Klocwork

The Proven Leader. Software Security. Software Quality.

Software Modernization as a Path to Secure Software
- Modernization Checklist -

SD Times 2005 100

Tuesday, October 18, 2005

Klocwork Inc.
35 Corporate Drive
Burlington, MA 01803
1.866.556.2967
www.klocwork.com

Agenda

- The Facts about Software Security and Software Quality
- Review of Common Software Vulnerabilities
- How it Happens: Software Companies Victims of their Success
- Key Considerations for Managing Software Security
- Checklist for Success
- Options to Implement the Checklist
- Summary

Klocwork

The Proven Leader. Software Security. Software Quality.

Poor Software quality - Root Cause of Security Vulnerabilities

- 35% of all successful attacks are a result of software defects

Gartner

- Most vulnerabilities come from software implementation (coding) errors (*Congressional Testimony, Richard D. Pethia, CERT Director)*

CERT® Coordination Center

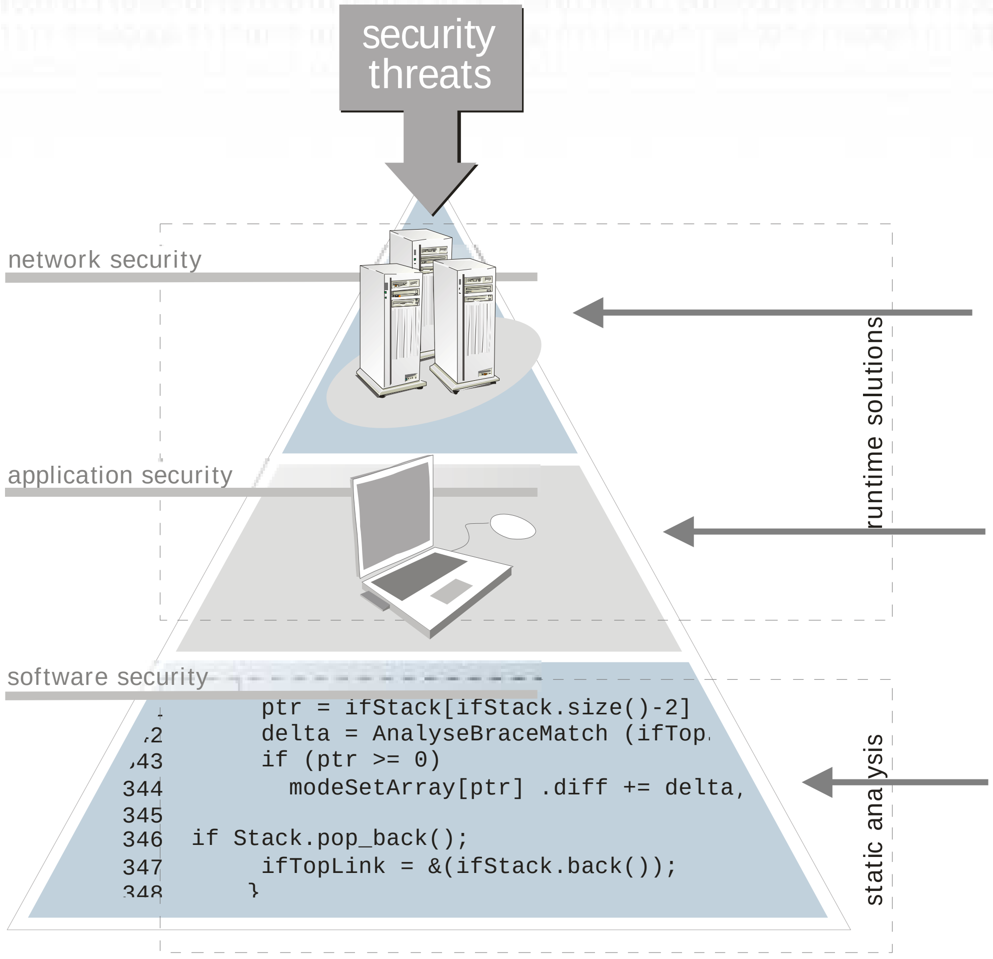
- Traditional, black-box functional testing will not identify security problems, since it looks for predictable user behavior, not unpredictable hacker attacks (*Watts Humphreys of the SEI Institute)*

Carnegie Mellon Software Engineering Institute

- 64 percent of [developers surveyed] rated writing secure code as a key new skill that they want to acquire, and they want tools to really be able to go in and audit what they're doing." (*Bill Gates referencing a Microsoft survey, RSA Conference 2005: "Security: Raising the Bar" 02/15*)

Klocwork
The Proven Leader. Software Security. Software Quality.

How big is the problem?



```
        ptr = ifStack[ifStack.size()-2]
        delta = AnalyseBraceMatch (ifTop
        if (ptr >= 0)
          modeSetArray[ptr] .diff += delta,

  if Stack.pop_back();
        ifTopLink = &(ifStack.back());
      }
```

You locked the doors!

Put the bars on windows!

However, the bad guys are already in the house!

Klocwork
The Proven Leader. Software Security. Software Quality.

We Have Become Complacent?

Software Security Warnings Issued Daily

- *Real Player flaw allows customers' PCs to be hijacked - Internet News*
- *High Alert: Security Flaws Found in Symantec Software*
- *Security Flaw Found in Nokia Phone - PC World*
- *Vulnerability found in mpg123, the open-source audio player*
- *WinZip contains flaw allowing remote users to execute malicious code*

We accept these warnings as routine

- ***This has created complacency in the industry***
- ***One-time patches or fixes for security or other quality issues are merely standard practice rather than addressing security root causes***

What are the risks in software code?

1. Unintentional vulnerabilities due to known programming mistakes
   - Coding Bugs
     - e.g. Buffer Overflow – programmer does not properly check the boundaries of the buffer, creating a hole that can be exploited
   - Design-related Bugs
     - e.g. Unvalidated data – programmer does not validate user or other input properly
   - Architecture-related defects
     - e.g. System failure – unchecked system calls and no syslog entries about failures

2. Intentionally inserted vulnerabilities (trojan horses, backdoors, etc)
   - Reduce risk by enabling focused manual code inspections:
     - Inspect code that has been changed (granular churn measures can accurately isolate where changes have been made)
     - Inspect areas of software with high complexity

Ongoing risk: Software development trends

Code Size & Complexity

- Software continues to grow in size and complexity: by 2010 70% of all developers will be working on existing code rather than new development
- **Risk:** most vulnerabilities are caused by minor programming errors, which are difficult to inspect and isolate in a large, multi-million LOC system that has millions of potential software usage paths

Software Reuse

- Common and economical approach to development of new software that leverages and hence, propagates code which was developed potentially many years in the past
- **Risk:** approaches to developing secure software is only now becoming a mainstream part of modern programming, therefore systems that were developed years ago in older languages couldn't have been developed using today's best practices for secure software

Ongoing risk: Software development trends

Use of Open Source Software

- Becoming a standard part of both commercially developed apps (i.e. OEM) and is also deployed widely across enterprise IT infrastructures (e.g. Apache web server)
- **Risk:** Both malicious and unintentional vulnerabilities can be into the software and then deployed in your environment.

Software Offshoring

- Common practice for many software organizations, large and small since offshore locations can provide talented yet inexpensive source of development talent
- **Risk:** Have programmers from the offshore team been trained on secure programming techniques? What are the risks of malicious vulnerabilities being inserted into the software? Will you have time to inspect and test the software once you receive it back from the offshore team?

Klocwork
The Proven Leader. Software Security. Software Quality.

Klocwork

The Proven Leader. Software Security. Software Quality.

Review of Common Software Vulnerabilities

Klocwork Inc.
35 Corporate Drive
Burlington, MA 01803
1.866.556.2967
www.klocwork.com

Tuesday, October 18, 2005

C/C++ Security

- Buffer overruns
- Non-Null Terminated Strings
- Tainted data
- Access problems
- Format string
- Insecure storage
- Injection flaws
- Ignored return values

C/C++ Security: Buffer Overruns

- Very common error and is dangerous
  - Overflow on stack can provide code injection entry point
  - Common way for hackers to establish shell on vulnerable host
- Applies to all cases of buffer and array out of bounds errors
  - Stack overflows
  - Buffer Underwrite
  - Miscalculated or missing null termination
  - String manipulation, buffers, arrays, etc.

