Insight V4 Metrics

2000 West Park Drive
Westborough MA 01581 USA
Phone: 508 389 7300 Fax: 508 366 9901

virtusa®
Accelerating Business Outcomes

Insight V4 Metrics Definitions

virtusa

What does ERA Insight Report?

- Size Metrics
- Code Metrics
- Defects
  - Style Defects
  - Technical Defects
  - Security Defects
- Weighted Defect Density
- Reuse Metrics
- Churn Metrics
- Unit Test Metrics
- Build Stability Metrics

virtusa

Size metrics

Why size metrics?

- ‘Size’ is a predominant metric in software engineering
- Can be the fundamental mode of insight to a project
- Can expose ‘code smells’ or ‘anti-patterns’
- Can be used for estimations for maintenance
- Can be used to calculate productivity of Greenfield Development projects

Size metrics

What sizes do we measure?

- **Effort size**
  - "How much of effort has been put in for the implementation?"
  - Calculated by the ***GTO-Metrics*** tool.
    - Counts physical and logical lines.
    - In sync with SEI standards for sizing.

- **Work Size**
  - "How much of business functionality has been implemented?"
  - Calculated using the ***Vsize*** tool.
    - Calculates 'Backfired Function Points'
    - Follows the 'Gearing Factors' derived by QSM
    - Identifies multiple content types and languages within a single source file.

Size metrics

Effort Size measures:

- **Total Lines of Code**
  - The number of lines visible from a physical layout perspective.
- **Total Statements (Logical Lines of Code)**
  - Number of lines from a logical perspective.
- **Average methods per class**
  - Gives an indication of how much functionality is cramped inside classes.
  - Can identify ‘God Classes’, ‘Spaghetti Code’ etc.
  - Can see in advance what classes may become maintenance nightmares.
  - Can check for redundant functionality and ability to split.
- **Average statements per method**
  - Gives insight to how ‘cohesive’ a method is.
  - “is a method doing too much work?”
  - Can identify good candidates to make use of reuse.

Size metrics

Work Size measures:

- **Total Function Points**
  - Function Points is a concept which unifies sizing across languages.
  - An ISO recognized software metric.
  - Looks with the abstraction of business functionality rather than effort (LOC)
  - '**Backfired**' using '**Gearing Factors**'

- Eg:

Language : Java (G.F : 59)	Language : SQL (G.F : 35)
Lines of Code = 2500	Lines of Code = 2500
Function Points = 2500 / 59	Function Points = 2500 / 35
= 42.37	**= 71.42**

Comparable and possible to normalize similar metrics for different languages!

What does ERA Insight Report?

- **Size Metrics** ✓
- Code Metrics
- Defects
  - Style Defects
  - Technical Defects
  - Security Defects
- Weighted Defect Density
- Reuse Metrics
- Churn Metrics
- Unit Test Metrics
- Build Stability Metrics

Code Metrics

- **Why code metrics?**
  - Size alone cannot give a full picture.
  - Size measures can be used as parameters to derive additional metrics.
  - There is more meaning when measurement is done focusing on the content type.
  - Captured using the ***GTO-Metrics*** tool.

Code Metrics

- **Code size measures**
  - **Comment Ratio**
    - Total comment ratio
    - Documentation comment ratio
    - Standard comment ratio
  - **Complexity**
    - Maximum complexity
    - Average complexity
  - **Maintainability Index**
    - Comment weighted
    - Without comments

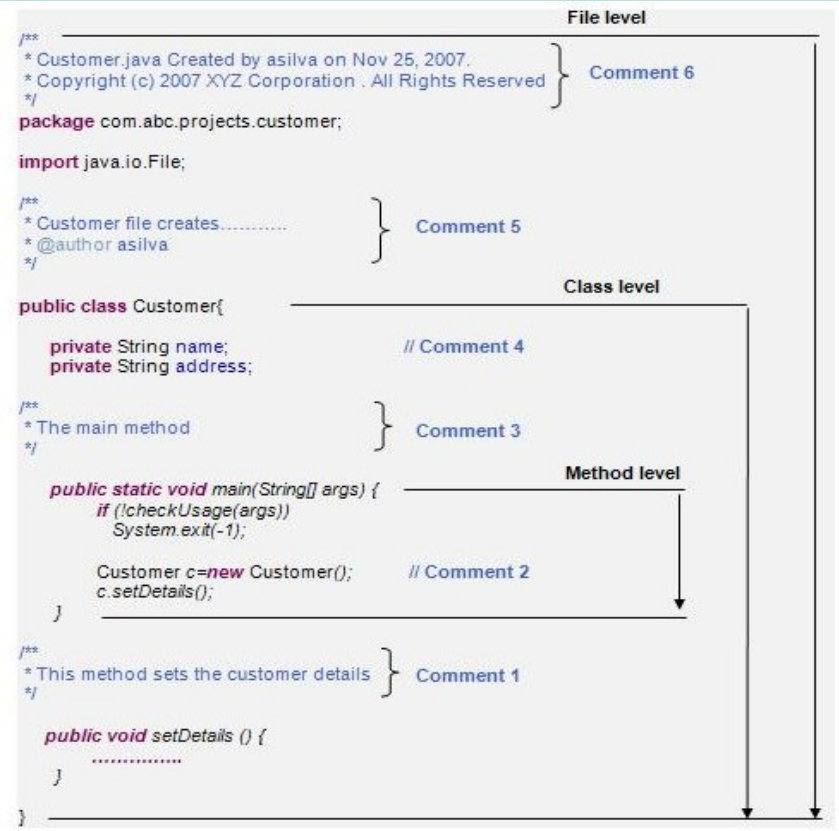
• The maintainability of the code base
• The steepness of the learning curve
• The attention to detail

• The 'readability' of the code base
• The easiness to test functionality

• Considers both Cyclomatic Complexity and Halstead Complexity to evaluate maintainability

Code Metrics

- **Comment Ratio**
  - CR = Comments / (Comments + Code)



Doc. Comments = Comment1+Comment3+Comment5+Comment6

Std Comments = Comment2+ Comment4

Doc. Comment Ratio = Doc.Comments / LOC
Std. Comment Ratio = Std.Comments / LOC

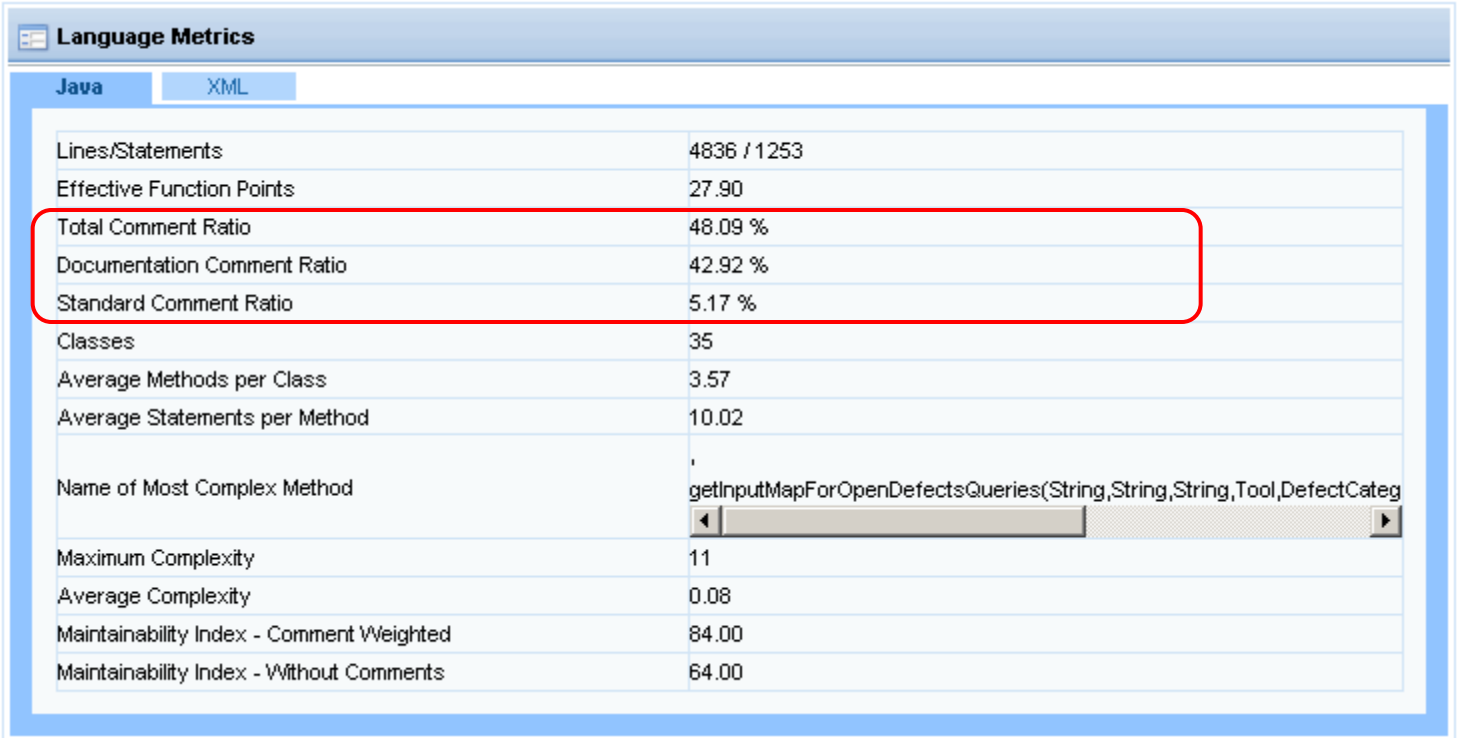
Total Comment Ratio = Doc. Comment Ratio + Std. Comment Ratio

