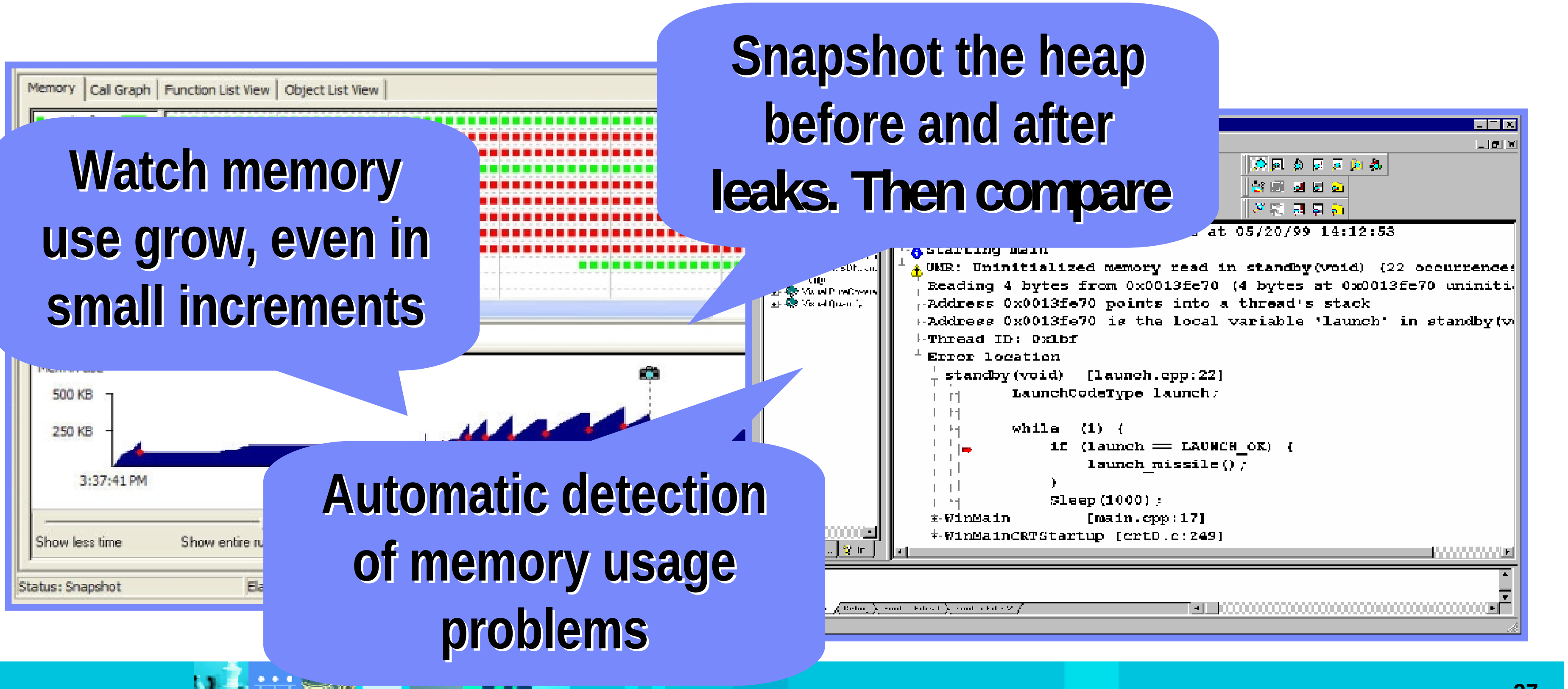
Memory “Leaks” In Garbage Collected Environments

- Common myth – memory leaks cannot happen in garbage collected environments, such as Java or .NET
  - No direct “twiddling” with pointers
  - Garbage collection
- The reality
  - Most common C/C++ ways to trash memory are not possible in Java and the .NET languages
  - But “leaks” can still occur

IBM Rational PurifyPlus detects Java and .NET “leaks”

Finding Memory Usage Errors Using Rational Purify

- ✓ Automatically identify memory usage problems
- ✓ Identify Memory Leak Detection problems in C/C++
- ✓ Identify Java and .NET memory usage errors with memory profiling
- ✓ Identify memory leaks and usage problems in third-party code – even without source