C/C++ Security: Tainted Data

- Data received from user or from an external system is used unchecked in potentially dangerous functions
- User input is normally assumed to be clean, seldom is radically different user input used for testing
- Vulnerability exists when trace between data source and data sink is detected

Klocwork
The Proven Leader. Software Security. Software Quality.

C/C++ Security: Buffer Overruns from Tainted Data

- Enhanced tools can link tainted data to discovered buffer overrun errors
- Result is a report for buffer overruns that can be linked to data coming from outside the system

C/C++ Security: Non-Null Terminated Strings

- Similar to buffer overrun but specific to strings with missing termination
- Common coding problem and equally common exploit
- Also should link to tainted data analysis to generate different error code for prioritization

C/C++ Security: Tainted Data Causing other Errors

- Tainted data can be linked to other error types
  - Non-null terminated strings
  - Buffer overruns
- Users can set different severity and error message for these errors
  - This type of error is typically much more exploitable than other errors

C/C++ Security: Access Problems

- Monitors calls to functions that perform security-sensitive actions
  - Application requires administrator privileges
  - Applications raising operating privilege
  - Application not running with least privilege
  - Time of creation, time of use race condition
  - Improper sequencing of security-sensitive functions
  - Using easily guessable temporary file names

Klocwork
The Proven Leader. Software Security. Software Quality.

C/C++ Security: Format String Vulnerabilities

- Formatting and copying a string larger than assigned buffer (e.g., sprintf(), snprintf())
- Not using string format size in format string, i.e., %s instead of %10s
- Using a variable format size for string
- Use of variable or no format size in scanf()

Klocwork
The Proven Leader. Software Security. Software Quality.

C/C++ Security: Ignored Return Values

- Application is ignored important return values
  - Detection of failure in called function is critical for maintaining secure operation
  - Example: On Unix/Posix/Linux, not checking GetLastError on security sensitive functions

The Proven Leader. Software Security. Software Quality.

C/C++ Security: Calls to Dangerous Functions

- Warning on use of dangerous functions
  - Example of insecure coding practices
  - Usually insecure to use and more secure alternatives exist or
  - Must be careful of their use and validate parameters used

The Proven Leader. Software Security. Software Quality.

Java Security: Common Vulnerabilities

Attack Category	Vulnerability Category	Vulnerability Type
Denial of Service	Product stability	Resource is not disposed properly
		Null pointer dereference
		Un-initialized read of field in constructor
		Resource leaks
		Array bounds violations
		Tracing size of array allocations from user input.
		Finding resource leaks that are attributable to user input (specialized form of existing resource leak detection with inter-procedural capability)

The Proven Leader. Software Security. Software Quality.

Java Security: Common Vulnerabilities

External Attack	SQL Injection (executeQuery) – Tainted Data	Finds a trace where parameter can injected into SQL database, forming a different request
	Process Injection (Runtime.exec) – Tainted data	Finds a trace where parameter can injected into exec(), forming a different command.
	Cross-site Scripting (Servlet Response) – Tainted Data	Find strings printed into HttpServletResponse.getWriter() stream, which are coming from parameters
	Path Injection (file operations) – Tainted Data	Find file manipulating methods and trace parameters up call chain to see if they came from unchecked data
	Unchecked Email – Tainted Data	Check for tainted data for parameter used to receive user email addresses.
	Shared Variables	Find unsynch. access to static variable in sessions – possible exploitation of thread safety violation.

Klocwork

The Proven Leader. Software Security. Software Quality.

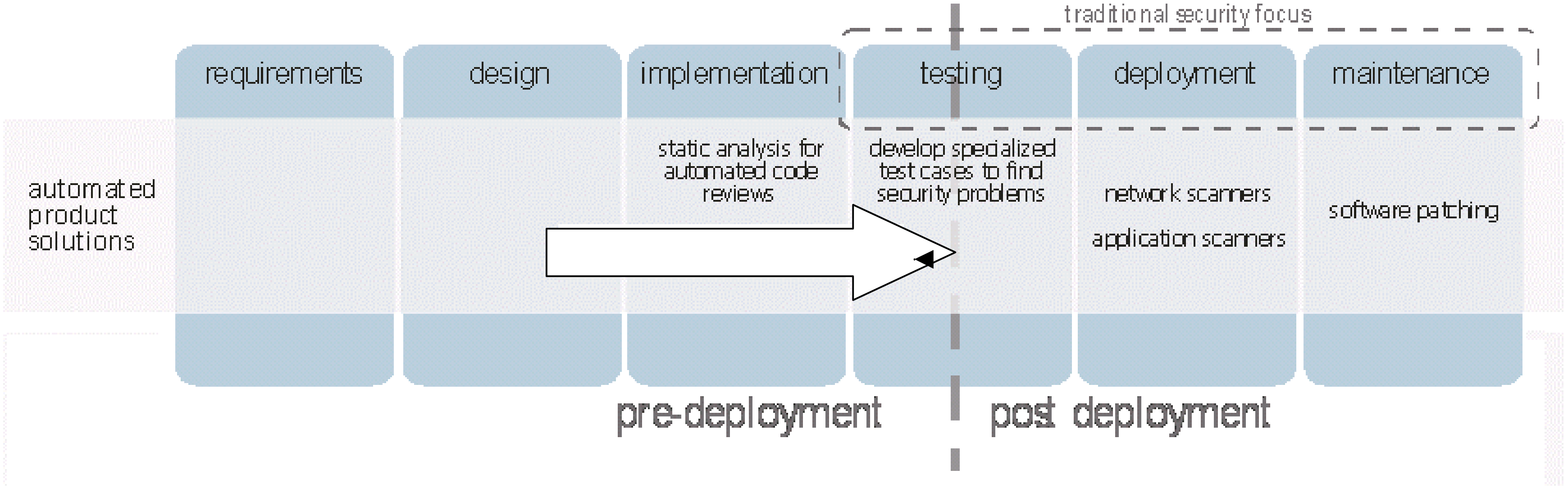
Java Security: Common Vulnerabilities

Misuse of Code	Class/Field/Method mutation	Comparing class objects by name
	Exposed Representation	Method may expose internal representation by returning reference or storing a mutable object (scoped by user)
	Protected Finalizer	Checks for unprotected finalizer methods (scoped by user).
	Detect Mutable Fields	Non-private static field is mutable object such as array or Hashtable
	Exposed Interface	A final static field that is defined in an interface references a mutable object such as an array or hashtable. This mutable object could be changed by malicious code or by accident from another package. To solve this, the field needs to be moved to a class and made package protected to avoid this vulnerability.

The Proven Leader. Software Security. Software Quality.

Where in your process can vulnerabilities be found?

- *“Time Zero” – the lowest cost correction point in the development process.*


traditional security focus
requirements
design
implementation
testing
deployment
maintenance
automated product solutions
static analysis for automated code reviews
develop specialized test cases to find security problems
network scanners
application scanners
software patching
pre-deployment
post deployment

Cost *	**$250**	**$2,500**	**$10,000**	**$30,000**
Stage	**Code Inspect**	**Designer Test**	**System Test**	**Post Release**

***Building Software with Quality and Speed (07/09/2004)**
Srivibhavan Balaram, R&D Director, Hewlett Packard
Systems Technology & Software Division, Bangalore

Klocwork

The Proven Leader. Software Security. Software Quality.