Severity	Range
S2	5% - 14%

virtusa

Code Metrics

- **Comment Ratio**
  - Eg:



Language Metrics

Java | XML

Lines/Statements	4836 / 1253
Effective Function Points	27.90
Total Comment Ratio	48.09 %
Documentation Comment Ratio	42.92 %
Standard Comment Ratio	5.17 %
Classes	35
Average Methods per Class	3.57
Average Statements per Method	10.02
Name of Most Complex Method	, getInputMapForOpenDefectsQueries(String,String,String,Tool,DefectCateg
Maximum Complexity	11
Average Complexity	0.08
Maintainability Index - Comment Weighted	84.00
Maintainability Index - Without Comments	64.00

Code Metrics

- **Benefits of maintaining a higher Comment Ratio**
  - The maintainability of the code base
  - The steepness of the learning curve
  - The attention to detail

Code Metrics

- **Cyclomatic Complexity**
  - The number of independent paths in a method.
  - Captured by counting the decision points in a method.
    - Eg: if, for, while, case, *[else]*

The cyclomatic complexity **M** is then defined as:[

$$M = E - N + 2P$$

where

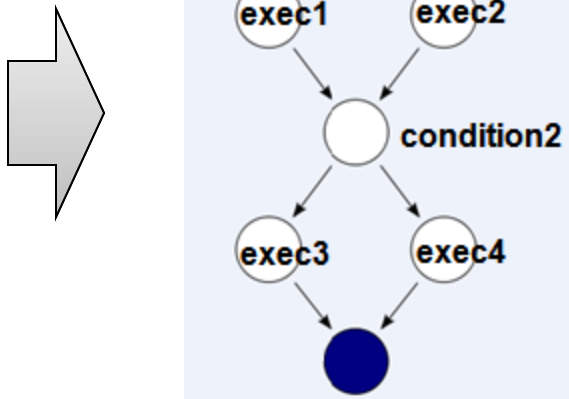
E = the number of edges of the graph
N = the number of nodes of the graph
P = the number of connected components

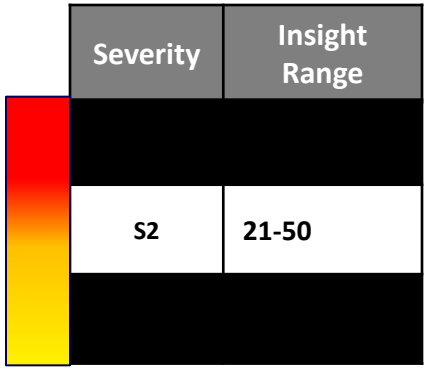
Code Metrics

An alternative formulation is to use a graph in which each exit point is connected back to the entry point. In this case, the graph is said to be *strongly connected*, and the cyclomatic complexity is defined as:

$$M = E - N + P$$

```
method() {

if( condition1)
      exec1();
else
      exec2();

if( condition2)
      exec3();
else
      exec4();
}
```





Cyclomatic Complexity

Code Metrics

Cyclomatic Complexity = "The number of decision points" + 1

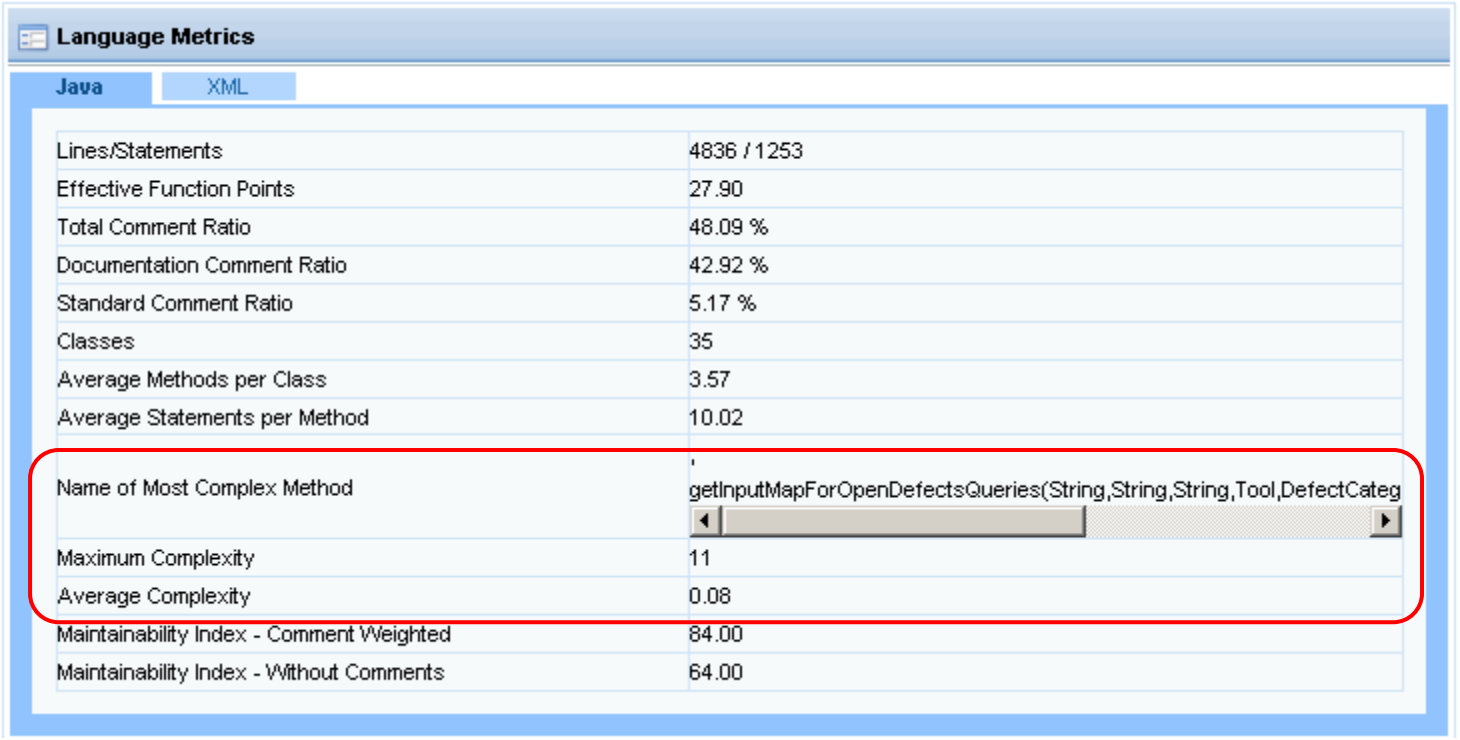
Conditionals and loops add to the complexity of a method. Each additional if, case, while, etc, adds 1 to your Cyclomatic Complexity score because you're adding another potential path through the method.

In general,

add 1 for each if statement.
add 1 for each for statement.
add 1 for each while loop
add 1 for each do-while loop.
add 1 for each && (an implied if statement).
add 1 for each || (an implied if statement).
add 1 for each ? (an implied if statement).
add 1 for each . (an implied if statement).
add 1 for each case statement.
add 1 for each default statement.
add 1 for each catch statement.
add 1 for each finally statement.
add 1 for each continue statement.

Code Metrics

- **Cyclomatic Complexity**
  - Eg:



Language Metrics

Java | XML

Lines/Statements	4836 / 1253
Effective Function Points	27.90
Total Comment Ratio	48.09 %
Documentation Comment Ratio	42.92 %
Standard Comment Ratio	5.17 %
Classes	35
Average Methods per Class	3.57
Average Statements per Method	10.02
Name of Most Complex Method	getInputMapForOpenDefectsQueries(String,String,String,Tool,DefectCateg
Maximum Complexity	11
Average Complexity	0.08
Maintainability Index - Comment Weighted	84.00
Maintainability Index - Without Comments	64.00

Code Metrics

- **Benefits of knowing Cyclomatic Complexity**
  - Improving the ‘readability’ of the code base by reducing Cyclomatic Complexity
  - The easiness to test functionality: Determining the number of test cases that are necessary to achieve thorough test coverage of a particular module.
  - A positive correlation between cyclomatic complexity and defects: modules that have the highest complexity tend to contain higher number of defects
  - Module with higher complexity would tend to have lower cohesion

Code Metrics

- **How to reduce Cyclomatic Complexity?**
  - Reduce the number of unique decisions in a given method

Code Metrics

- **Maintainability Index**
  - Maintainability Index is an SEI defined metric and considers both Cyclomatic Complexity and Halstead Complexity– it is a useful indicator of the maintainability of the codebase.

