Software Code Quality Checking (SCQC)
No Clearance for This Secret: Software Assurance is MORE Than Security

Nominee
International Security Executives (ISE®)
Information Security Project of the Year
North America
Government Sector
POINT OF CONTACT

• IT Specialist, SCQC Task Manager
  – John Keane
  – 703-380-5554
  – jkeanejr@aol.com
  – Level 3 Certified Test & Evaluation Engineering
  – Level 2 Certified Program Management
  – Level 1 Certified
    • Information Technology
    • Life Cycle Logistics
    • Business – Financial Management
    • Systems Planning, RD&E – Systems Engineering
A WORD OF CAUTION

• This presentation is not meant to endorse or recommend the use of any of the products identified within.

• The presentation is intended to provide insight into specific lessons-learned through the application of a disciplined methodology, supplemented by a suite of tools as implemented by skilled practitioners.
BOTTOM LINE UP FRONT

• **Software Assurance is a BLEND of Code Quality and Software Security**
  – QUALITY is the biggest driver of sustainment cost
  – SECURITY causes the most “pain and agony”
• The foundation of ASSURED software is QUALITY software.
  – Software Assurance is 5 parts Code Quality with 2 parts Software Security.
• **Code Quality scans should ALWAYS precede Security scans**
  – Improves effectiveness and efficiency of the security scans
  – Validated by MULTIPLE reports and analyses based on hands-on experience
  – Discovers risks in addition to security (e.g., performance)
  – Prevents “securing” bad quality/poorly performing code.
WHAT IS SCQC?

Software Code Quality Checking (SCQC) is

- A scan of the source code, executables, and related artifacts, e.g., documentation, to ensure that the System Under Review can
  - Continue with development, demonstration, and test; and
  - Can meet the stated performance, maintainability, and usability requirements within cost (program budget), schedule (program schedule), risk, and other system constraints.

- **Encompasses the use of**
  - Static code analysis,
  - Static security analysis,
  - Dynamic code analysis,
  - Dynamic security analysis, and
  - Architectural analysis and

- Is **USUALLY** performed using automated tools.
**Static Analysis** is the analysis of computer software and related documentation that is performed without actually executing programs built from the software. *It is a recognized BEST practice that SHOULD precede Static Security Analysis.*

**Static Security Analysis** is the analysis of computer software that is performed without actually executing programs to detect and report weaknesses that can lead to security vulnerabilities.

– *Static security analysis, in conjunction with manual review of the code, is recommended in section 5.4 of the DISA Applications Security and Development Security Technical Implementation Guide.*

**Dynamic Program Analysis** is the analysis of computer software and related documentation that is performed by executing programs built from that software on a real or virtual processor.

**Dynamic Security Analysis** is the analysis of computer software that is performed by executing programs to detect and report weaknesses that can lead to security vulnerabilities.

**Architectural Analyses** may be supported by automated tools but are usually conducted by manual walk-through of documentation and visual inspection of the code.
Software Assurance is the level of confidence that software is free from vulnerabilities, either intentionally designed into the software or accidentally inserted at anytime during its lifecycle and that the software functions in the intended manner. (CNSS Instruction No. 4009, 26 April 2010) - Implemented through FISMA

DOD Software Assurance is the level of confidence that software functions as intended and is free of vulnerabilities, either intentionally or unintentionally designed or inserted as part of the software throughout the lifecycle.

- Mandated by Federal Law - Section 932, 2011 NDAA
- Defined by Federal Law – Section 933, 2013 NDAA
RELATIONSHIP TO INFORMATION ASSURANCE

• **Information Assurance (IA)** relates to measures that protect and defend information and information systems by ensuring their availability, integrity, authentication, confidentiality and non-repudiation. These measures include providing for restorations of information systems by incorporating protection, detection, and reaction capabilities. Information systems include the software that controls the systems and processes data and information.