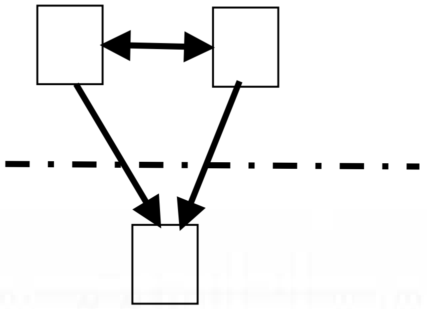
Software Industry-
Victims of our own Success

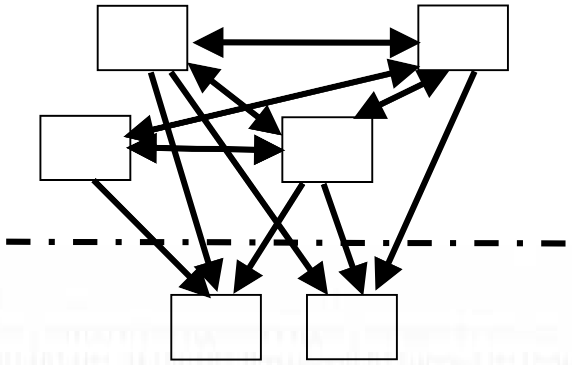
Klocwork Inc.
35 Corporate Drive
Burlington, MA 01803
1.866.556.2967
www.klocwork.com

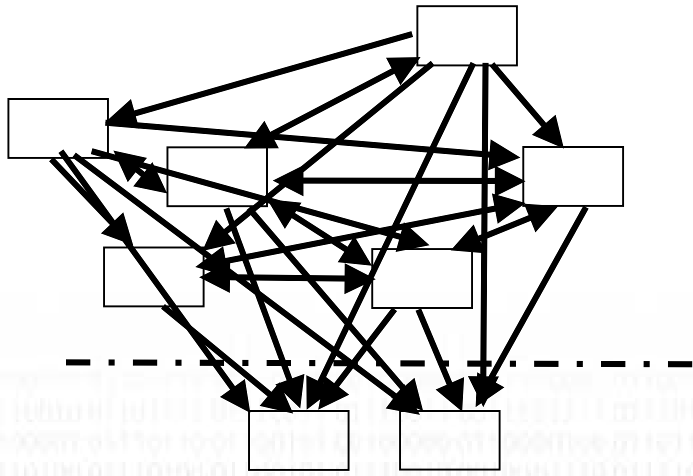
Tuesday, October 18, 2005

Software companies are victims of their own success

- How Success Destroys Architecture Ultimately Eroding Software
  - A story about real projects, real people …a story about every successful product
  - Skunk works project turns into a large development organization
    - The new designers never participated in the kickoff meetings held by the architects in preparation for the first release – they just got a document
    - They have no idea how the architectural pictures relate to the current code base, nor do they understand the "design intent" of the architecture
    - They don't really understand the customer needs or the product
    - They came to CODE, and by God, they will!
  - Business exceeds expectations…what is the encore!!!
    - The product demands & implementation are outstripping the architecture
    - Patches to fix things that used to work, are either broken, or running much slower
    - There is pressure from marketing to release each of the product variants on separate schedules





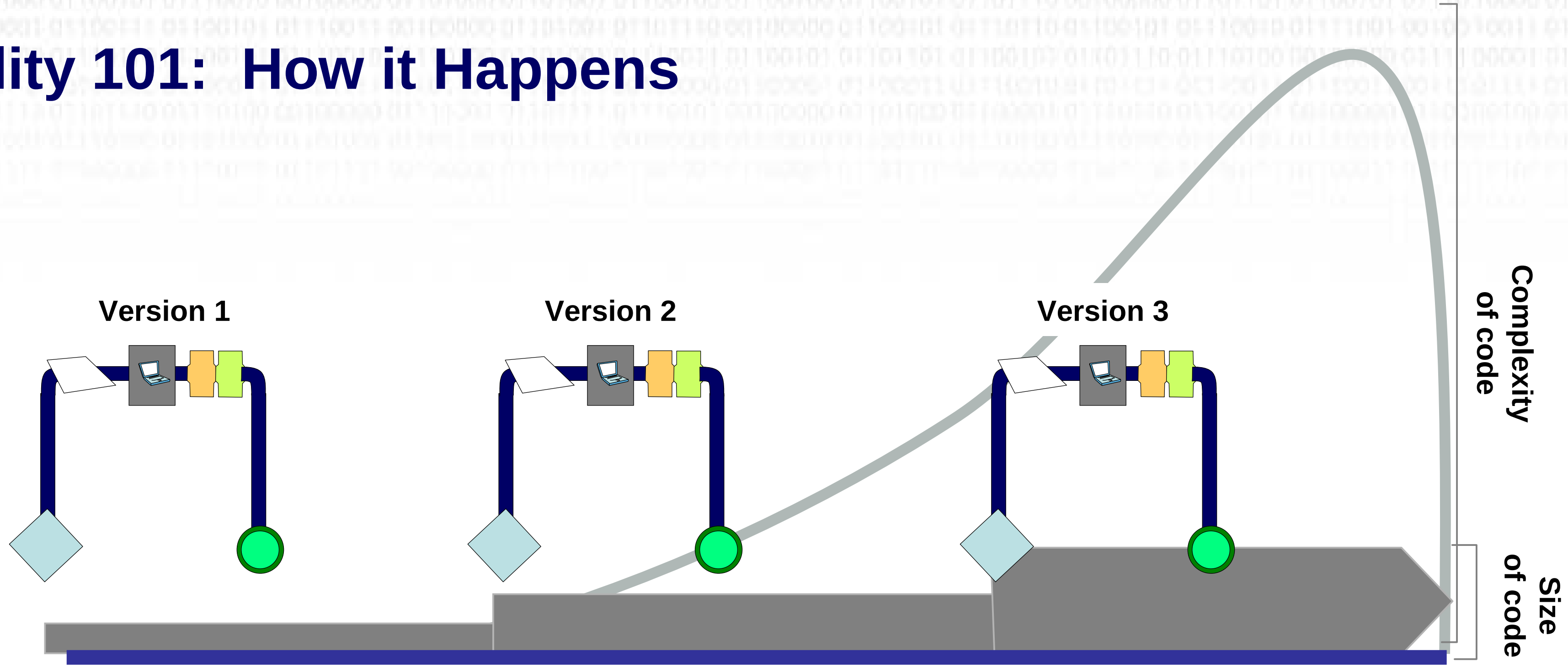


Klocwork
The Proven Leader. Software Security. Software Quality.

Software erosion begins with …

- Code Defect Fixes
  - Fixes are introduced in isolation, not understanding the full scope and impact
- Security Vulnerability bandages
  - Security Panic
    - Quick fixes with many shortcuts and not full investigation for possible “weak links”
    - New capability is not built into system as a part of the architecture, rather it had to be shoehorned in as a retrofit in response to yet another security panic
- Code Bloat
  - Extra “includes” causing long builds and messy interfaces
  - Cloning practices - copied functionality with adjustment for new feature
  - Dead code overhead
  - Memory constrains
- Performance Improvements
  - Local change – global effect never investigated
- Bad Practices
  - Flawed approaches to design and bad coding practices