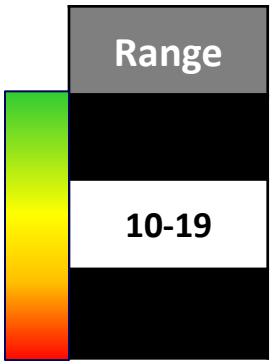
Comment Weighted Maintainability Index

$$MI = 171 - 5.2 \times \ln(aveE) - 0.23 \times aveV(g') - 16.2 \times \ln(aveLOC) + 50 \times \sin(\sqrt{2.4 \times perCM})$$

Maintainability Index without Comments

$$MI = 171 - 5.2 \times \ln(aveV) - 0.23 \times aveV(g') - 16.2 \times \ln(aveLOC)$$

  - GTO-Metrics tool uses Microsoft's variation of maintainability index that sets the range between 0 and 100.

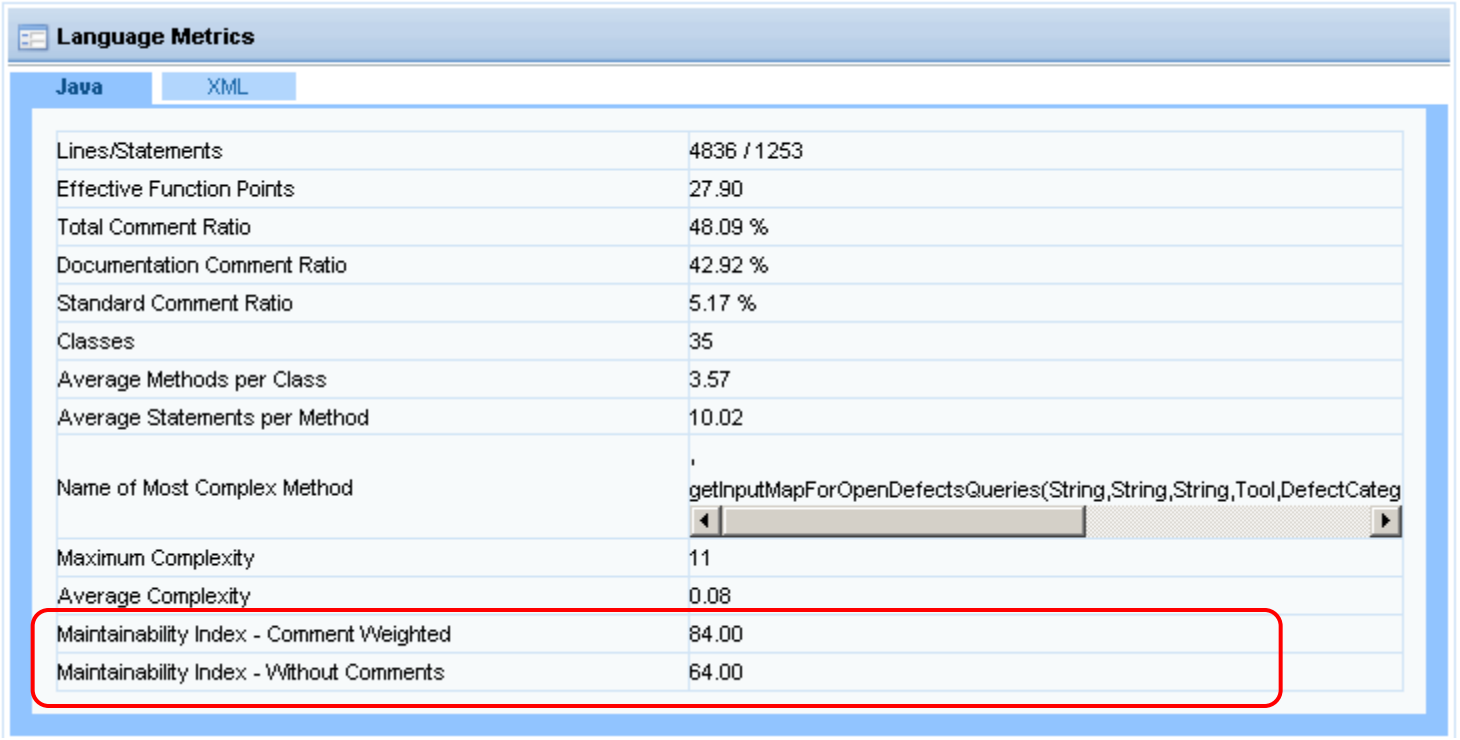


aveE - average Halstead Effort per module ,
aveV - average Halstead Volume per module,
aveV(g') - average extended Cyclomatic complexity per module,
aveLOC - average lines of code per module,
perCM - average percent of lines of comments per module.

virtusa

Code Metrics

- **Maintainability Index**
  - Eg:



Language Metrics

Java | XML

Lines/Statements	4836 / 1253
Effective Function Points	27.90
Total Comment Ratio	48.09 %
Documentation Comment Ratio	42.92 %
Standard Comment Ratio	5.17 %
Classes	35
Average Methods per Class	3.57
Average Statements per Method	10.02
Name of Most Complex Method	getInputMapForOpenDefectsQueries(String,String,String,Tool,DefectCateg
Maximum Complexity	11
Average Complexity	0.08
Maintainability Index - Comment Weighted	84.00
Maintainability Index - Without Comments	64.00

Code Metrics

- **Benefits of knowing Maintainability Index**
  - A high value means better maintainability
  - Easily understandable code

- **How to improve the Maintainability Index?**
  - Reduce Cyclomatic Complexity
  - Increase Comment Ratio
  - Reduce the number of operators and operands per statement

What does ERA Insight Report?

- **Size Metrics** ✓
- **Code Metrics** ✓
- Defects
  - Style Defects
  - Technical Defects
  - Security Defects
- Weighted Defect Density
- Reuse Metrics
- Churn Metrics
- Unit Test Metrics
- Build Stability Metrics

virtusa

Defects

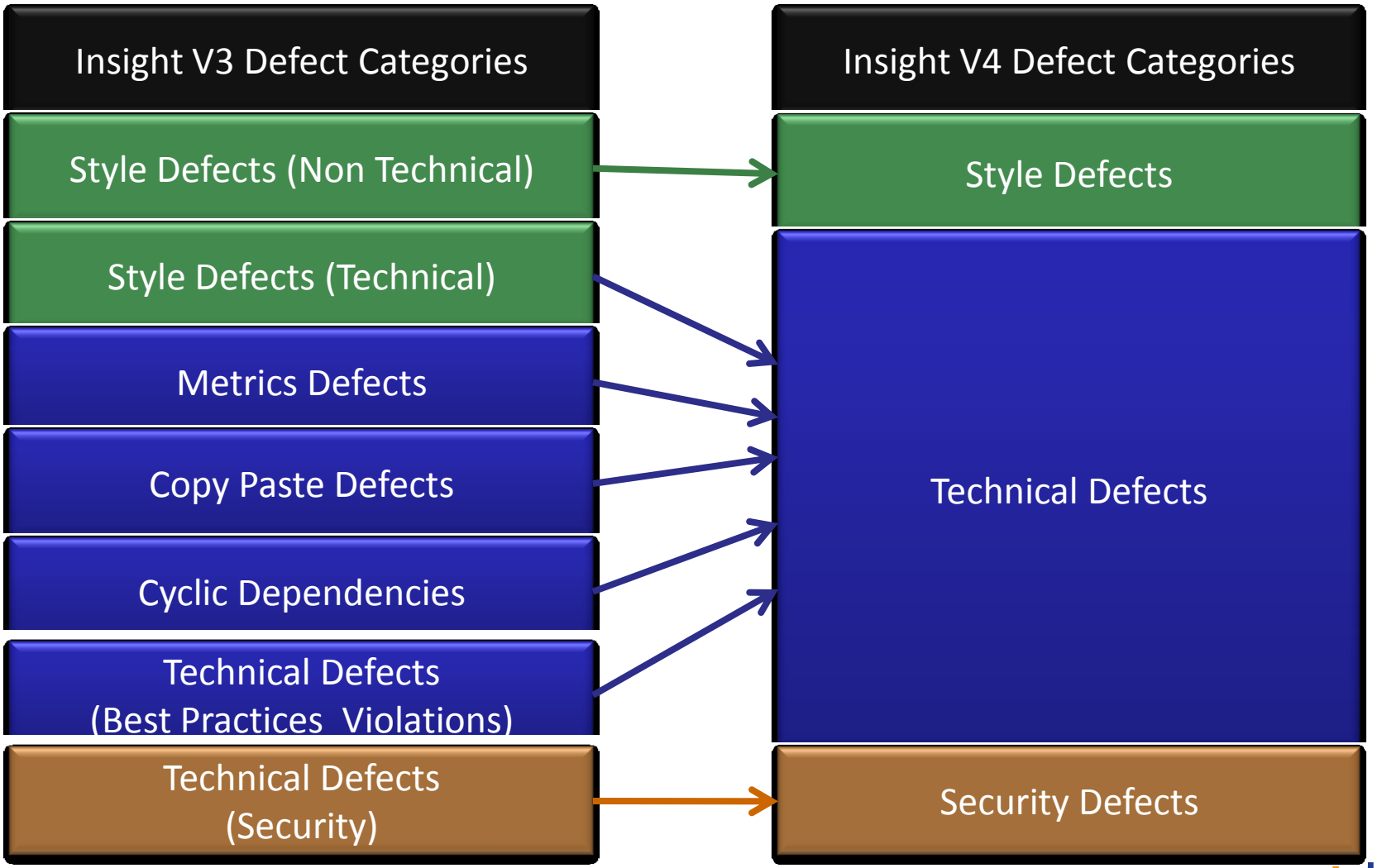
ERA Insight captures defects based on the rules specified in code analysis tools. Based on the types of rules, following categories of defects have been defined.