• Therefore, measures must be used to protect systems from (both) software vulnerabilities and unintended software processing that expose a system to compromises in availability, integrity and other security properties. **Software Assurance (SwA)** provides these measures.
“A Fool With A Tool is Still a Fool”
PMT256 - Program Management Tools Course
• TST203 - Intermediate Test and Evaluation
• Director, Federal Reserve Information Technology

To achieve success you need a combination of:
  Skilled People
  • Disciplined Processes
  • Enabling Tools and Technologies
SOME THINKING ABOUT TOOLS (2)

• Open Source Tools CAN Achieve SOME Measure of Success
  – Pre-Screen Before Using More Powerful Tools
  – Get rid of the “That Wasn’t Very Smart” Defects BUT
  – Only 60% to 65% effective AND
  – Don’t Get the “We should have NEVER done THAT” Mistakes

• Sooner or Later You WILL Have to Invest in the High Quality Licensed Tools To Get The Desired Results.
  – Can help prioritize what needs to get fixed from BOTH a security AND code quality perspective.
The Value Proposition
### DEFECT DENSITY - BASIC MODEL*

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error Distribution</strong></td>
<td>10%</td>
<td>20%</td>
<td>55%</td>
<td>15%</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hours to Correct</strong></td>
<td></td>
<td>50</td>
<td>120</td>
<td>380</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cost per Hour</strong></td>
<td></td>
<td>$100</td>
<td>$100</td>
<td>$100</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cost to Fix 1000 Errors</strong></td>
<td>$1,000,000</td>
<td>$6,600,000</td>
<td>$5,700,000</td>
<td>$13,300,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Stewart-Priven Group, 2009 Presentation to PMI-MHS “Software Inspection Success”

DAU Advanced Test and Evaluation (TST 303)
RETURN ON INVESTMENT
## 20% Defect Removal ROI Model

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Distribution</td>
<td>10%</td>
<td>20%</td>
<td>55%</td>
<td>15%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hours to Correct</td>
<td>50</td>
<td>120</td>
<td>380</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost per Hour</td>
<td>$100</td>
<td>$100</td>
<td>$100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost to Fix 1000 Errors</td>
<td>$1,000,000</td>
<td>$6,600,000</td>
<td>$5,700,000</td>
<td>$13,300,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### SCQC Applied

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Error Distribution</td>
<td>10%</td>
<td>40%</td>
<td>45%</td>
<td>5%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hours to Correct</td>
<td>50</td>
<td>120</td>
<td>380</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost per Hour</td>
<td>$100</td>
<td>$100</td>
<td>$100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cost to Fix 1000 Errors</td>
<td>$2,013,518</td>
<td>$5,400,000</td>
<td>$1,800,000</td>
<td>$9,213,158</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Cost Avoidance**: $1,013,518
- **SCQC Investment**: $1,868,230
- **ROI**: 118.75%
**OBSERVED SCQC BENEFITS**

- **Testing by itself is time consuming and not very efficient.***
  - Most forms of testing only find about 35% of the bugs that are present.

- **Static analysis** prior to testing is very quick and about 85% efficient.
  - As a result, when testing starts there are so few bugs present that testing schedules are cut down by perhaps 50%.
  - Static analysis will also find some structural defects that are not usually found by testing.

- **Static Security Analysis** prior to DIACAP testing *may* find, and be able to help correct, a large number of the **Applications Source Code** defects identified during Information Assurance testing.
  - When combined with Manual Code Review and Dynamic Analyses, can reduce “False Positives.”

---

*Capers Jones - Distinguished Advisor to the Consortium for IT Software Quality (CISQ). CISQ brings together industry executives from Global 2000 IT organizations, system integrators, outsourcers, and package vendors to jointly address the challenge of standardizing the measurement of IT software quality and to promote a market-based ecosystem to support its deployment.*
# 85% Defect Removal ROI Model

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Error Distribution</strong></td>
<td>10%</td>
<td>20%</td>
<td>55%</td>
<td>15%</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Hours to Correct</strong></td>
<td>50</td>
<td>120</td>
<td>380</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cost per Hour</strong></td>
<td>$100</td>
<td>$100</td>
<td>$100</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cost to Fix 1000 Errors</strong></td>
<td>$1,000,000</td>
<td>$6,600,000</td>
<td>$5,700,000</td>
<td>$13,300,000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

|                          | SCQC Applied               |                  |                    |                      |                      |
|--------------------------|----------------------------|------------------|--------------------|----------------------|
| **Error Distribution**   | 10%                        | 80%              | 7%                 | 3%                   |
| **Hours to Correct**     | 50                         | 120              | 380                |                      |
| **Cost per Hour**        | $100                       | $100             | $100               |                      |
| **Cost to Fix 1000 Errors** | $2,960,000             | $621,600         | $843,600           | $4,425,000           |
| **Cost Avoidance**       | **$1,960,000**            | $5,978,400       | $4,856,400         | $8,874,000           |
| **SCQC Investment**      |                             |                  |                    | **$1,868,230**       |
| **ROI**                  |                             |                  |                    | 375.04%              |
HOW DO WE ENFORCE IT?
ENABLING MECHANISMS