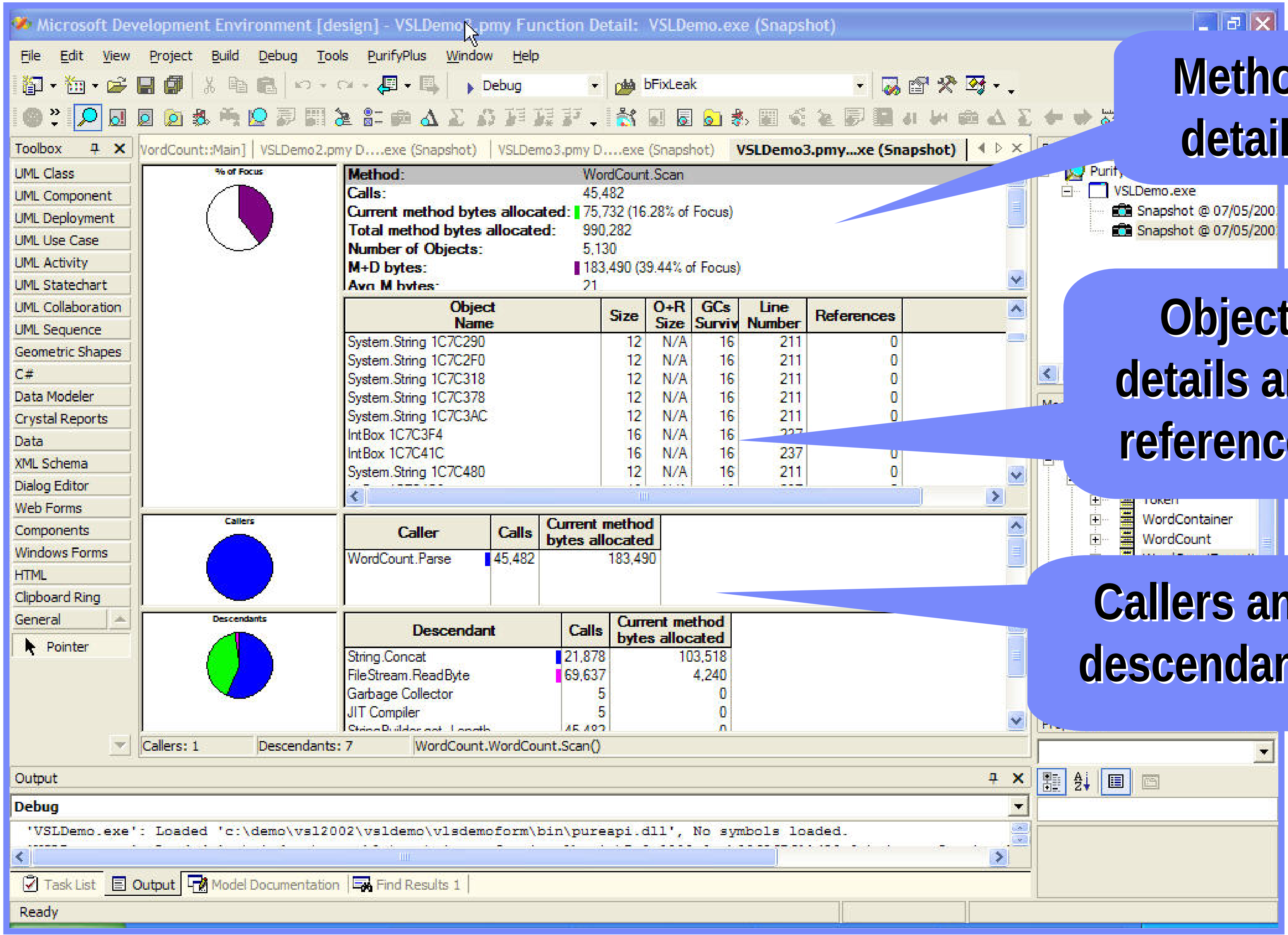
Watch memory use grow, even in small increments

Snapshot the heap before and after leaks. Then compare

Automatic detection of memory usage problems

```
at 05/20/99 14:12:53
Starting main
UMR: Uninitialized memory read in standby(void) {22 occurrences
Reading 4 bytes from 0x0013fe70 (4 bytes at 0x0013fe70 uniniti
Address 0x0013fe70 points into a thread's stack
Address 0x0013fe70 is the local variable 'launch' in standby(v
Thread ID: 0x1bf
Error location
  standby(void)   [launch.cpp:22]
          LaunchCodeType launch;

          while  (1) {
              if (launch == LAUNCH_OK) {
                  launch_missile();
              }
              Sleep(1000);
  WinMain            [main.cpp:17]
  WinMainCRTStartup [crtD.c:249]
```



Method: WordCount.Scan
Calls: 45,482
Current method bytes allocated: 75,732 (16.28% of Focus)
Total method bytes allocated: 990,282
Number of Objects: 5,130
M+D bytes: 183,490 (39.44% of Focus)
Avg M bytes: 21

Object Name	Size	O+R Size	GCs Surviv	Line Number	References
System.String 1C7C290	12	N/A	16	211	0
System.String 1C7C2F0	12	N/A	16	211	0
System.String 1C7C318	12	N/A	16	211	0
System.String 1C7C378	12	N/A	16	211	0
System.String 1C7C3AC	12	N/A	16	211	0
IntBox 1C7C3F4	16	N/A	16	237	[illegible]
IntBox 1C7C41C	16	N/A	16	237	0
System.String 1C7C480	12	N/A	16	211	0

Caller	Calls	Current method bytes allocated
WordCount.Parse	45,482	183,490

Descendant	Calls	Current method bytes allocated
String.Concat	21,878	103,518
FileStream.ReadByte	69,637	4,240
Garbage Collector	5	0
JIT Compiler	5	0

Callers: 1 Descendants: 7 WordCount.WordCount.Scan()

Debug

```
'VSLDemo.exe': Loaded 'c:\demo\vsl2002\vsldemo\vlsdemoform\bin\pureapi.dll', No symbols loaded.
```

Method details

Object details and references

Callers and descendants

Makes You More Productive – IBM Rational Purify

- ✓ Improves schedule predictability
  - Identifies, in seconds, what would take a developer days or weeks
  - Your developers can spend less time debugging
- ✓ Fastest and Easiest tool to use within the PurifyPlus package
- ✓ Debugs the entire application
  - Including third-party libraries