- Style Defects
- Technical Defects
- Security Defects

ERA Insight uses three Severity levels to indicate the severity of the defects

- Severity 1 (S1) – Highest Severity
- Severity 2 (S2) – Medium Severity
- Severity 3 (S3) – Low Severity

Comparison of Insight V3 and V4 Defects Categories



Style Defects

This category can check the following types of issues:

- Does the code adhere to standard coding, formatting, commenting, documentation and naming conventions?
- Does the copyright info exist in the headers of all the source files?
- Is indenting (tabs and spaces) being used consistently as defined in the coding standards?
- Uses CheckStyle for Java / FxCop for .Net

Why is this important?

- Obtain consistency across the board
- Automate both the adherence to the coding standard as well as automatically catch when standards are violated
- Automating the mundane gives more time for the important!

Fixing Style Defects

How to minimize your style defects?

- Define your coding standard
- If your main IDE is Eclipse
  - Prepare a CheckStyle template for the coding standard
  - Prepare a Code Formatter template and a Code Cleanup template for Eclipse
  - Install CheckStyle plug-in for Eclipse
- If your main IDE is Visual Studio
  - On the **Tools** menu, click **Options**. Click **Text Editor** and configure required properties under proffered language (e.g. C#) -> Formatting
- Before you check-in your code, run a code formatter and a cleanup
- Use Insight to see the details of the defects that the formatter and cleanup does not correct.

Technical Defects

This category can check for violations of coding best practices and common design mistakes (code smells).

- In other words, an automated level 2 code review.

Examples of defects caught:

- Incorrect use of exceptions (e.g. swallowing)
- Coding flaws that can throw null pointer exceptions at run-time
- Common mistakes with Singleton and Thread usage
- Sub-optimal concatenation of strings inside loops etc. etc.
- Code duplications
- Metrics violations

Technical Defects (cont…)

Why is this important?

- Catch defects and potential defects early in the life-cycle!
- Great source of knowledgebase of common mistakes and resolutions – good hands-on training for new resources (an expert code reviewer at their fingertips)
- Past data within Virtusa has shown a strong correlation between code defects (overall code metrics for that matter) and QA defects

Technical Defects Examples

```
19    public static void doInvalidCast() {
20        Object fileReader = new FileWriter();
21        Entry invalidCastedObject = (Entry) fileReader;
22
23        System.out.println(invalidCastedObject);
24
25        if (fileReader instanceof String) {
26            System.out.println("This line is never printed");
27        }
28    }
```

Expected com.virtusa.gto.pb.core.Entry

[BC] Impossible cast [BC_IMPOSSIBLE_CAST]

This cast will always throw a ClassCastException.

Defects Details

Short Description

Impossible cast from com.virtusa.gto.pb.io.FileWriter to com.virtusa.gto.pb.core.Entry in com.virtusa.gto.pb.core.TestClass.doInvalidCast()

Severity : High

Tool Name : FindBugs

Automatically Assigned To : smario

Manually Assigned To :

Code Unit(s)

	Package	File	Class	Method	Type
1	com.virtusa.gto.pb.core	com/virtusa/gto/pb /core/TestClass.java	TestClass	doInvalidCast():void	Method

Start Line : 21

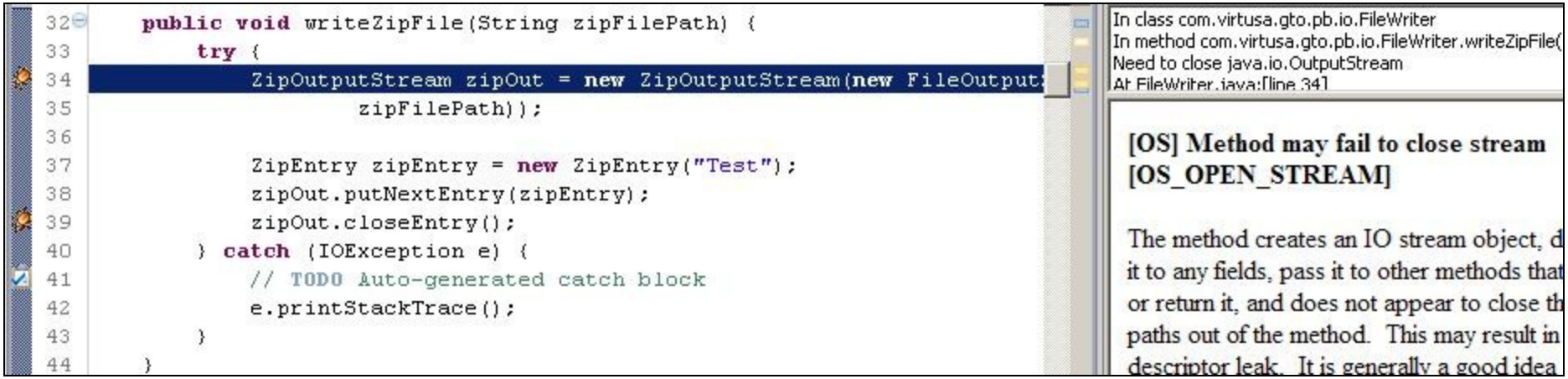
End Line : 21

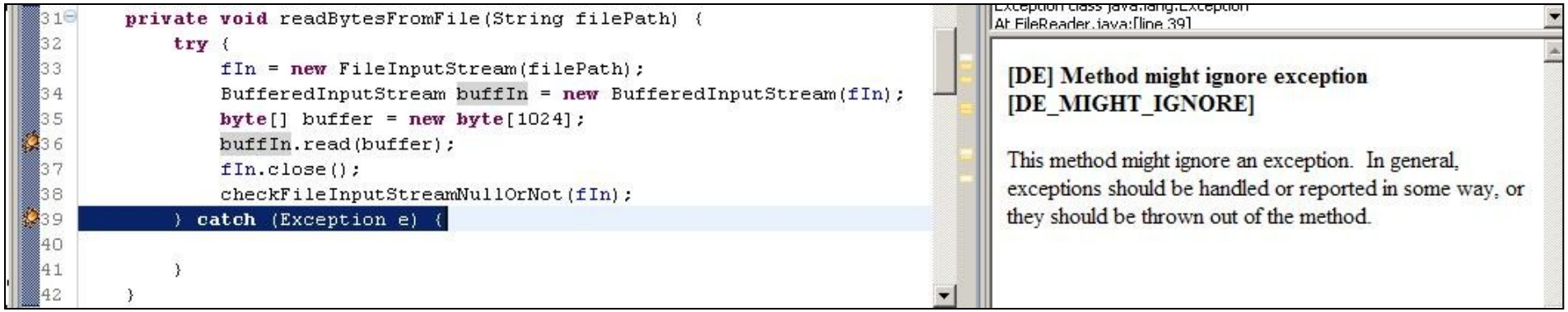
Detailed Description:

Start Line : 21 End Line : 21

This cast will always throw a ClassCastException.