Reality 101: How it Happens



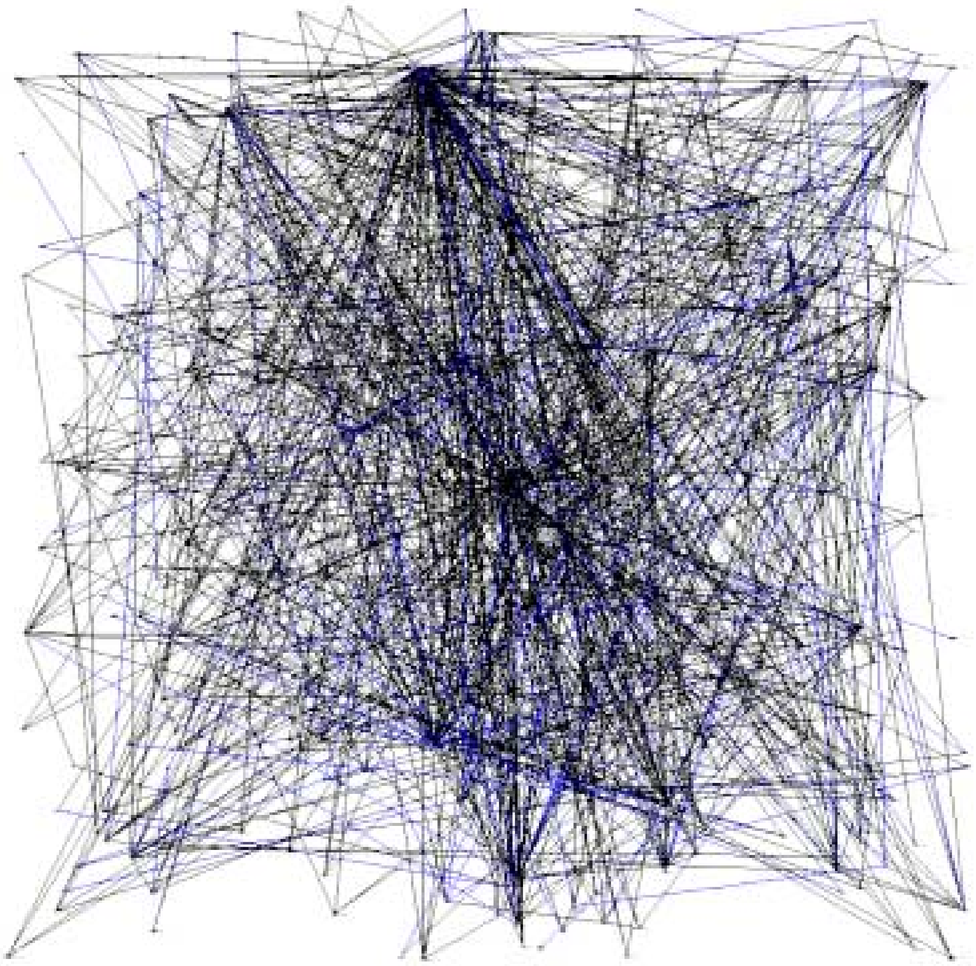
Plan	Reality
▪Tight design spec ▪Solid architecture ▪Comprehensive Functionality	▪Reduced functionality ▪Architecture erosion ▪Limited testing to meet ship date ▪Plan to, “fix bugs in v1.5.”

Plan	Reality
▪Clean up bugs ▪Take app to “next level” ▪Respond to 1st customer fixes ▪Control Software Complexity	▪Key architects & developers gone ▪Documentation is non-existent ▪New team on learning curve ▪Everything changed but the ship date ▪Performance degrading

Plan	Reality
▪Must fix the architecture ▪No new defects ▪Reuse components for other applications ▪Outsource some development ▪Improve performance	▪Complexity and code is growing too quickly ▪Bug fix versus feature battle underway ▪Architecture is unknown ▪Security concerns

Klocwork
The Proven Leader. Software Security. Software Quality.

Reality 101: Typical Customer Code Base



Eroded architecture resists comprehension

Code base profile:
- **C/C++/JAVA**
- **Multi MLOCs**
- **post release 1**

Klocwork
The Proven Leader. Software Security. Software Quality.

Klocwork

The Proven Leader. Software Security. Software Quality.

Key Considerations for Managing Modernization to Achieve Higher Quality and Better Security in Your Software

Klocwork Inc.
35 Corporate Drive
Burlington, MA 01803
1.866.556.2967
www.klocwork.com

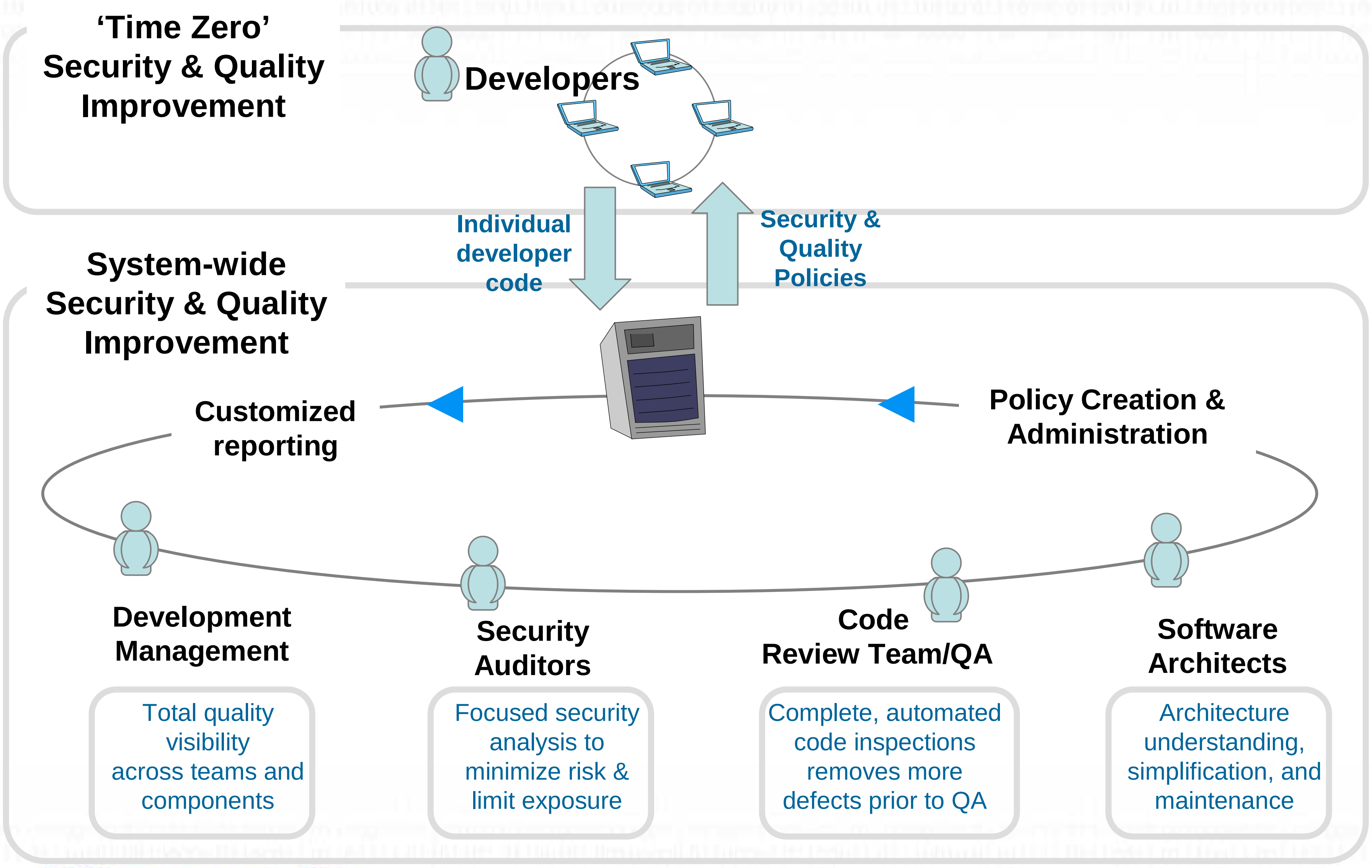
Tuesday, October 18, 2005

Smart Software Development Process

Smarter software development process supported by tools will:

- Reduce the risk to an organization created when an individual programmer is the sole source of knowledge about a given application
  - Visibility into an application's context and its health at any point in the development life cycle
  - Manage software assets as any other business assets
- Improve development management to direct developers to build applications based on a solid and secure architecture and coding practices
  - Managers being proactive instead of reactive
  - Require tools to help developers be more productive while applying solid and secure coding practices during application development

Solution: Integrate static analysis into your process



'Time Zero'
Security & Quality
Improvement
Developers
Individual developer code
Security & Quality Policies
System-wide
Security & Quality
Improvement
Customized reporting
Policy Creation & Administration
Development Management
Security Auditors
Code Review Team/QA
Software Architects
Total quality visibility across teams and components
Focused security analysis to minimize risk & limit exposure
Complete, automated code inspections removes more defects prior to QA
Architecture understanding, simplification, and maintenance

Klocwork

The Proven Leader. Software Security. Software Quality.