Automatically pinpoints hard to find bugs

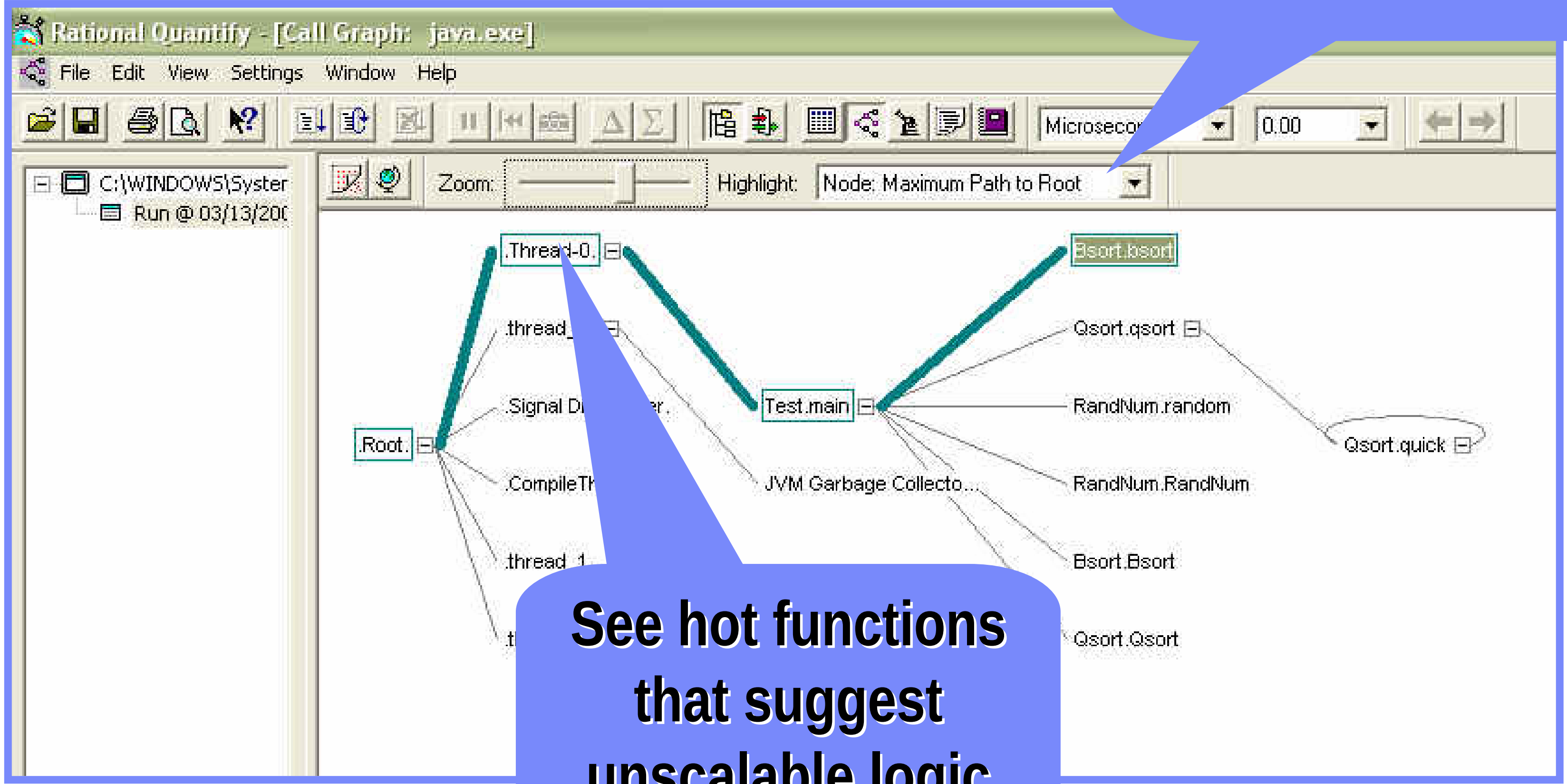
Why Is Improving Performance Difficult?

- Fixing bugs leaves less time for performance tuning
- Data collection is sensitive on other processes running on the same machine
- Timing information can be elusive

Voice of the Customer: Clair Cates - SAS
"The developers, and myself included, feel like Purify and Quantify are worth their weight in gold."

Performance tuning is often left to the end of the development lifecycle or ignored altogether