- **Measurable/Enforceable Standards**
  - Derived from early assessments

- **Processes**
  - Based on early assessments
  - Continuously updated and refined
  - Benchmarked against industry and other government agency best practices

- **Contracting Language**
  - Specific to all new contracts
  - Integrated into overall Quality Assurance Process
  - Payment Withheld if Standards are Not Met

- **Assessment Frameworks**
  - CAST Health Factors
  - Software Improvement Group (SIG) Maintainability Model with Technical Debt

- **Automated Tools**

- **Rigorous Testing in SCQC Lab**

- **Detailed reports with defects AND recommended corrective actions**

- **Face-to-Face exchanges with product managers and developers**
PROCESS CHANGES FROM CDR TO TRR (WATERFALL)

- To support SCQC code drops to Government starts after CDR and before TRR1
- Government will conduct an inspection before the DIT testing phase at Vendor site
- Government will conduct Smoke Test prior to TRR1
ANALYSIS TOOLS*

- **CAST Application Intelligence Platform (CAST)**
  - Analyze source code (Static Analysis)
  - Supports multiple languages
  - Checks against multiple code quality metrics

- **Fortify (Fortify Source Code Analyzer)**
  - Analyzes source code security vulnerabilities
  - Supports multiple languages
  - Checks against multiple security standards

- **FindBugs**
  - Open Source Java Code Analyzer
  - Static Quality Analysis
  - Used to find additional flaws and remove tool bias.

- **NDepend**
  - Measure software quality against standard code metrics
  - Currently supports *Microsoft’s*.Net
  - Inspects assembly dependencies extensively

- **Sonar**
  - Measure software quality against Microsoft Quality Standards
  - Runs quality analysis based on findings.
  - Inspects assembly dependencies extensively

- **FxCop**
  - Measure software quality against Microsoft Best Coding Practices
  - Inspects .Net managed code assemblies

*Purify Plus, XDepend, *WebLayers
HOW DO WE CONDUCT AND REPORT OUR SECURITY FINDINGS?
SECURITY ANALYSIS AND REPORTS

- FORTIFY is primary tool
- Focus is on Software WEAKNESSES
  - External AND Internal Security Vulnerabilities/Violations
  - Also Identifies Related Code Quality Issues
- Automated Scan
  - Manual review required for removal of “False Positives”
- Reports
  - High Level DISA STIG Summary with Level of Effort
  - Detailed Violation Report with Suggested Remedial Action for CAT 1 Violations
  - Spreadsheet with Individual Defects Identified Down to the Line of Code
  - .FPR file for vendors with the tool
  - Face-to-Face Consultation When Asked
### REPRESENTATIVE FINDINGS