How to get there …

Keys to Success:

1. Management
   - understand that there is not silver bullet
   - be more engaged
2. Process
   - establish a process that will guide the team through a focused and incremental change in development outcome
   - this process needs to be highly automated
3. Tools
   - identify required tools for your environments that will support automation
4. Developers practices
   - quality/security Audits are important, but managers must think in terms of an ‘In Process’ development change
   - sustained software quality/security success requires a modification of current development practices

Klocwork

The Proven Leader. Software Security. Software Quality.

Changing the organizational culture …

- Audit is very effective for assessing security and quality of existing software
  - Measure and assess overall security and quality status of your software
  - Identify critical issues
  - Understand which components are highest risk

- However it is important to note that audits alone will not change development behaviour and ensure the development of secure coding practices

- Development practices can only change with an In-Process solution
  - Integrated with development environment
  - Solution that is easy-to-use for developers, works within their current tools
  - Progress is tracked 'build over build' and allows both development team & management to measure progress and implemented focused & incremental change

Klocwork
The Proven Leader. Software Security. Software Quality.

Key criteria for a successful In-Process solution

- Solution must easily integrate with current build environments, enabling analysis that runs each build
- Needs to scale to meet the needs of large scale development shops
- Must provide easy of developer adoption: integration into large number of IDEs (Eclipse, VS 6 & .NET, text editors, etc) key to developer deployments

Microsoft Visual Studio.net

eclipse FOUNDATION MEMBER

Klocwork
The Proven Leader. Software Security. Software Quality.

Klocwork

The Proven Leader. Software Security. Software Quality.

Modernization Checklist for Success

Klocwork Inc.
35 Corporate Drive
Burlington, MA 01803
1.866.556.2967
www.klocwork.com

Tuesday, October 18, 2005

Checklist for Success

✓	Perform a system-wide security and quality audit of your software
✓	"Stop the bleeding" for new development
✓	Perform a more detailed In-Process audit
✓	Convert In-Process audit findings into code improvements
✓	Measure improvement
✓	Manage iterations of security and quality improvements

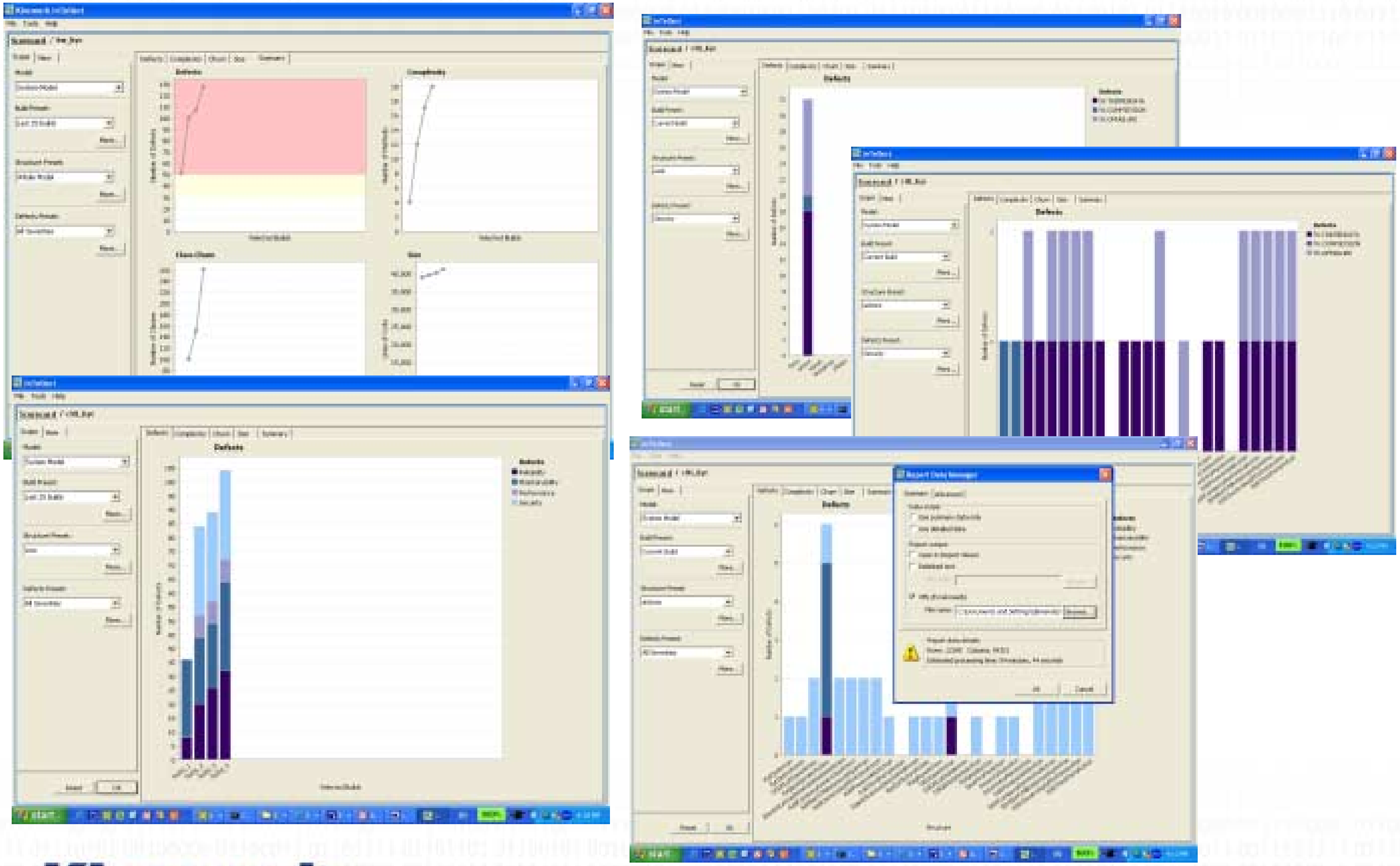
Klocwork

The Proven Leader. Software Security. Software Quality.

Perform a system-wide security and quality audit of your software

- The goal is to uncover the weaknesses in your software and risks that they present to your organization
- Specific areas that can be addressed with ‘fact based’ metrics;
  - How extensively has the development team been using bad/unsafe coding practices
  - How many quality and security implementation defects (e.g., resource leaks, buffer overflows) are discovered
  - How many security flaws (e.g., the usage of un-validated data) are discovered
  - Which components are highest risk? Who developed these components?
  - Build tactical roadmap
    - Identify and isolate high priority issues that need to be taken care of immediately
  - Start building strategic roadmap
    - Start the list of issues that require strategic panning

Manage Applications as Business Assets



Klocwork

The Proven Leader. Software Security. Software Quality.