Technical Defects Examples (cont…)





Technical Defects Examples - Duplications

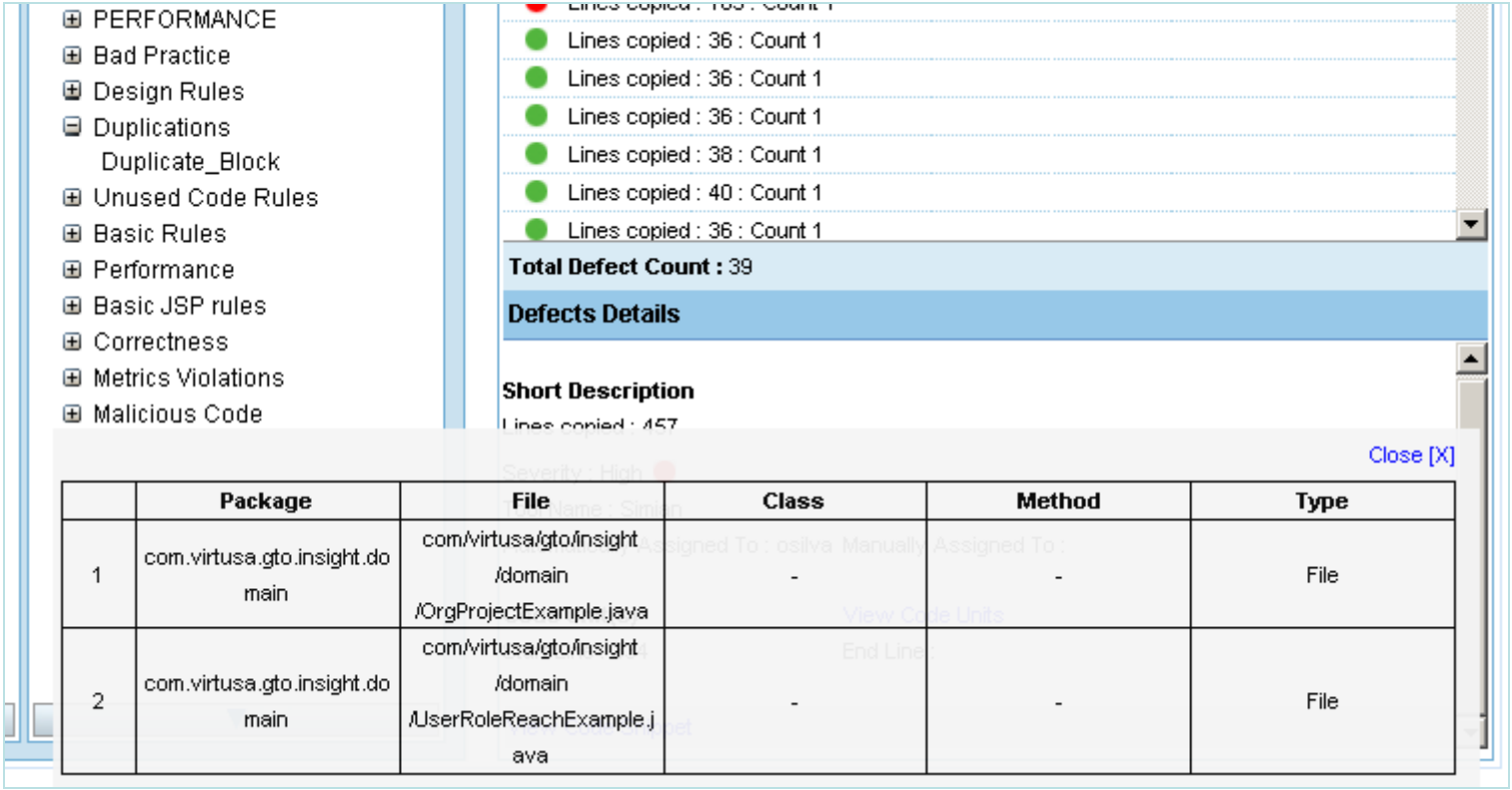
- Captures the code segments copied and pasted across the given code base.
- Severity is decided depending on the amount of tokens (i.e. the language content) of the copy-paste instance.
- Schema for deciding the severity for duplications

Tokens copied and pasted (t)	Insight Severity
t > 100	S1
75 < t <= 100	S2
50 < t <= 75	S3

- P3 duplication capturing is disabled by default as that level of copy paste instance are common and hard to fix

Technical Defects Examples – Duplications (cont..)

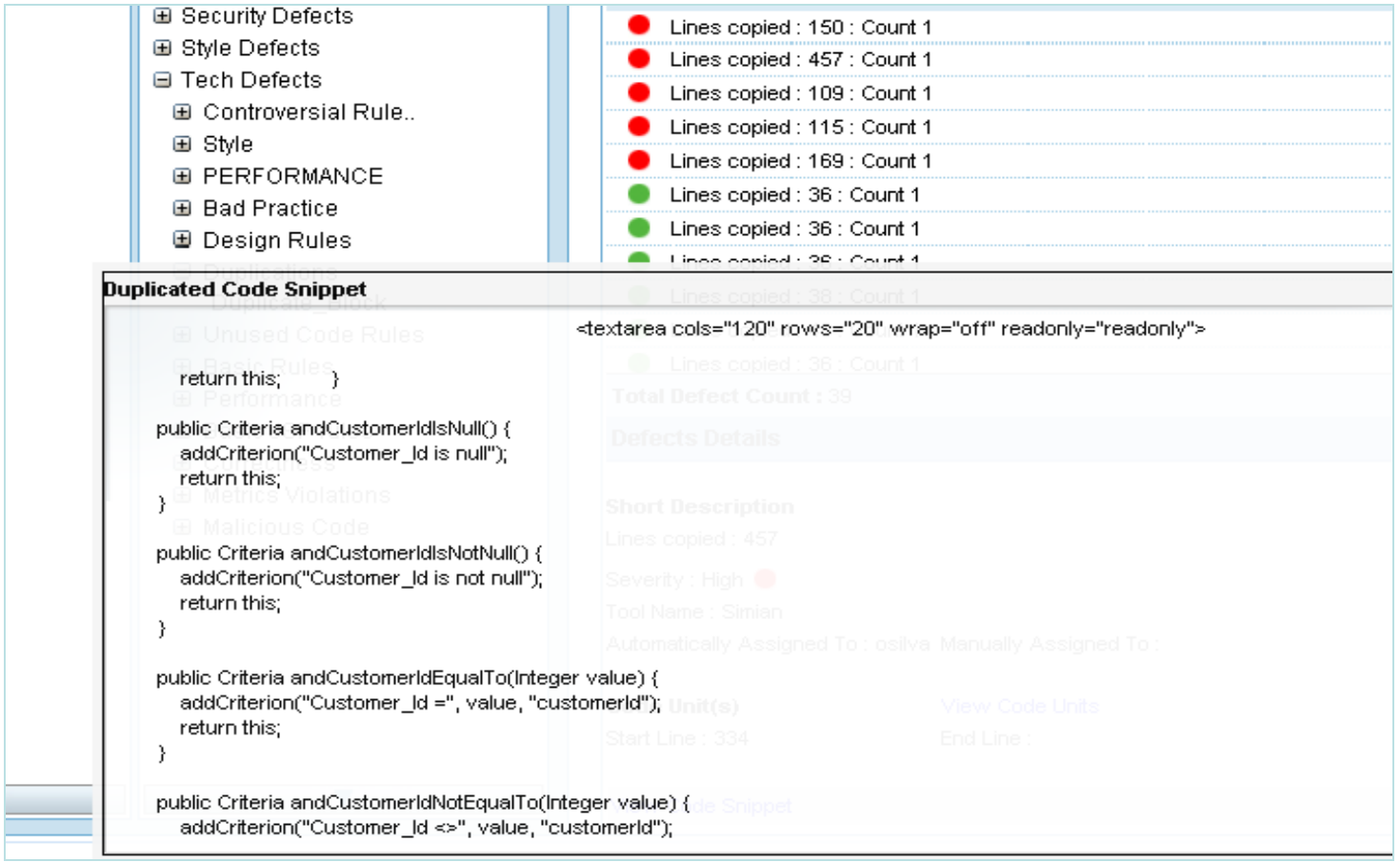
- Duplicated code units as seen in Insight Dashboard – Defect Details



	Package	File	Class	Method	Type
1	com.virtusa.gto.insight.domain	com/virtusa/gto/insight/domain/OrgProjectExample.java	-	-	File
2	com.virtusa.gto.insight.domain	com/virtusa/gto/insight/domain/UserRoleReachExample.java	-	-	File

Technical Defects Examples – Duplications (cont..)

- Duplicated code snippet as seen in Insight Dashboard – Defect Details



Technical Defects Examples – Duplications (cont..)

Why is this important ?