Highlight Performance Bottlenecks

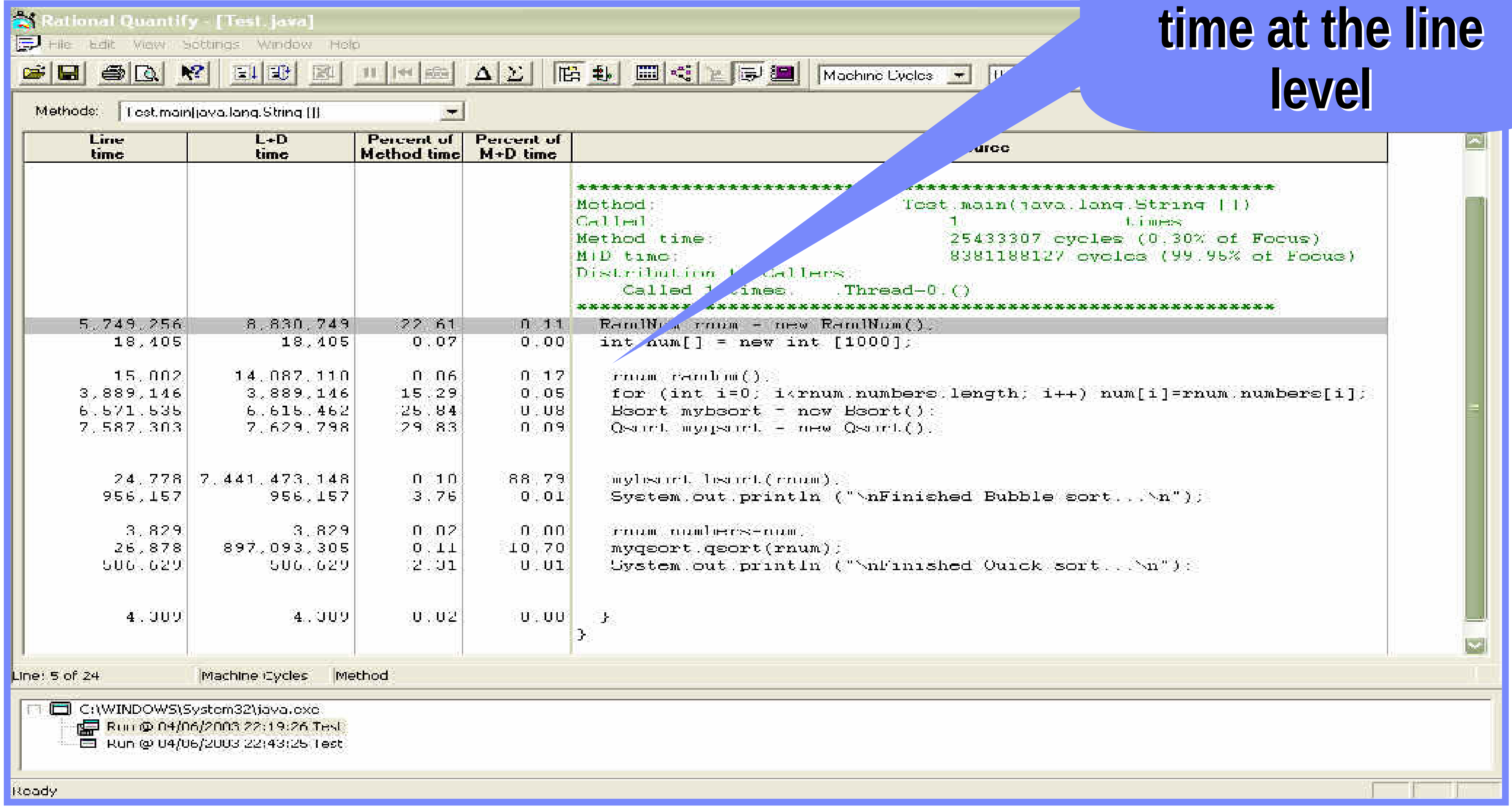


Highlight: Node: Maximum Path to Root

Visualize runtime algorithms for bottlenecks

See hot functions that suggest unscalable logic

Analyze Performance Line Per Line


Rational Quantify - [Test.java]
File Edit View Settings Window Help
Machine Cycles

Methods: Test.main(java.lang.String [])

```
Line        L+D            Percent of   Percent of   Source
time        time           Method time  M+D time

                                                     ***********************************************************
                                                     Method:                  Test.main(java.lang.String [])
                                                     Called:                  1                 times
                                                     Method time:             25433307 cycles (0.30% of Focus)
                                                     M+D time:                8381188127 cycles (99.95% of Focus)
                                                     Distribution to callers:
                                                         Called 1 times.    .Thread-0.()
                                                     ***********************************************************
5,749,256   8,830,749      22.61        0.11           RandNum rnum = new RandNum();
   18,405      18,405       0.07        0.00           int num[] = new int [1000];

   15,002   14,087,110      0.06        0.17            rnum.random();
3,889,146    3,889,146     15.29        0.05            for (int i=0; i<rnum.numbers.length; i++) num[i]=rnum.numbers[i];
6,571,535    6,615,462     25.84        0.08            Bsort mybsort = new Bsort();
7,587,303    7,629,798     29.83        0.09            Qsort myqsort = new Qsort();


   24,778 7,441,473,148     0.10       88.79            mybsort.bsort(rnum);
  956,157      956,157      3.76        0.01            System.out.println ("\nFinished Bubble sort...\n");

    3,829        3,829      0.02        0.00            rnum.numbers=num;
   26,878  897,093,305      0.11       10.70            myqsort.qsort(rnum);
  506,629      506,629      2.01        0.01            System.out.println ("\nFinished Quick sort...\n");


    4,009        4,009      0.02        0.00           }
                                                     }
```

Line: 5 of 24 | Machine Cycles | Method

```
C:\WINDOWS\System32\java.exe
    Run @ 04/06/2003 22:19:26 Test
    Run @ 04/06/2003 22:43:25 Test
```

Ready

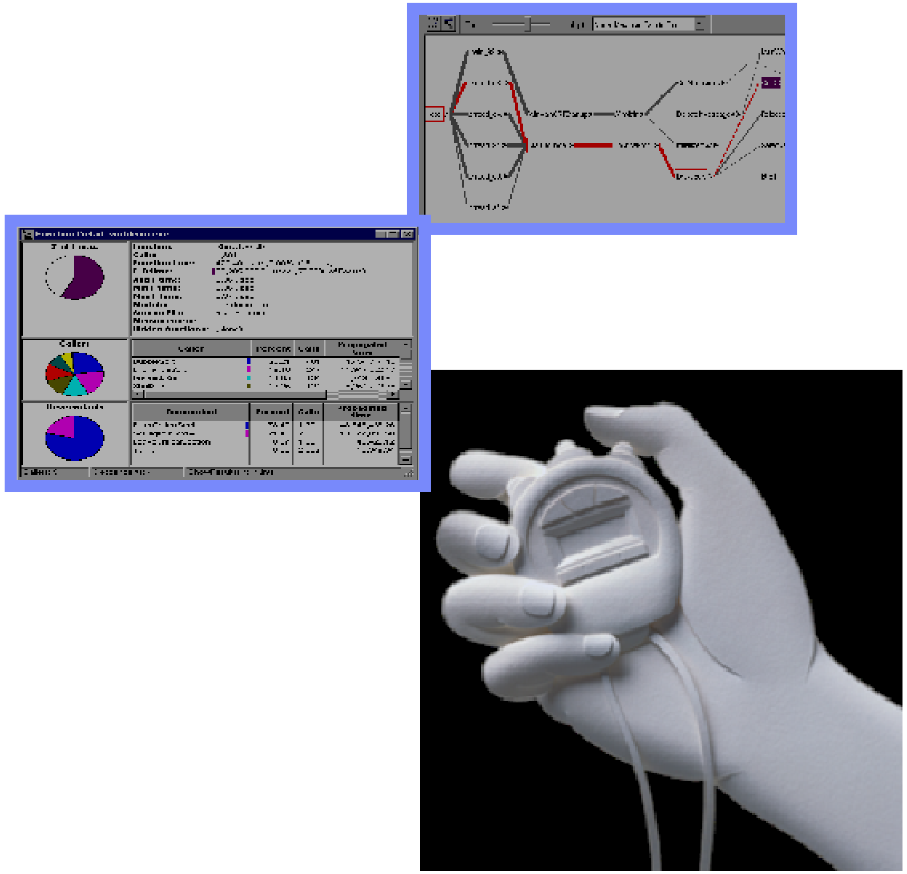
Analyze the distribution of time at the line level