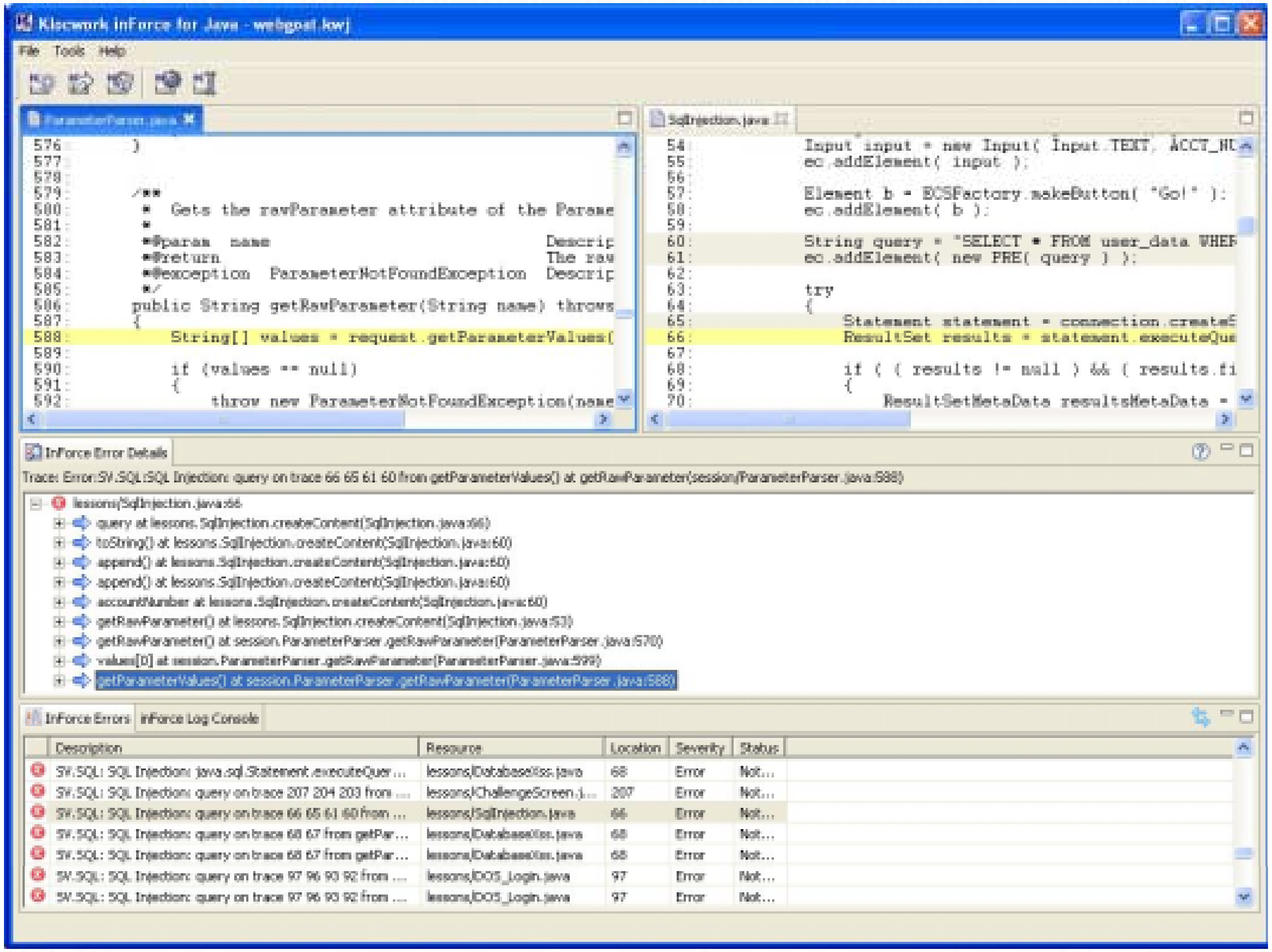
"Stop the bleeding" for new development

- The integration of a "stop the bleeding" process into your development environment is an important step to get immediate control high priority issues, overcome weaknesses & show progress
- This can be accomplished by deploying an extensible, rules-based 'quality/security compiler' to ensure that tactical issues are addressed and ensure good coding practices during new development

- Three possible integration points:
  - **Time0** - Integrate the desktop quality/security compiler into the developers' favorite IDE (e.g., Eclipse or VisualStudio) so they can run it while coding.
  - **Time0+1** - Integrate the desktop quality/security compiler into the configuration management system and the security compiler will automatically check after the developer has coded but before code is submitted into loadbuild
  - **Time0+2** - Integrate the system quality/security compiler – if not done in previous phase – into the product's build process

Klocwork
The Proven Leader. Software Security. Software Quality.

Improve System by “Focusing on What Matters”



Klocwork inForce for Java - webgoat.kwj

ParameterParser.java

```
576:        }
577:
578:
579:        /**
580:         *  Gets the rawParameter attribute of the Parame
581:         *
582:         *@param  name                                      Descrip
583:         *@return                                           The raw
584:         *@exception  ParameterNotFoundException            Descrip
585:         */
586:        public String getRawParameter(String name) throws
587:        {
588:            String[] values = request.getParameterValues(
589:
590:            if (values == null)
591:            {
592:                throw new ParameterNotFoundException(name
```

SqlInjection.java

```
54:             Input input = new Input( Input.TEXT, ACCT_NU
55:             ec.addElement( input );
56:
57:             Element b = ECSFactory.makeButton( "Go!" );
58:             ec.addElement( b );
59:
60:             String query = "SELECT * FROM user_data WHER
61:             ec.addElement( new PRE( query ) );
62:
63:             try
64:             {
65:                 Statement statement = connection.createS
66:                 ResultSet results = statement.executeQue
67:
68:                 if ( ( results != null ) && ( results.fi
69:                 {
70:                     ResultSetMetaData resultsMetaData =
```

InForce Error Details

Trace: Error:SV.SQL:SQL Injection: query on trace 66 65 61 60 from getParameterValues() at getRawParameter(session/ParameterParser.java:588)

```
lessons/SqlInjection.java:66
    query at lessons.SqlInjection.createContent(SqlInjection.java:66)
    toString() at lessons.SqlInjection.createContent(SqlInjection.java:60)
    append() at lessons.SqlInjection.createContent(SqlInjection.java:60)
    append() at lessons.SqlInjection.createContent(SqlInjection.java:60)
    accountNumber at lessons.SqlInjection.createContent(SqlInjection.java:60)
    getRawParameter() at lessons.SqlInjection.createContent(SqlInjection.java:53)
    getRawParameter() at session.ParameterParser.getRawParameter(ParameterParser.java:570)
    values[0] at session.ParameterParser.getRawParameter(ParameterParser.java:599)
    getParameterValues() at session.ParameterParser.getRawParameter(ParameterParser.java:588)
```

InForce Errors | inForce Log Console

Description	Resource	Location	Severity	Status
SV.SQL: SQL Injection: java.sql.Statement.executeQuer…	lessons/DatabaseXss.java	68	Error	Not…
SV.SQL: SQL Injection: query on trace 207 204 203 from …	lessons/ChallengeScreen.j…	207	Error	Not…
SV.SQL: SQL Injection: query on trace 66 65 61 60 from …	lessons/SqlInjection.java	66	Error	Not…
SV.SQL: SQL Injection: query on trace 68 67 from getPar…	lessons/DatabaseXss.java	68	Error	Not…
SV.SQL: SQL Injection: query on trace 68 67 from getPar…	lessons/DatabaseXss.java	68	Error	Not…
SV.SQL: SQL Injection: query on trace 97 96 93 92 from …	lessons/DOS_Login.java	97	Error	Not…
SV.SQL: SQL Injection: query on trace 97 96 93 92 from …	lessons/DOS_Login.java	97	Error	Not…