- Code maintenance becomes more difficult and expensive
- If a change has to be done to a copied and pasted instance, you have to do it for all the copied and pasted instances
  - Takes a lot of effort
  - Failing to apply the change in a single instance will cause functional flaws in the application
- Copy-paste instances increases the LOC of the code base, thus giving a wrong impression on the effort (derived by the LOC or Function Points)
- You are not reusing your code
- May lead to design violations
  - Copied and pasted code in multiple packages may lead to cyclic dependencies

Technical Defects Examples – Duplications (cont..)

How to minimize Copy Paste Defects

- Refactor your code base

Refactoring options that can be used

- Extract methods
- Move methods to super classes
- Extract classes or super classes
- Extract interfaces
- Define your local component libraries

Tips and Tricks

- Use Eclipse or Visual Studio featured for refactoring
- Do your unit tests after refactoring

Technical Defects Examples - Metrics Violations

- Reported when Size and Code Metrics exceed their thresholds.
- Aimed at resolving the bad practices at the earliest.
- Threshold Summary:

Priority	Cyclomatic Complexity of Methods	Comment Ratio of Files	Statements per Method	Statements per Class
S1	> 50	< 5%	> 400	> 1000
S2	> 20 and <= 50	>= 5% and < 15%	> 300 and <= 400	> 600 and <= 1000
S3	> 10 and <= 20	>= 15% and < 25%	> 200 and <= 300	> 400 and <= 600

Technical Defects Examples - Cyclic Dependencies

Cyclic method calls among packages

If these method calls are in different layers defined in your architecture, they become architectural violations

Why is this important?

- When you have cyclic dependencies, your code (i.e. classes and packages) are tightly coupled
- Thus code maintenance will be come difficult and expensive
- Responsibility assignment for your classes has not been done correctly
- Will lead to spaghetti code

In Insight, Cyclic Dependencies are always treated as Severity 1 defects

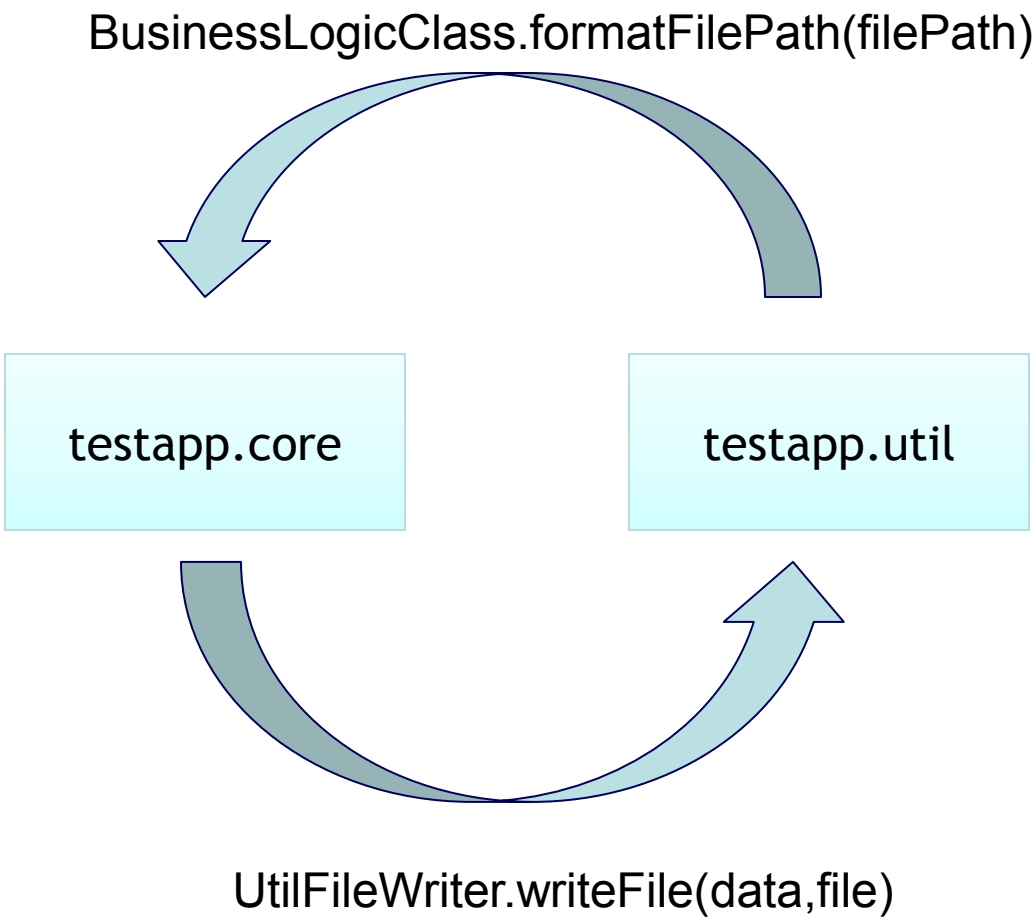
Technical Defects Examples - Cyclic Dependencies (cont…)

```java
package testapp.core;

import testapp.util.UtilFileWriter;

public class BusinessLogicClass {

    // Business function
    public void writeData(String fileName) {
        UtilFileWriter.writeFile("test", "testPath");
    }

    // Utility function
    public String formatFlePath(String filePath) {
        // Format the input string
        return filePath.replace("\\", "/");
    }
}
```

```java
package testapp.util;

import testapp.core.BusinessLogicClass;

public class UtilFileWriter {

    public static void writeFile(String data, String filePath) {
        BusinessLogicClass blc = new BusinessLogicClass();
        String formattedPath = blc.formatFlePath(filePath);

        // Write file
    }
}
```

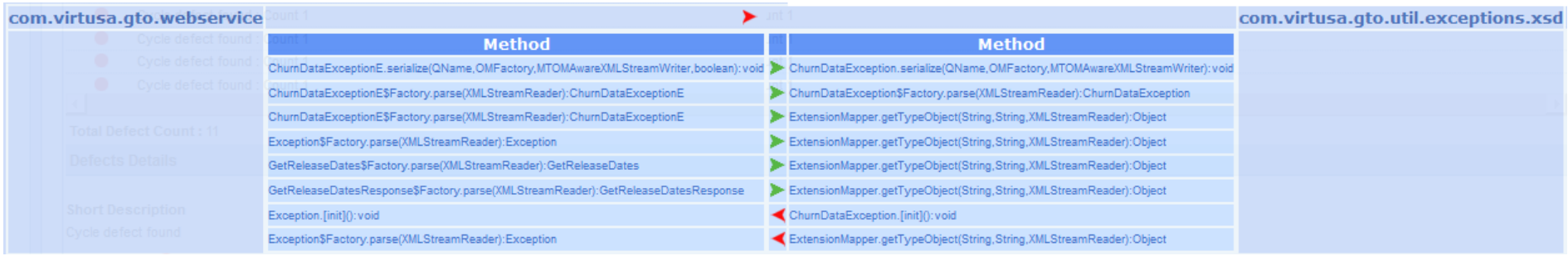
Technical Defects Examples - Cyclic Dependencies (cont…)

Method calls in packages



Technical Defects Examples - Cyclic Dependencies (cont…)

- As seen in Insight Dashboard



com.virtusa.gto.webservice			com.virtusa.gto.util.exceptions.xsd
	Method	Method	
	ChurnDataExceptionE.serialize(QName,OMFactory,MTOMAwareXMLStreamWriter,boolean):void	ChurnDataException.serialize(QName,OMFactory,MTOMAwareXMLStreamWriter):void	
	ChurnDataExceptionE$Factory.parse(XMLStreamReader):ChurnDataExceptionE	ChurnDataException$Factory.parse(XMLStreamReader):ChurnDataException	
	ChurnDataExceptionE$Factory.parse(XMLStreamReader):ChurnDataExceptionE	ExtensionMapper.getTypeObject(String,String,XMLStreamReader):Object	
	Exception$Factory.parse(XMLStreamReader):Exception	ExtensionMapper.getTypeObject(String,String,XMLStreamReader):Object	
	GetReleaseDates$Factory.parse(XMLStreamReader):GetReleaseDates	ExtensionMapper.getTypeObject(String,String,XMLStreamReader):Object	
	GetReleaseDatesResponse$Factory.parse(XMLStreamReader):GetReleaseDatesResponse	ExtensionMapper.getTypeObject(String,String,XMLStreamReader):Object	
	Exception.[init]():void	ChurnDataException.[init]():void	
	Exception$Factory.parse(XMLStreamReader):Exception	ExtensionMapper.getTypeObject(String,String,XMLStreamReader):Object	

The above view displays the method calls that happens between packages

Thus it's easier to identify which methods calls are invalid

Technical Defects Examples - Cyclic Dependencies (cont…)

How to break a cycle?