Improve Code Performance With Rational Quantify

- ✓ Leading product that provides repeatable performance data
  - Build better quality C/C++, Java, VB6, and .NET software
  - Get your team solving the correct problems
- ✓ Fast and easy to use
  - Provides concise, graphical views of performance data
  - “River of Time” – a straightforward way to view and understand performance profiling data
  - Easily integrates into your nightly builds
- ✓ Collects performance data from code you intend to deliver
  - No need to maintain multiple source baselines

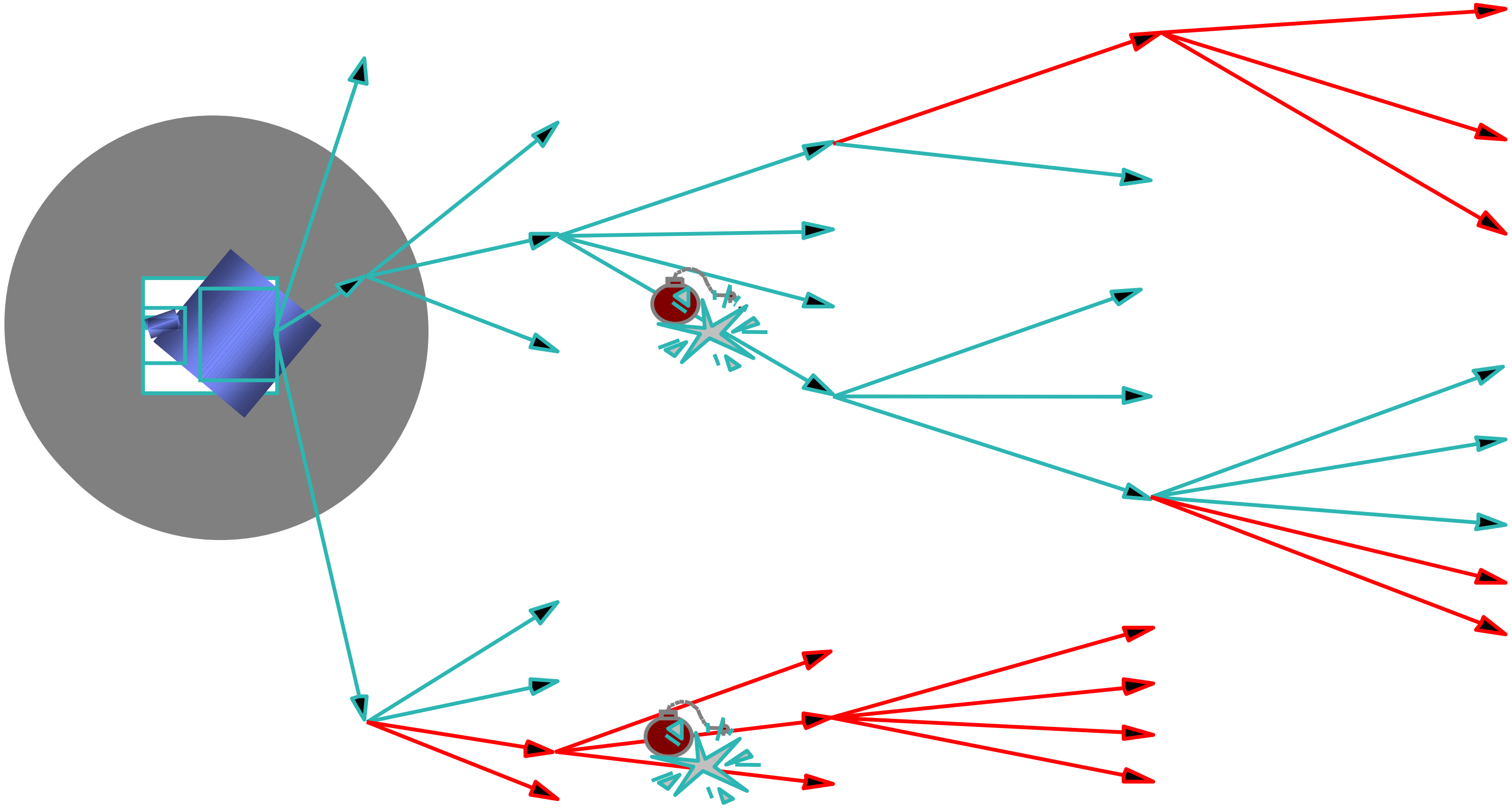
Key To Improving Performance – IBM Rational Quantify

- Eliminates guesswork out of code performance
- Incorporates easily into new or existing project
- Works on entire application
  - Including third party libraries
- Can be used for selected modules only



Highlights Performance Bottlenecks

What Have I Missed?

