Adhering to Coding Standards and Finding Bugs in Code Automatically

Presenters:
Muhammad Wasim
Hammad Ali Butt

Al-khawarzimi Institute of Computer Science
Agenda

- Introduction
- Software Quality Assurance
- Code: The base for Quality
- Code: Writing it
- Code: Reviewing it
- How Review tools help?
- Selecting a Code Review Tools
- How they compare to each other?
- Conclusion
Introduction

- Software Quality
- Quality Attributes
  - Maintainability
  - Testability
  - Reusability
  - Correctness
  - Efficiency
  - Usability
  - Flexibility
  - Portability
  - Interoperability
  - Reliability
  - Integrity
- Code: The Base for Quality
  - Software Quality comes with a lot of benefits
SQA and Benefits

- **Development/Technical:**
  - Increase Readability
  - Easy to locate problem area
  - Performance Enhancements
  - Compliance between Specs & Code
  - Criteria for software acceptance

- **Management:**
  - Better Progress visibility
  - Better decision criteria
  - Reduced Maintenance cost
Code: The Base for Quality

- Better Coding Styles (Structured to OOP)
- Commenting of Code
- Coding Conventions
- Design Patterns
- Low Coupling
- High Cohesiveness
- Resource Management
- Remove Repetition of Code
Code: Writing it

- Well Managed Code
- Well Written Code
- Standardized Code
- Defensive Coding

- An ideal programmer won’t leave any bugs for compiler or QA team.
Code: Reviewing It

- Review Meetings
- Peer Review

**Benefits:**
- Code Optimization
- Reduced Cost
- Programmer Improvement
- Static Code Analysis is used for Code Review.
Code Review on a Typical Project

![Bar Chart]

Before Code Review

- Bugs Remaining: 463

After Code Review

- After Development
  - Bugs Remaining: 321
- After QA/Test
  - Bugs Remaining: 194

Cost of fixing bugs: $174k

Cost of 194 latent bugs: $194k

Total Cost: $368k

http://smartbear.com/docs/articles/before-code-review.jpg
Code Review on a Typical Project

http://smartbear.com/docs/articles/after-code-review.jpg
Code Review & Issues

- Human is error prone.
- Tedious Job
- Slow Process

Solution

“Automatic Code Review”
How Review tools help?

- The use of continuous **automatic** static code **reviews** is a really helpful for quality code.

- We all make **mistakes** and the more you code the more that will happen.

- *Static code reviews* aimed at **eating bugs** are unbiased and neutral.

- Static code reviews are viable. You can get rid of quite a lot of near surface defects quickly.
What’s the benefit?

- Enforce coding convention
- Uniform coding structure
- Ensure 100% checking
- Reduce review time
- Developer development
- Cost effective
Automated Review Tools

Vulnerabilities discovered after these tests:

- Manual Source Code Review: 60%
- Black Box Penetration Test: 30%
- Automated Code Audit: 10%
Popular Code Analysis Tools for Java

- **Checkstyle**
  - Besides some static code analysis, it can be used to show *violations* of a configured *coding standard*

- **FindBugs**
  - An open-source static *bytecode analyzer* for Java (based on Jakarta BCEL) from the University of Maryland.

- **PMD**
  - A static rule set based Java *source code analyzer* that identifies potential problems.
Popular Code Analysis Tools for Java

- Hammurapi
  - (Free for non-commercial use only) versatile code review solution.

- Sonar
  - A platform to manage source code quality (checkstyle, PMD and find bug)
How a Review Tool can be Selected?

- Ability to modify rules
- Ability to implement coding standard at various levels
- Importing/Exporting rules
- Integration with the IDE
Configuration of tools

- Customization of two commonly used tool on a popular IDE

- Such that:
  - CheckStyle
  - FindBug
Configuration CheckStyle (NetBeans)
Configuration FindBug (NetBeans)
A Standard XML file that defines a set of modules that are used to verify source code.

The Checks which want enable are define in module tags

In hierarchy of module root module called the Checker module.
Modules can also contain sub-modules.

TreeWalker module parses individual Java source code files.

The majority of modules must be nested in the TreeWalker module.
A simple Checkstyle configuration is probably as

```xml
<?xml version="1.0"?>
<!DOCTYPE module PUBLIC
   "-//Puppy Crawl//DTD Check Configuration 1.2//EN"
   "http://www.puppycrawl.com/dtds/configuration_1_2.dtd">

<module name="Checker">
   <module name="TreeWalker">
   </module>
   <property name="tabWidth" value="4"/>
   <property name="charset" value="UTF-8"/>
   <module name="JavadocMethod"/>
   <module name="JavadocVariable">
      <property name="scope" value="protected"/>
   </module>
   <module name="AvoidStarImport"/>
</module>
<module name="PackageHtml"/>
</module>
```
Customizing CheckStyle  (Example)

Line Length
<module name="LineLength">
  <property name="severity" value="ignore"/>
  <property name="max" value="70"/>
</module>

Documentation
<module name="JavadocType">
  <property name="authorFormat" value="\S"/>
</module>

Package Name Convention
<module name="PackageName">
  <property name="format" value="^[a-z]+(\.[a-z][a-z0-9]*)*$"/>
</module>
1- Comparison of some Bug Tools

<table>
<thead>
<tr>
<th>Name</th>
<th>Version</th>
<th>Input</th>
<th>Interfaces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandera</td>
<td>0.3b2</td>
<td>Source</td>
<td>CL, GUI</td>
</tr>
<tr>
<td>ESC/Java</td>
<td>2.0a7</td>
<td>Source</td>
<td>CL, GUI</td>
</tr>
<tr>
<td>FindBugs</td>
<td>0.8.2</td>
<td>Bytecode</td>
<td>CL, GUI, IDE, Ant</td>
</tr>
<tr>
<td>JLint</td>
<td>3.0</td>
<td>Bytecode</td>
<td>CL</td>
</tr>
<tr>
<td>PMD</td>
<td>1.9</td>
<td>Source</td>
<td>CL, GUI, Ant, IDE</td>
</tr>
</tbody>
</table>

# 2- Comparison of some Bug Tools

<table>
<thead>
<tr>
<th>Bug Category</th>
<th>Example</th>
<th>ESC/Java</th>
<th>FindBugs</th>
<th>JLint</th>
<th>PMD</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Null dereference</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Possible deadlock</td>
<td>✓ ✓</td>
<td>✓ ✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Exceptions</td>
<td>Possible unexpected exception</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Array</td>
<td>Length may be less than zero</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mathematics</td>
<td>Division by zero</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Conditional, loop</td>
<td>Unreachable code due to constant guard</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td>✓ ✓</td>
</tr>
<tr>
<td>String</td>
<td>Checking equality using == or !=</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Object overriding</td>
<td>Equal objects must have equal hashcodes</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td>✓ ✓</td>
</tr>
<tr>
<td>I/O stream</td>
<td>Stream not closed on all paths</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unused or duplicate statement</td>
<td>Unused local variable</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td>✓ ✓</td>
</tr>
<tr>
<td>Design</td>
<td>Should be a static inner class</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Unnecessary statement</td>
<td>Unnecessary return statement</td>
<td>✓ ✓</td>
<td></td>
<td></td>
<td>✓ ✓</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>5</td>
<td>8</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>


Al-khawarizimi Institute of Computer Science
DEMO
Conclusion

- SQA is an important task for developing error free code.
- Few errors, although not caught by IDE, may create problems later.
- Using SCA tools makes it easy to apply the boring task of code review.
- Organizations should motivate SQA department and even to programmers to use such kind of tools to improve the performance, quality and maintenance of a software.