<table>
<thead>
<tr>
<th>File Name</th>
<th>Line #</th>
<th>Category</th>
<th>CWE</th>
<th>STIG</th>
</tr>
</thead>
<tbody>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/controls/EncounterTabUserControl.ascx</td>
<td>142</td>
<td>Cross-Site Scripting: Persistent</td>
<td>CWE ID 79, CWE ID 80</td>
<td>APP3510 CAT I, APP3580 CAT I</td>
</tr>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/patients.aspx</td>
<td>115</td>
<td>Cross-Site Scripting: Persistent</td>
<td>CWE ID 79, CWE ID 80</td>
<td>APP3510 CAT I, APP3580 CAT I</td>
</tr>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/controls/EncounterTabUserControl.ascx</td>
<td>95</td>
<td>Cross-Site Scripting: Persistent</td>
<td>CWE ID 79, CWE ID 80</td>
<td>APP3510 CAT I, APP3580 CAT I</td>
</tr>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/controls/RegistryAdminInfoTabUserControl.ascx</td>
<td>88</td>
<td>Cross-Site Scripting: Persistent</td>
<td>CWE ID 79, CWE ID 80</td>
<td>APP3510 CAT I, APP3580 CAT I</td>
</tr>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/controls/D*****BaseControl.ascx.cs</td>
<td>106</td>
<td>Dead Code: Unused Method</td>
<td>CWE ID 561</td>
<td>APP3050 CAT II</td>
</tr>
<tr>
<td>D*****_v1.0.0.405/BusinessManager/Utilities/Dynamic.cs</td>
<td>142</td>
<td>Dead Code: Unused Method</td>
<td>CWE ID 561</td>
<td>APP3050 CAT II</td>
</tr>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/controls/EncounterTabUserControl.ascx.cs</td>
<td>165</td>
<td>Dead Code: Unused Method</td>
<td>CWE ID 561</td>
<td>APP3050 CAT II</td>
</tr>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/Referral.aspx.cs</td>
<td>1101</td>
<td>Dead Code: Unused Method</td>
<td>CWE ID 561</td>
<td>APP3050 CAT II</td>
</tr>
<tr>
<td>D*****_v1.0.0.405/Business/LINQEntityBase.cs</td>
<td>147</td>
<td>Missing XML Validation</td>
<td>CWE ID 112</td>
<td>APP3510 CAT I</td>
</tr>
<tr>
<td>D*****_v1.0.0.405/Business/SerializerUtil.cs</td>
<td>43</td>
<td>Missing XML Validation</td>
<td>CWE ID 112</td>
<td>APP3510 CAT I</td>
</tr>
<tr>
<td>D<em><strong><strong>_v1.0.0.405/D</strong></strong></em>_UserInterface/App_Code/HtmlUtilities.cs</td>
<td>85</td>
<td>Poor Error Handling: Empty Catch Block</td>
<td>CWE ID 391</td>
<td>APP3120 CAT II</td>
</tr>
<tr>
<td>D*****_v1.0.0.405/Business/LINQEntityBase.cs</td>
<td>758</td>
<td>Poor Error Handling: Empty Catch Block</td>
<td>CWE ID 391</td>
<td>APP3120 CAT II</td>
</tr>
</tbody>
</table>
## RECENT SECURITY FINDINGS

<table>
<thead>
<tr>
<th></th>
<th>LINES OF CODE</th>
<th>CAT 1</th>
<th>CAT 2</th>
<th>OTHER</th>
<th>TOTAL</th>
<th>DENSITY</th>
<th>REMEDIATION EFFORT (HOURS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project 1a</td>
<td>87,393</td>
<td>274</td>
<td>1751</td>
<td>83</td>
<td>2108</td>
<td>2.41%</td>
<td>52</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>110</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>162</td>
</tr>
<tr>
<td>Project 1b</td>
<td>74,623</td>
<td>123</td>
<td>1070</td>
<td>806</td>
<td>1999</td>
<td>1.65%</td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>108</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>133</td>
</tr>
<tr>
<td>Project 2</td>
<td>5,126</td>
<td>23</td>
<td>99</td>
<td>13</td>
<td>135</td>
<td>2.63%</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>Project 3a</td>
<td>43,867</td>
<td>143</td>
<td>352</td>
<td>n/a</td>
<td>495</td>
<td>1.13%</td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>n/a</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>92.1</td>
</tr>
<tr>
<td>Project 3b</td>
<td>22,393</td>
<td>108</td>
<td>366</td>
<td>n/a</td>
<td>474</td>
<td>2.11%</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>44.6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70.6</td>
</tr>
<tr>
<td>Project 4a</td>
<td>27,639</td>
<td>35</td>
<td>58</td>
<td>16</td>
<td>109</td>
<td>0.39%</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>15.5</td>
</tr>
<tr>
<td>Project 4b</td>
<td>15,155</td>
<td>2</td>
<td>15</td>
<td>n/a</td>
<td>17</td>
<td>0.11%</td>
<td>.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.25</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5.5</td>
</tr>
<tr>
<td>Project 4c</td>
<td>37,299</td>
<td>175</td>
<td>513</td>
<td>n/a</td>
<td>688</td>
<td>1.97%</td>
<td>20.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>70.8</td>
</tr>
<tr>
<td>Project 5a</td>
<td>49,664</td>
<td>376</td>
<td>738</td>
<td>39</td>
<td>1153</td>
<td>2.32%</td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>100.8</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>166.8</td>
</tr>
<tr>
<td>Project 5b</td>
<td>35,329</td>
<td>126</td>
<td>485</td>
<td>n/a</td>
<td>611</td>
<td>1.7%</td>
<td>36</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>84.1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>120</td>
</tr>
</tbody>
</table>
HOLD ON TO YOUR HATS!!!!!