- Identify the method calls that violate the layered architecture principle. Remove them first
- Use Insight and find out the package link that has the least number of method calls
- Check whether responsibility assignment is done correctly
- Move methods to different packages depending on the relevance.
- Introduce new packages corresponding to your architecture

Fixing Technical Defects

Tools that can be used to find tech defects

- For java - FindBugs or PMD or both
- For.NET – Microsoft FxCop

FindBugs Defect Priorities

- FindBugs uses 3 priority levels P1, P2 and P3.
- These are directly mapped to Insight severity level S1,S2 and S3 respectively.

PMD Defect Priorities

- PMD uses 5 priority levels 1 – 5
- These priorities are mapped to Insight severity levels using the following schema;

PMD Priority	Insight Severity
1 and 2	1
3	2
4 and 5	3

Fixing Technical Defects (cont…)

How to minimize Technical Defects

- Read the suggestions given by the tool. This provides the information about the issue and the resolution
- Integrate the FindBugs /PMD plug-in or FxCop with your IDE. Use it frequently.
- Consult your technical lead
- Use the best practices suggested by the tools when doing coding

What to Fix First ?

- A S1 defect and the S3 defect can have the same sub category
- Thus, rather than picking a category of defects and fixing it, select the high severity defects and fix them

Security Defects

This category can check the following types of issues:

- Hardcoded constant database password
- Empty database password
- HTTP cookie formed from untrusted input
- HTTP Response splitting vulnerability
- Non-constant string passed to execute method on an SQL statement
- A prepared statement is generated from a non-constant String
- JSP reflected cross site scripting vulnerability
- Servlet reflected cross site scripting vulnerability
- And many security types using Ounce and FxCop (for .Net)

Why is this important?

- Identify vulnerable code
- Avoid configuration mistakes such as empty database passwords

Fixing Security Defects

How to minimize Security Defects

- Read the suggestions given by the tool. This provides the information about the issue and the resolution
- Integrate the FindBugs /PMD plug-in or FxCop with your IDE. Use it frequently.
- Consult your technical lead
- Use the best practices suggested by the tools when doing coding

What to Fix First ?

- A S1 defect and the S3 defect can have the same sub category
- Thus, rather than picking a category of defects and fixing it, select the high severity defects and fix them

What does ERA Insight Report?

- Size Metrics ✓
- Code Metrics ✓
- Defects
  - Style Defects
  - Technical Defects ✓
  - Security Defects
- Weighted Defect Density
- Reuse Metrics
- Churn Metrics
- Unit Test Metrics
- Build Stability Metrics

virtusa

Weighted Defect Density

A value derived by applying pre-defined weights for each of the defect category and severity and normalizing by the code size.

- The code size measured in Function Points

Weighted Defect Density is calculated using the following formula;

$$WDD = \left\{ \frac{\left(SD\_S1 + \frac{SD\_S2}{3} + \frac{SD\_S3}{10} \right) * 0.2 + \left(OD\_S1 + \frac{OD\_S2}{3} + \frac{OD\_S3}{10} \right) * 0.8}{Total\_FP} \right\} * 1000$$

Where;

FP – Function Points

OD – Other Defects → Technical Defects and Security Defects

SD – Style Defects

WDD – Weighted Defect Density

Weighted Defect Density

Why we need Weighted Defect Density?

- If we want to compare the code bases on quality, if we just take the defect counts, it would not be totally fair
- For example, a project with a 10 LOC code base having 10 defects is worse than a project with 1000 LOC code base with 10 defects.
- Thus we need a measure that can be used as a measuring stick for code bases across the Insight deck
- As weighted defect density is normalized by the code size (Function Points that is), it can be used for this purpose

Weighted Defect Density

What are the factors that influenced the defect weights?

- Importance of the defects. I.e. the defect severity
- Frequency of the defects reported

Thresholds used

- Following thresholds are used on Weighted Defect Density

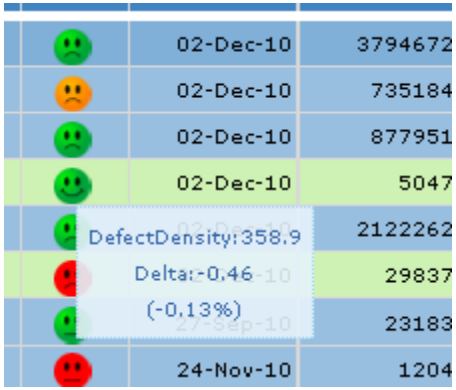
Weighted Defect Density (WDD)	Interpretation	Color Code
WDD <= 200	Good	
200 < WDD <= 800	Medium	
WDD > 800	Bad	

Weighted Defect Density

Defect Density Delta

Defect Density Delta = [Current Defect Density] - [Defect Density at least 7 days back]

- This compares the current defect density with the previous defect density that's at least 7 days old
- This figure is used to check how your code quality is now compared to your code quality last week
- How it's displayed in Insight Dashboard



Weighted Defect Density

Smiley Faces ☺

- ERA Insight uses colored smiley faces to indicate how good / bad your code quality is
- The following is the available smiley faces and their interpretations

Smiley Face	Insight Jargon	Code Quality	Trend
	Green Flat Face	Good	Not enough Data
	Yellow Flat Face	Medium	Not enough Data
	Red Flat Face	Bad	Not enough Data
	Green Happy Face	Good	Quality is improving
	Yellow Happy Face	Medium	Quality is improving
	Red Happy Face	Bad	Quality is improving
	Green Sad Face	Good	Quality is worsening
	Yellow Sad Face	Medium	Quality is worsening
	Red Sad Face	Bad	Quality is worsening

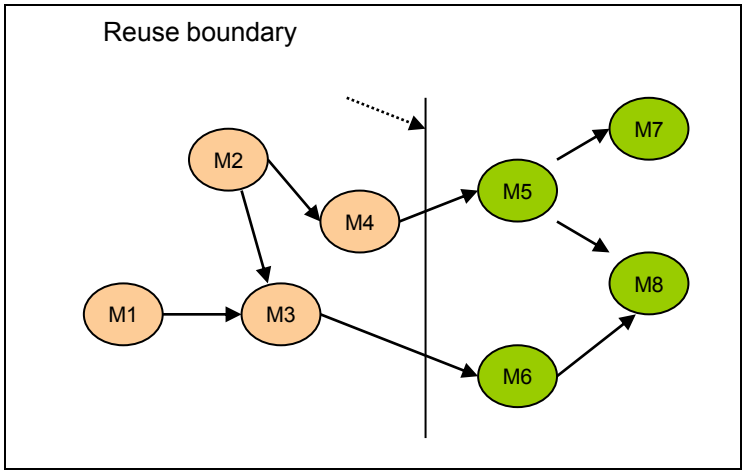
What does ERA Insight Report?

- Size Metrics ✓
- Code Metrics ✓
- Defects
  - Style Defects
  - Technical Defects ✓
  - Security Defects
- Weighted Defect Density ✓
- Reuse Metrics
- Churn Metrics
- Unit Test Metrics
- Build Stability Metrics

virtusa

Reuse Metrics

Reuse Defined



Code units (methods) created within the measurement scope

Reused code units created outside the measurement scope

- Above diagram shows the call graph
- Reuse boundary depends on the scope of reuse measurement (can be a module, project, organization unit, or the company level)
- Created Byte Code = Size of M1+M2+M3+M4
- Reused Byte Code = Size of M5+M6+M7+M8
- Reuse Surface (API code) Size = Size of code units directly called across the reuse boundary = Size of M5+M6

Reuse Metrics

Reuse Defined

$$\text{Reuse Index (Reuse Percentage)} = \frac{\text{Reused Code}}{(\text{Created Code} + \text{Reused Code})} \text{ x } 100\%$$

$$\text{Reuse Quality} = \left(1 - \frac{\text{Reuse Surface}}{\text{Reused Code}}\right) \text{ x } 100\%$$

Reuse Metrics

Why is Reuse Important ?

- Learning curve ("cost" of reuse) is proportional to Reuse Surface while the benefit of reuse is proportional to the Reused Code
- Well designed products minimize the proportion between reuse surface and reused code

What is Technical Reuse?