Klocwork

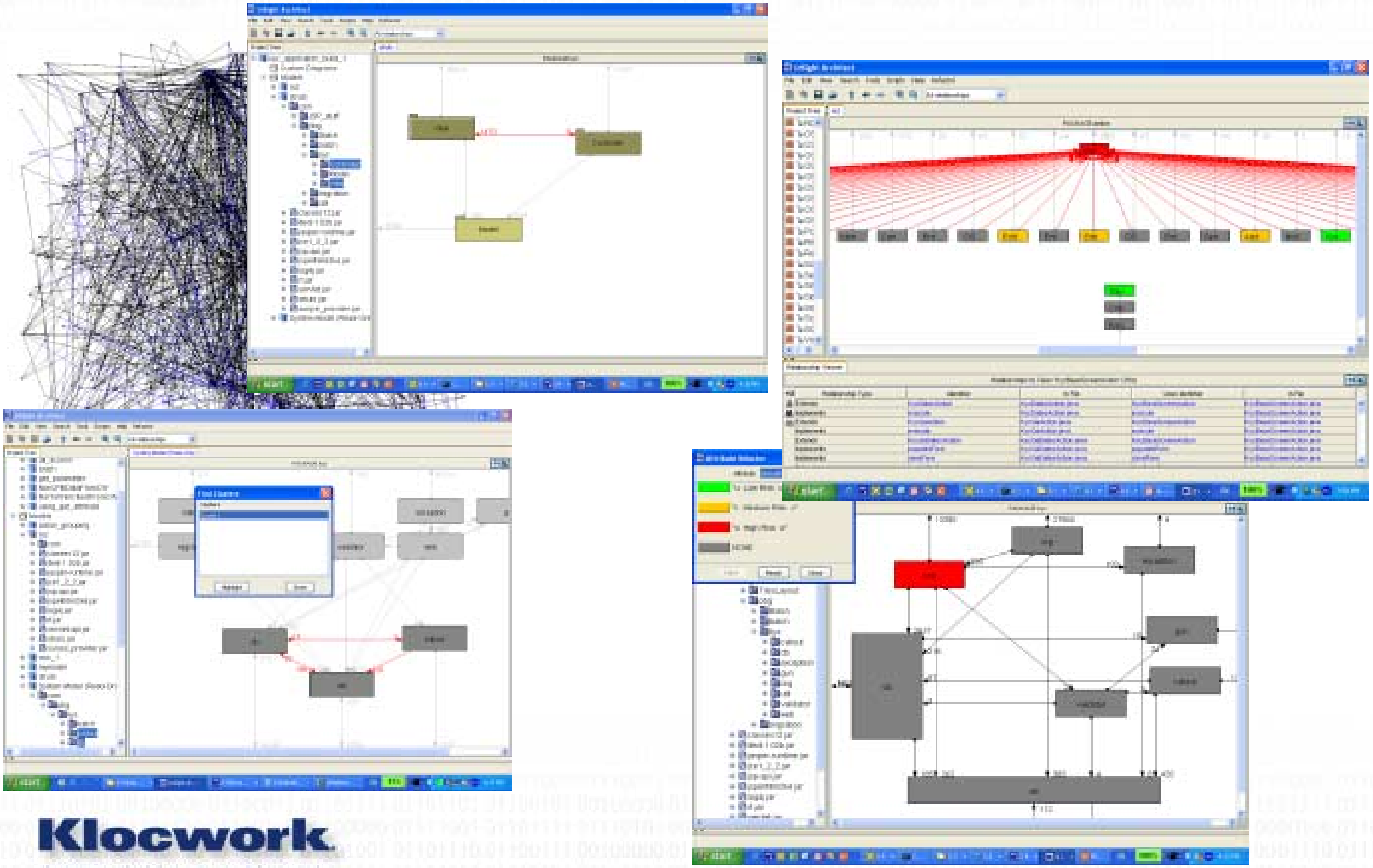
The Proven Leader. Software Security. Software Quality.

Perform a more detailed In-Process audit

- This analysis should go beyond the detection of implementation defects and looks for weaknesses in software design and architecture
- The findings would be used as planning input to your strategies

- Key Elements of this Step:
  - Excavate the software architecture & communicate results
  - Create a snapshot of the architecture by identifying the key components along with their boundaries and interfaces
  - For example, analysis of un-validated data becomes more meaningful, since it now takes the security architecture view into account and uses the exact "boundaries of trust" between components.

Business Risk Begins With Architecture Erosion



Klocwork
The Proven Leader. Software Security. Software Quality.

Convert In-Process audit findings into code improvements

- Clean up your ‘one-time fixes’ and update your rules-based quality/security compiler to include any new policy requirements.

- The quality/security compiler should be incrementally (priority-based) re-configured to enforce application-specific rules and security policies
  - Each policy should be given a time stamp as to when it will take effect
    - New checkers are automatically activated when their priority kicks in
  - Priorities and workload for the development teams can be managed in the context of ongoing development and release schedules
  - Enables management to proactively manage the new secure development roadmap.

- In addition, the list of all high-priority, one-time fixes with detailed actionable items should be generated and passed to the appropriate development teams for clean up and repair

Klocwork

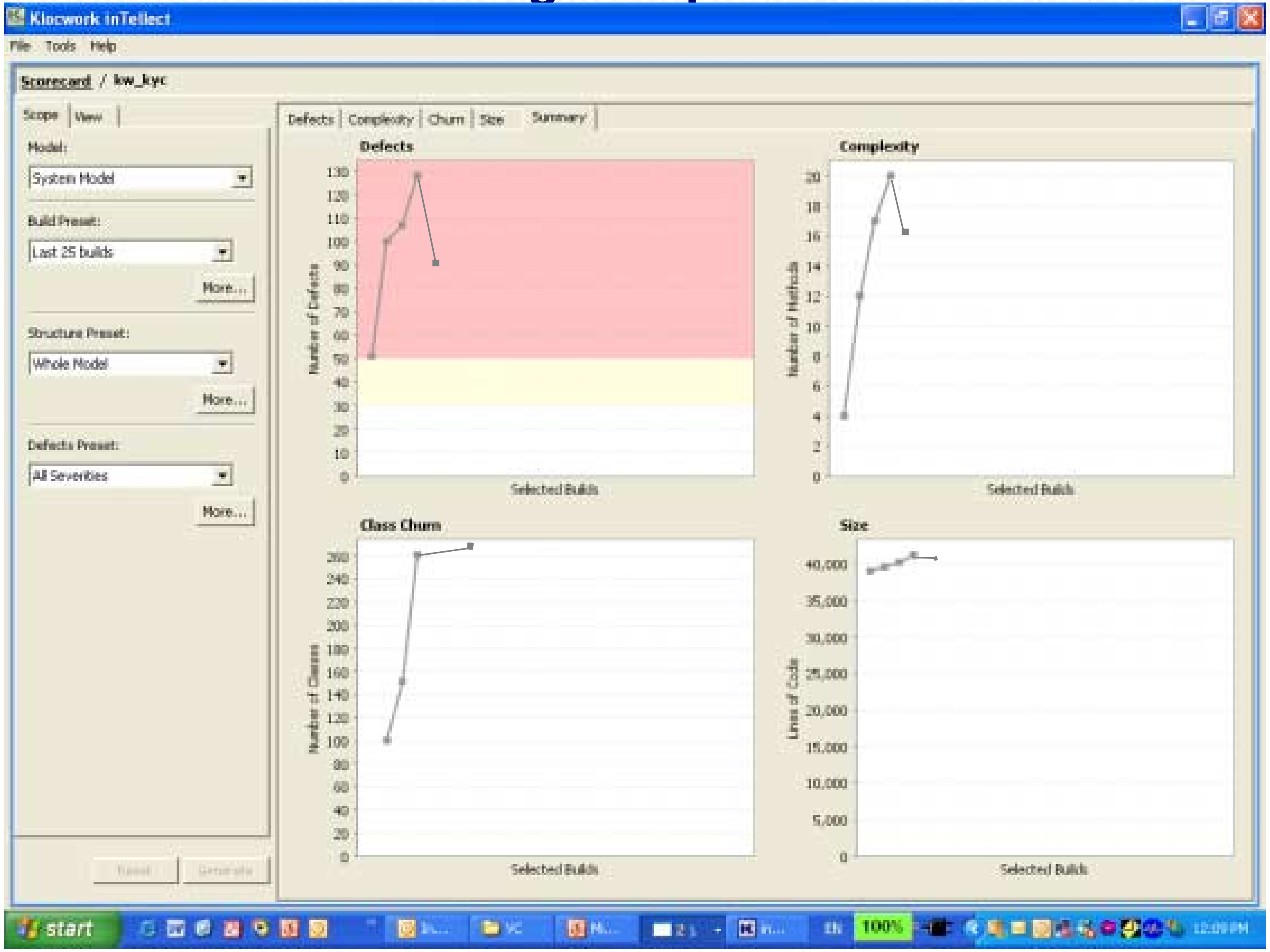
The Proven Leader. Software Security. Software Quality.

Measure improvement

- When implementing any security improvements it is imperative to measure them and monitor their trends, otherwise it will never be clear if you are winning or losing the battle
- Just as Business Intelligence solutions allow executives to track progress in other parts of a business, it is imperative to measure development improvements against objectives
- Perform the following steps:
  1. Configure thresholds for particular quality/security driven metrics based on your objectives
  2. Monitor the metrics trends over time to measure improvement
  3. Monitor how close or far the results are relative to your thresholds
  4. Assess if your objectives have been set properly (possible after several measurements) and if necessary, re-adjust objectives & timeframes for policy enforcement

Klocwork
The Proven Leader. Software Security. Software Quality.