Initial Low Defect Densities Achieved Without Running A Single Security Scan
CONTINUING OBSERVATIONS

• **Software with High Security Defect Density**
  – Also Had High Number of Code Quality Defects
  – Demonstrated a Weak to Non-existent Software Architecture
  – Produced Numerous “Valid” False Positives
    • Required Manual Review of the Code
  – Produced Numerous **Invalid** False Positive Claims

• **If Bad Enough, “Broke” the Scanning Tools**
WHAT ABOUT CODE QUALITY?
WHAT IS TECHNICAL DEBT?

• Technical Debt is defined as the cost of fixing application quality problems that, if left unfixed, put the business at serious risk.
• Technical Debt includes only those application quality problems that are highly likely to cause business disruption (DISA STIG estimate to fix CAT I and II issues only). (Not Necessarily True.)
• Due to the fact that the calculation of the Technical Debt is based on the number and severity of structural quality violations in an application, and because these violations are demonstrably linked to business disruption, Technical Debt puts a monetary value on the business risk.
A MORE DETAILED VIEW

Lines of code
- 144,396
  - 222,942 lines
  - 84,153 statements
  - 1,303 files

Classes
- 1,794
- 269 packages
- 9,168 methods
- +7,938 accessors

Violations
- 6,478

Rules compliance
- 85.8%

Comments
- 12.3%
- 17,706 lines
- 22.0% docu. API
- 3,428 commented LOCs

Duplications
- 14.5%
- 32,306 lines
- 570 blocks
- 134 files

Complexity
- 2.7 /method
- 13.8 /class
- 19.0 /file
- Total: 24,742

SIG Maintain. Model
- (A)nalysability: 0
- (C)hangeability: +
- (S)tability: –
- (T)estability: +

Technical Debt
- 6.4%
- $149,738
- 300 man days

No information available on coverage
No information available on design

Quality Index
- 8.0
  - Coding: 0.0 /4.5
  - Complexity: 0.0 /2.0
  - Coverage: 2.0 /2.0
  - Style: 0.0 /1.5

Complexity factor
- 0.0%
- 0 methods
A VERY DETAILED VIEW
SOFTWARE IMPROVEMENT GROUP (SIG)
MAINTAINABILITY MODEL

- **Analyzability:** The capability of the software product to be diagnosed for deficiencies or causes of failures in the software, or for the parts to be modified to be identified. See *maintainability*.*

- **Changeability:** The capability of the software product to enable specified modifications to be implemented. See *maintainability*.*

- **Stability:** The capability of the software product to avoid unexpected effects from modifications in the software. See *maintainability*.*

- **Testability:** The capability of the software product to enable modified software to be tested. See *maintainability*.*

- **Maintainability:** The ease with which a software product can be modified to correct defects, modified to meet new requirements, modified to make future maintenance easier, or adapted to a changed environment.**


** Stolen from ISO 9126
TYPICAL SOLUTION
CAST DASHBOARD

Current Overall Status

Transferability 2.83 2.68
Changeability 2.09 2.35
Robustness 2.07
Performance 1.6
Security 2.35 2.28

Requires Significant Improvement

Financial Information

Debt $3,744,700
Debt added $3,744,700
Debt removed $0

Statistics

Critical Violations 634
- per File 0.44
- per KLOC 2.05
Complex Objects 1,276
- w/ violations 61

Current Snapshot 10-23-2012
First Snapshot 10-23-2012
Snapshots 1
<table>
<thead>
<tr>
<th>Rule Name</th>
<th>Rule Description</th>
<th>Number of Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declare as Static all methods not using instance fields</td>
<td>All methods that don't use instance fields and/or methods will be reported except if they have been extended in a subclass or inherited from their parents.</td>
<td>3150</td>
</tr>
<tr>
<td>Avoid instantiations inside loops</td>
<td>All artifacts include all methods and constructors.</td>
<td>366</td>
</tr>
<tr>
<td>Avoid direct definition of JavaScript Functions in a Web page</td>
<td>All JavaScript Functions should be defined in a JS file and not directly in the Web page (HTML, HTML, JSP).</td>
<td>174</td>
</tr>
<tr>
<td>Avoid indirect String concatenation inside loops</td>
<td>All methods that call indirectly a method that uses string concatenation in loops (for, while, do-while) at a depth level less than a depth level will be reported.</td>
<td></td>
</tr>
<tr>
<td>Avoid method invocation in a loop termination expression</td>
<td>Finds all loops (for, while, do-while) termination that call a method except the following methods coming from Iterator, ListIterator, Enumeration, StringTokenizer, ResultSet classes:</td>
<td>96</td>
</tr>
<tr>
<td>* java.util.Enumeration.hasMoreElements()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.Enumeration.nextElement()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.Iterator.hasNext()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.Iterator.next()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.ListIterator.hasNext()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.ListIterator.next()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.ListIterator.previous()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.ListIterator.previousIndex()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.sql.ResultSet.isAfterLast()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.sql.ResultSet.isBeforeFirst()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.sql.ResultSet.isLast()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.sql.ResultSet.isFirst()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.sql.ResultSet.next()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.sql.ResultSet.previous()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.StringTokenizer.hasMoreElements()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.StringTokenizer.nextElement()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.StringTokenizer.nextToken()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.StringTokenizer.nextElement()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* java.util.StringTokenizer.nextToken()</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Note that the filter applies only on the short name of the method: i.e. for example all method next() will not be reported as a violation even if it does not belongs to java.util.Iterator, java.util.ListIterator or java.sql.ResultSet)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>for C## language</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* ReadLine</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Read</td>
<td></td>
<td></td>
</tr>
<tr>
<td>* Read</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
EFFORT DENSITY
CONVERTING TECHNICAL DEBT TO EFFORT DENSITY