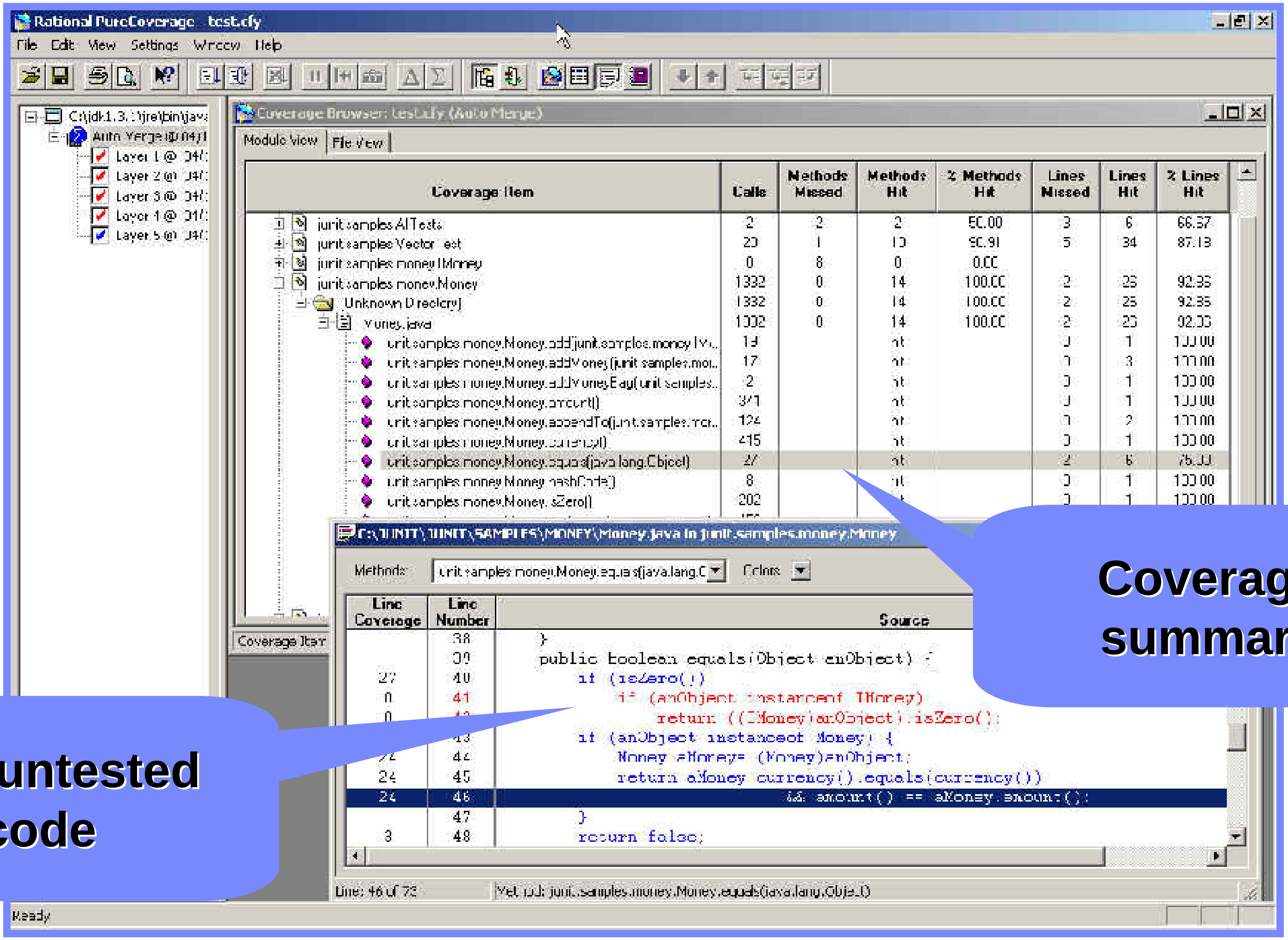


🟥 **Unexecuted** 🟩 **Executed**

Ensures You’re Covered – IBM Rational PureCoverage

- Untested code is unreliable code
- Integrated with testing tools
  - Fits within your existing processes
  - Multiple runs are automatically merged
- Test what you intend to deliver
- Works on object code, byte code or executables
  - No source required
  - Recompiles or separate source baselines not necessary

Code Coverage with Rational PureCoverage



Coverage Item	Calls	Methods Missed	Methods Hit	% Methods Hit	Lines Missed	Lines Hit	% Lines Hit
junit.samples.AllTests	2	2	2	50.00	3	6	66.67
junit.samples.VectorTest	20	1	10	90.91	5	34	87.18
junit.samples.money.IMoney	0	8	0	0.00			
junit.samples.money.Money	1332	0	14	100.00	2	25	92.35
[Unknown Directory]	1332	0	14	100.00	2	25	92.35
Money.java	1332	0	14	100.00	2	25	92.35
junit.samples.money.Money.add(junit.samples.money.IM...	19		n/a		0	1	100.00
junit.samples.money.Money.addMoney(junit.samples.mo...	17		n/a		0	3	100.00
junit.samples.money.Money.addMoneyBag(junit.samples...	2		n/a		0	1	100.00
junit.samples.money.Money.amount()	371		n/a		0	1	100.00
junit.samples.money.Money.appendTo(junit.samples.mon...	124		n/a		0	2	100.00
junit.samples.money.Money.currency()	415		n/a		0	1	100.00
junit.samples.money.Money.equals(java.lang.Object)	27		n/a		2	6	75.00
junit.samples.money.Money.hashCode()	8		n/a		0	1	100.00
junit.samples.money.Money.isZero()	202		n/a		0	1	100.00