Reverse the Trends – Begin Improvement!!



Manage iterations of security improvements

- Repeat the overall process (Analysis->Improve->Measure)

- The key is to streamline the change process based on priorities and balance it with the education and training of the team

- Changes should be focused on the highest priority weaknesses, while addressing secondary weaknesses more and more over time as the team becomes more educated

- At each iteration, the “quality/security compiler" should be re-configured to control the new baseline in an automatic fashion
  - This means that once the new baseline is agreed upon, new violations will be flagged by the quality/security compiler at the specified time

The Proven Leader. Software Security. Software Quality.

Klocwork

The Proven Leader. Software Security. Software Quality.

How to Implement Checklist

Klocwork Inc.
35 Corporate Drive
Burlington, MA 01803
1.866.556.2967
www.klocwork.com

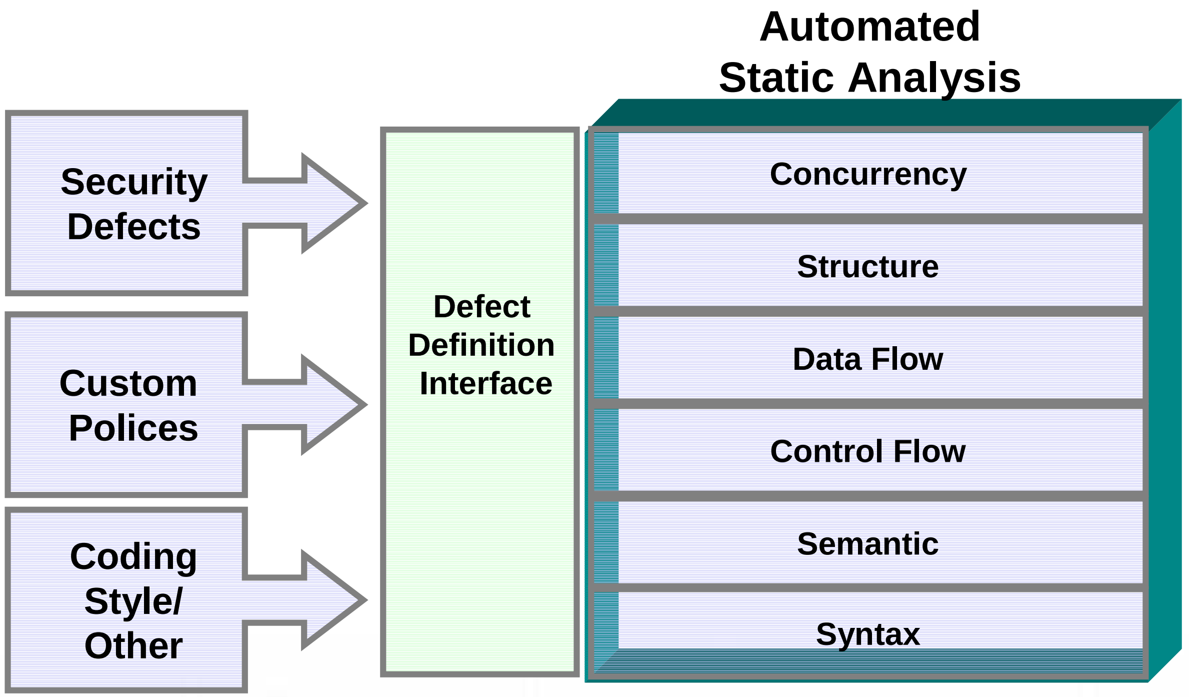
Tuesday, October 18, 2005

Implementing Checklist

1. Automated Static Analysis Technology
   - Greatly reduces the need for formal code reviews
   - Automation technology will isolate only high-probability vulnerabilities, enabling developers to ‘inspect & correct’ on their own
   - Generation of vulnerability tracking metrics & improvement data is automated
   - Provides multiple process integration points: at desktop, build level or combination
2. Manual Code Scans/Reviews and Capture SME Knowledge
   - Rigorous code reviews can enable detection of implementation level security vulnerabilities
   - Requires change in process and disciplined methodology
   - Collection of vulnerability data & improvement tracking is manual
3. Use of OMG Knowledge Discovery Metamodel (KDM) Specification (coming soon, end of 2005)
   - KDM defines holistic view of application and provides list of information that you need to collect from your system and understand before starting modernization
   - Facilitates interoperability of modernization tools (static analysis)

Automated Static Analysis Defined

- Multiple layers of analysis
- Multiple integration points
- Extensibility to create custom vulnerability checks
- Tools can be leveraged by entire development team

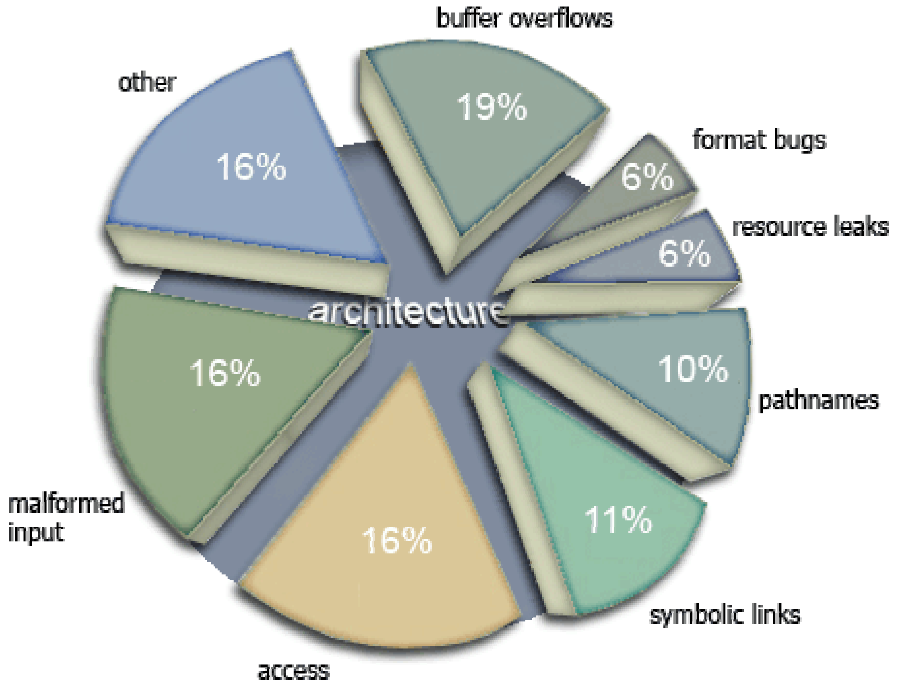


Klocwork
The Proven Leader. Software Security. Software Quality.

Automated Static Analysis - Vulnerability Coverage

Static Analysis Coverage
• Detects 7/8 high-level categories

Cost of finding defects:
• automated static analysis tools check 100LOC/sec
• dynamic testing requires the manual creation of .5-1 test line of code per application line of code [Ref.2]



1. Common Vulnerabilities and Exposures Database, Jan-Sep 2001[Evans & Larochelle, IEEE Software, Jan 2002.]

2. Test Driven Development as a Defect Reduction Practice White Paper, IBM& NC University, IEEE Symposium 2003

Klocwork

The Proven Leader. Software Security. Software Quality.

Summary

Klocwork Inc.
35 Corporate Drive
Burlington, MA 01803
1.866.556.2967
www.klocwork.com

Tuesday, October 18, 2005

Summary

- Sustained software quality/security success requires a modification of current development practices
- Quality/security Audits are important, but managers must think in terms of an ‘In Process’ development change
- Keys to Success:
  1. Focused and incremental change
  2. Measure improvement
  3. Select the right balance of new technology & processes

Thank-you

Klocwork
The Proven Leader. Software Security. Software Quality.