• Technical Debt is indirectly derived from the number of violations per thousands of lines of code (violations per KLOC) – the higher the level of business disruption, the higher the severity of the violation.

• The Security & Quality Technical Debt provide LOE hours to fix CAT I and II violations.

• Effort Density is a suitable method to compare Technical Debt evaluation among projects because the code size may vary.

  Effort Density = Security & Quality Technical Debt LOE hours / Total LOC

• Recommended target for Effort Density is ≤ 0.02.
EFFORT DENSITY - EVALUATION RESULT

Evaluation Result by Effort Density
(LOE Hrs/LOC)

<table>
<thead>
<tr>
<th>#</th>
<th>Project Evaluation Name</th>
<th>Security Evaluation (Fortify)</th>
<th>Quality Evaluation (Sonar)</th>
<th>Grand Total</th>
<th>Evaluation Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Level of Effort Evaluation Result (Security Technical Debt)</td>
<td>Level of Effort Evaluation Result (Quality Technical Debt)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Lines of Code</td>
<td>Critical</td>
<td>High</td>
<td>Medium</td>
</tr>
<tr>
<td>1</td>
<td>Project 1 Drop 2 (.Net)</td>
<td>4888</td>
<td>4</td>
<td>57</td>
<td>7</td>
</tr>
<tr>
<td>2</td>
<td>Project 1 Drop 2 (JAVA)</td>
<td>44542</td>
<td>18</td>
<td>1368</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>Project 2 Drop 4</td>
<td>37101</td>
<td>65</td>
<td>13</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>Project 3 Drop 2</td>
<td>18272</td>
<td>41</td>
<td>112</td>
<td>56</td>
</tr>
<tr>
<td>5</td>
<td>Project 4 Drop 2</td>
<td>37551</td>
<td>8</td>
<td>113</td>
<td>211</td>
</tr>
</tbody>
</table>
SELECTED FINDINGS (POSITIVE)

• Test Cycles for “SCQC-Compliant” Products Are Being Reduced
• Few or even NO “False Positives” in “good” Code
• Vendors Adapting Practices
  – Their Best Interest
  – Competitive for business elsewhere in the government
OTHER FINDINGS (NEGATIVE)

• Parent-child Dependencies In Bad Code
  – Tight Coupling Between Assemblies
  – Produces “Multiple” Reports Of A Single Security Defect
• Spending More Time Avoiding Rather Than Just Fixing The Defects
  – Using False Positives As The Excuse For Not Fixing Specific Problems.
  – Most “False Positives” From Security Scans Are, In Reality, Code Quality Defects And Are In Themselves (For The Most Part) “False Claims.”
CURRENT PLANS

• Move to a More Agile Development Process
• Mandate Use of Integrated Development Environment
  – Exploit Built-In Refactoring and Other Capabilities
• Continuous Integration
RECOMMENDATIONS

• If You Haven’t Done So Already, Create Your Own Internal SCQC/T&IVV Team
• Write Code Quality and Security Into All Contracts
• Withhold Payment If Standards Are Not Met
• Fire People and Organizations Who Are Unwilling To Play By The New Rules
• Ruthless Implementation of Static Code Analysis Should ALWAYS Precede Static and Other Types of Security Analyses
  — Supported by OWASP, CWE, DACS, NDIA and DHS
Questions?