- The reuse that we discussed is technical reuse
- You consider all the reused entities when you capture the Reused Byte Code

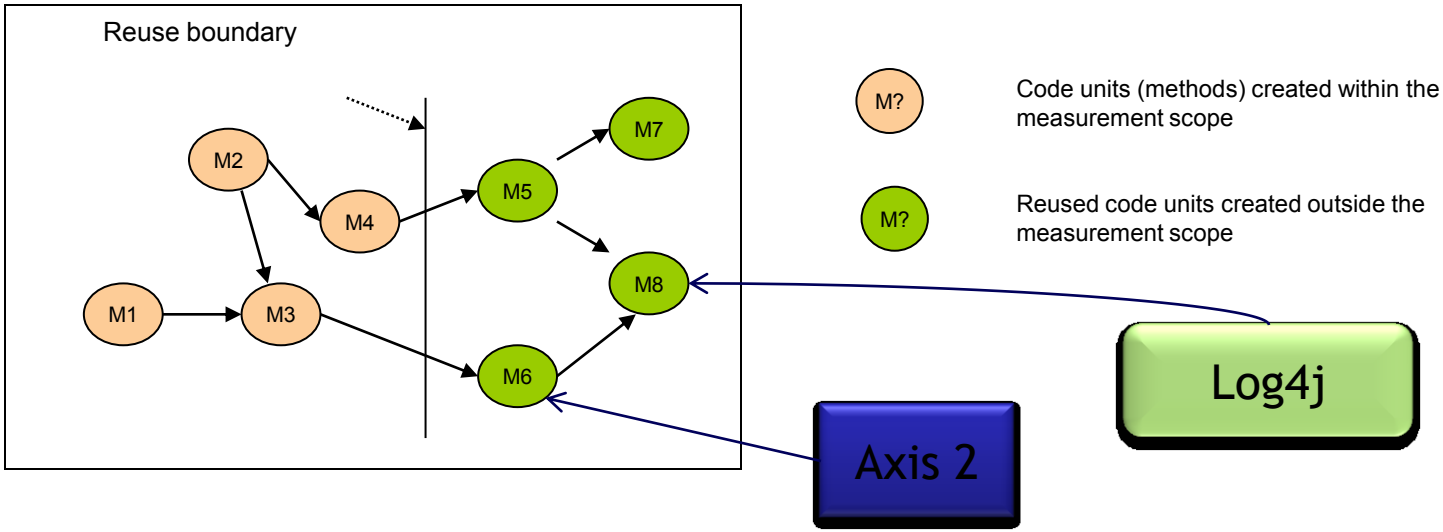
Reuse Metrics

What is Product Reuse

- Insight has the capability of manually mapping package names to products
- The package names comes to the system via tools like GTO-Metrics and Vdepend
  - E.g. org.apache.axis2.* can be mapped to Axis 2
- Thus, some of the reused entities (i.e. packages therefore classes and methods under the packages) can be mapped to reuse

Reuse Metrics

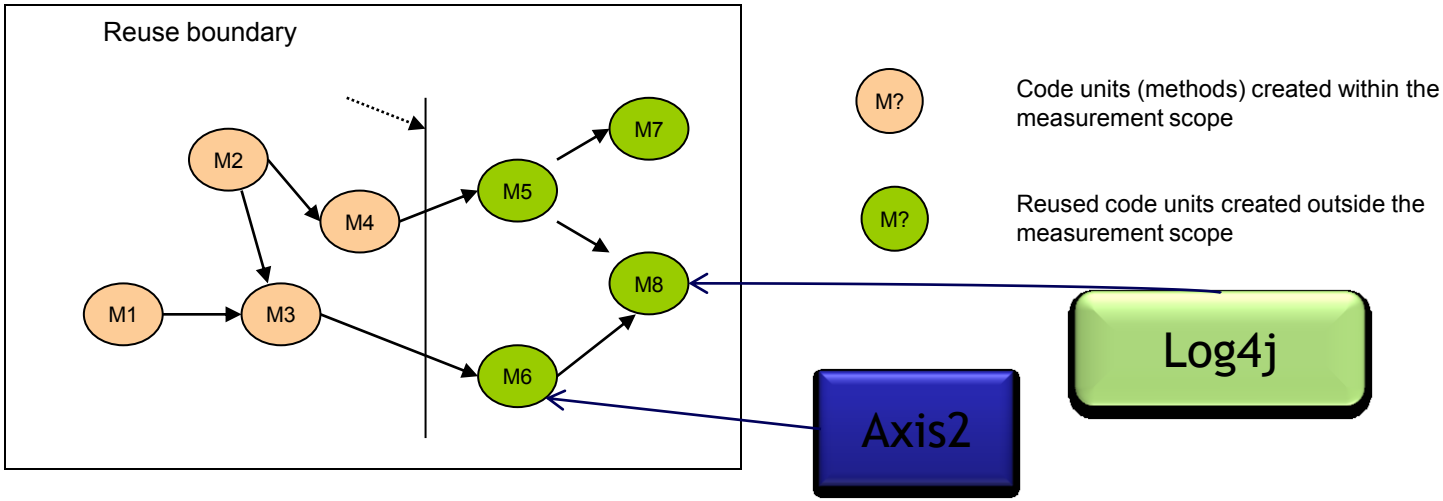
Product Reuse Defined



- M6 and M8 are mapped to products (Log4J and Axis 2)
- Created Byte Code = Size of M1+M2+M3+M4
- Reused Byte Code = Size of M6+M8
- Reuse Surface (API code) Size = Size of code units directly called across the reuse boundary = Size of M6

Reuse Metrics

Product Reuse Defined



- When you calculate the product reuse, the formulas still remain the same
- But how you capture the “Reuse” changes. That is the scope of reuse changes
- Now the scope is only the reuse made on the elements mapped to the products
- Thus, Product Reuse is a sub set of Technical Reuse

Which means, MAX (Product Reuse) = Technical Reuse

What does ERA Insight Report?

- Size Metrics ✓
- Code Metrics ✓
- Defects
  - Style Defects
  - Technical Defects ✓
  - Security Defects
- Weighted Defect Density ✓
- Reuse Metrics ✓
- Churn Metrics
- Unit Test Metrics
- Build Stability Metrics

virtusa

Churn Metrics

What is Churn?

- "Churn is the number of customers who switch from one supplier to another"

  » *Telecom Industry*

- What we are looking at is a behaviorally equivalent occurrence in software.

- ***Code churn is defined as lines added, modified or deleted to a file from one version to another.***

Churn Metrics

Why Code Churn?

- Traditional software development processes are changing
  - Present software development processes are more agile.
  - Code generation and other automation mechanisms are in abundance.
  - Huge knowledge repositories are in existence to help development communities.
- Therefore, traditional LOC based metrics become insufficient
- We need to look at the 'evolution' of the code base in addition to the 'snapshot' view of the code base.
- Enables;
  - Stability assessment
  - Rework identification
  - Defect density predictions
  - Net work vs. total work calculation etc

Churn Metrics

What Insight Captures?

- Total churn trend (for added, changed and deleted content)
  - How the deletions, changes and additions happened to the code base in the version controlling system.
- Total churn summary
  - Consolidates churn information to churn types, content types and languages.
- Churn drilldown
  - Enables drilldown to churn at nesting levels of
    - Developer
    - Content type
    - Language
- Backfired function points for churn
  - Work size is derived from the magnitudes of changes captured.
- Net work size
  - Calculates the effective work that has happened on the code base by comparing the present state of the code base with the project's/module's starting date's state.

What does ERA Insight Report?

- Size Metrics ✓
- Code Metrics ✓
- Defects
  - Style Defects
  - Technical Defects ✓
  - Security Defects
- Weighted Defect Density ✓
- Reuse Metrics ✓
- Churn Metrics ✓
- Unit Test Metrics
- Build Stability Metrics

virtusa

Unit Test Metrics

- ERA Insight does not run Unit Tests or Unit Test Coverage
- If you have incorporated automated unit test execution and unit test coverage calculation, ERA Insight can extract data from the outputs and present it in the dashboard
- Insight can support following unit test tools
  - JUnit for Java
  - NUnit and MSTest for .NET

- Insight can support following unit test coverage tools
  - Emma or Corbertura for Java
  - NCover and MSCover for .NET

What does ERA Insight Report?

- Size Metrics ✓
- Code Metrics ✓
- Defects
  - Style Defects
  - Technical Defects ✓
  - Security Defects
- Weighted Defect Density ✓
- Reuse Metrics ✓
- Churn Metrics ✓
- Unit Test Metrics ✓
- Build Stability Metrics

virtusa

Build Stability Metrics

If you have a continuous integration environment like CruiseControl, Insight can do the following

- Analyze the build logs for a given day
- Find out the total number of builds, the number of successful builds and thus the build success rate
- This is called the Build Stability

Build Stability Metrics

Why is this important ?

- Build stability can indicate how often the build fails in a given build environment
- If the build frequently fails, it can be
  - Developers have not checked in code or references used
  - Developers are not savvy with version controlling or the automated build
  - Conflicts between developed components (i.e. integration failures)
- Thus the risks can be identified early in the life cycle

What does ERA Insight Report?

- Size Metrics ✓
- Code Metrics ✓
- Defects
  - Style Defects
  - Technical Defects ✓
  - Security Defects
- Weighted Defect Density ✓
- Reuse Metrics ✓
- Churn Metrics ✓
- Unit Test Metrics ✓
- Build Stability Metrics ✓

virtusa

Q&A

virtusa

www.virtusa.com

US - Boston, New York **UK** - Windsor, London **India** – Hyderabad, Chennai **Sri Lanka** - Colombo