C:\JUNIT\JUNIT\SAMPLES\MONEY\Money.java in junit.samples.money.Money

Methods: junit.samples.money.Money.equals(java.lang.C

Colors

```
Line      Line
Coverage  Number  Source
          38          }
          39          public boolean equals(Object anObject) {
27        40              if (isZero())
0         41                  if (anObject instanceof IMoney)
0                                 return ((IMoney)anObject).isZero();
          43              if (anObject instanceof Money) {
24        44                  Money aMoney= (Money)anObject;
24        45                  return aMoney.currency().equals(currency())
24        46                                          && amount() == aMoney.amount();
          47              }
3         48              return false;
```

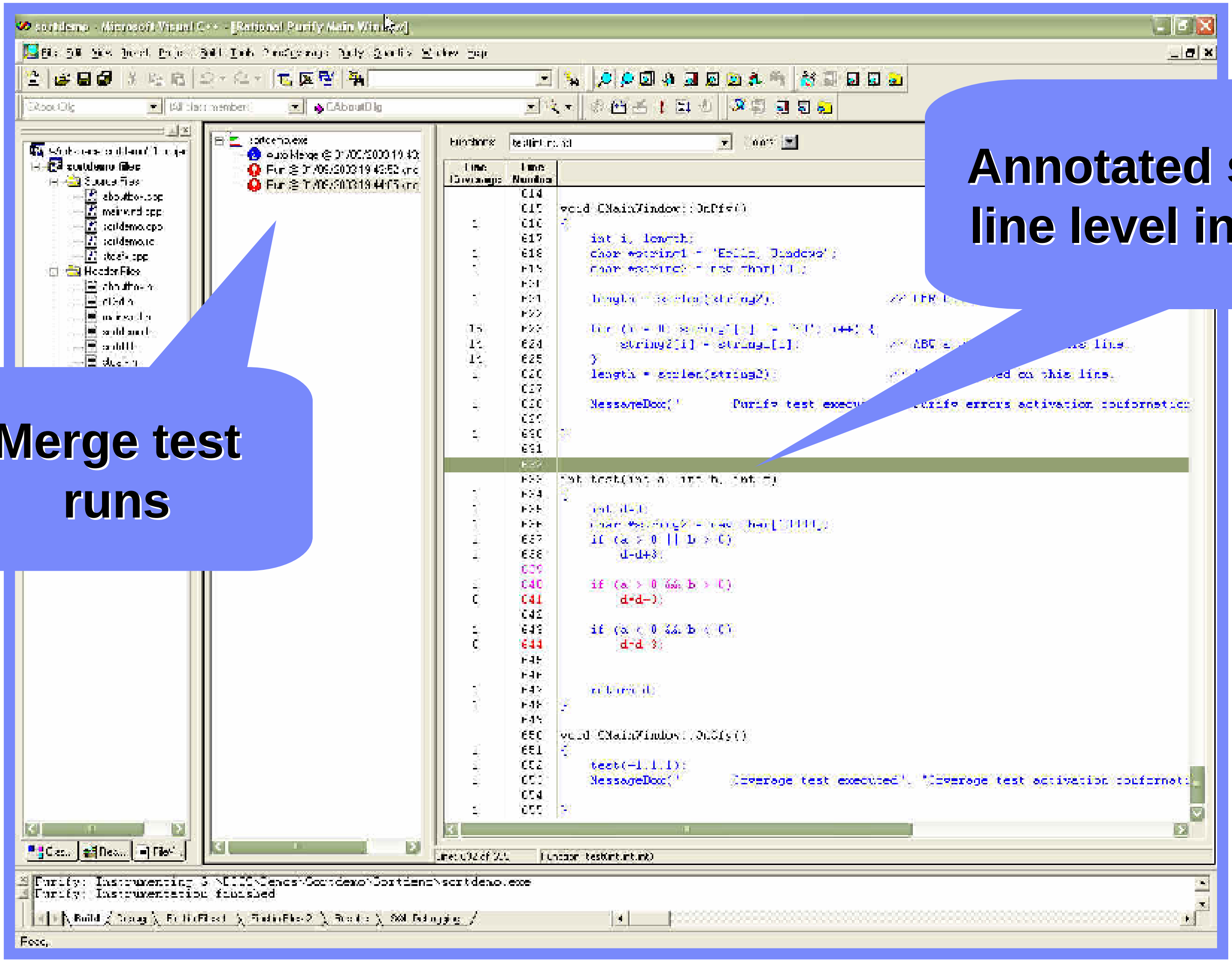
Line: 46 of 73

Method: junit.samples.money.Money.equals(java.lang.Object)

Coverage summary

Find untested code

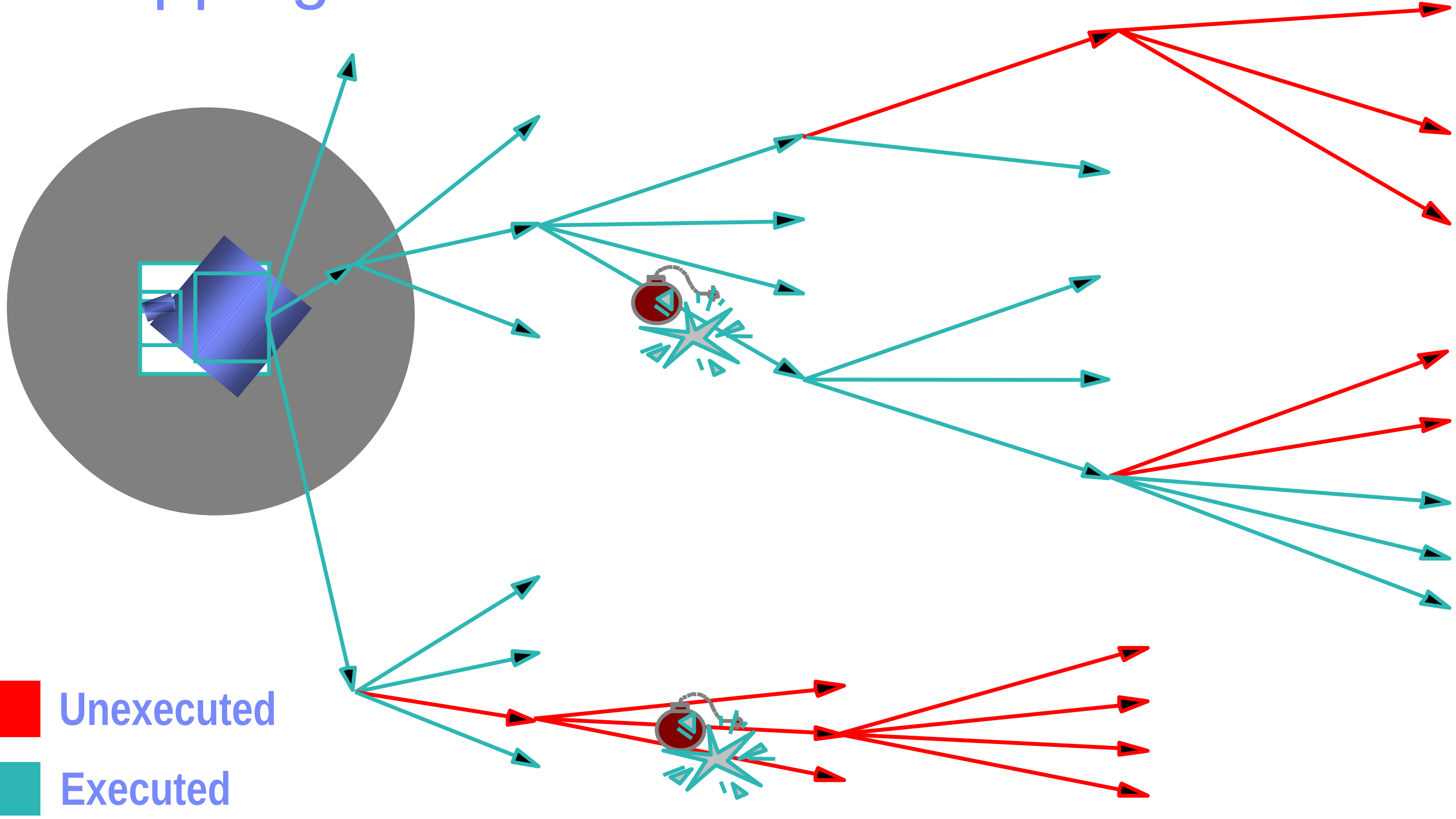
Code Coverage with Rational PureCoverage



Annotated source for line level information

Merge test runs

Avoid Shipping Untested Code



🟥 Unexecuted

🟩 Executed

Avoid shipping untested code.
IBM Rational PurifyPlus finds what you may have missed.

IBM Rational PureCoverage

- ✓ Helps developers deliver higher quality code
  - You know the unit testing is complete
  - Helps identify “dead code” or unexercised paths in legacy code
- ✓ Helps improve your QA process
  - Easily see what has and has not been tested
  - Validate the quality and completeness of your test suite
  - Easy to learn and easy to use out of the box
- ✓ With Rational PureCoverage, you know that you are covered!

Avoid shipping untested code

IBM Rational PurifyPlus

- Key components of IBM Rational PurifyPlus
  - Provides run-time data collection and analysis capabilities
  - Keys for proactive debugging
- Supports Unix, Windows, Java, .NET and Linux environments

Linux	**Windows**	**Solaris**	**SGI-IRIX**	**HP-UX**	**AIX**
C/C++	*C/C++*	*C/C++*	*C/C++*	*C/C++*	*C/C++*
RedHat	*VB6*	*Java*	*Full 64bit*	*Full 64bit*	
	.NET	*Full 64bit*			
	Java				

SGI-IRIX:
- **Quantify***
- **PureCoverage***

AIX:
- **Quantify***
- **PureCoverage***

***Not Available Yet**

IBM Rational PurifyPlus Offerings

Windows – Authorized User

- IBM Rational Purify for Windows
  - Memory profiling and error detection
- IBM Rational PurifyPlus for Windows
  - Memory profiling and error detection
  - Application Performance Analysis
  - Code coverage analysis

Linux and UNIX – Authorized User

- IBM Rational Purify for Linux and UNIX
  - Memory profiling and error detection
- IBM Rational PurifyPlus for Linux and UNIX
  - Memory profiling and error detection
  - Application Performance Analysis
  - Code coverage analysis

Multi-Platform - Floating

- IBM Rational PurifyPlus Enterprise Edition
  - Memory profiling and error detection
  - Application Performance Analysis
  - Code coverage analysis