Extracting Functional Coverage from Visual Studio Code Coverage file.

Author : Srujan Saggam

This idea came to me when I was evaluating Microsoft Visual Studio Team System for one of my projects.

Introduction :

VSTS has special tools integrated with the IDE . which enable us to do the code bullet-proofing activities (unit test cases , static code analysis , code coverage etc.)

Right now we are interested only in code coverage . (we wont be discussing the other code bullet-proofing concepts)

VSTS gives a very brief code coverage analysis report after instrumenting the code(dlls) and monitoring using vsperfcmd.exe. (these are the tools which come with studio)

So when everything is done we have a .coverage file and we have the report in studio ,generated from that .coverage file which can be further exported to .xml format.

What does the report contain?

Namespaces

--Methods Blocks covered/uncovered lines covered/uncovered etc....

Since we are interested in functional coverage , which is unfortunately not shown in studio . We need to find an alternative.

Clearly, we can see the namespaces and methods and blocks/lines covered adn uncovered in those methods . So that means studio has stored the information about methods and their respective blocks covered.

Doesn't that give solution to our problem? we have information somewhere from which we can extract the functional coverage.

When you export the report in .xml carefully analyze the file . You can see how studio manages the report.

Here is the sample of .xml report:

```
&lt;Method&gt;

&lt;MethodKeyName&gt;TestProject1.dll!211c&lt;/MethodKeyName&gt;

&lt;MethodName&gt;InitializeComponentTest()&lt;/MethodName&gt;

&lt;MethodFullName&gt;InitializeComponentTest()&lt;/MethodFullName&gt;

&lt;LinesCovered&gt;3&lt;/LinesCovered&gt;

&lt;LinesPartiallyCovered&gt;1&lt;/LinesPartiallyCovered&gt;

&lt;LinesNotCovered&gt;1&lt;/LinesNotCovered&gt;

&lt;BlocksCovered&gt;3&lt;/BlocksCovered&gt;

&lt;BlocksNotCovered&gt;1&lt;/BlocksNotCovered&gt;

&lt;Lines&gt;
```

.

.

.

.

So each method has got a &lt;method&gt; tag which embeds the method information as an object.

If we can parse all such method tags using somekind of xmlreader object and check with some condition like:

if(blocksCovered&gt;0)

functionsCovered++;

we can extract the functional coverage data.

Here is the Logic that I have used to design a reader for .xml files generated from VSTS:

```
//Author:Srujan
```

```
//Purpose : Extraction of Functional coverage from VSTS report (.xml)

float total = 0; float func = 0;

string filepath = Path.GetFullPath("srujan.xml");//filepath=filepath;

XmlTextReader reader = new XmlTextReader(@filepath); // path to the .xml file from VSTS

while (reader.Read())

{if (reader.Name.Equals("Method")&amp;&amp; reader.NodeType!=XmlNodeType.EndElement)

{func++; // gives the total number of funtions present in the instrumented .dll/.exe

while (reader.Read())

{if (reader.Name.Equals("BlocksCovered") &amp;&amp; reader.NodeType!=XmlNodeType.EndElement)

//enter only if &lt;BlocksCovered&gt; not &lt;/BlocksCovered&gt;

{

reader.Read();
if (Convert.ToInt32(reader.Value) &gt; 0)// if blocks covered is greater than 0

{

total++; // Functions that are covered goes with this count

}

break;

}

}

}

}

Console.Write("Total number of functions:"+func.ToString()+"\n Number of functions
covered:"+total.ToString()+"\n Total number of functions uncovered:"+(func-total).ToString());

Console.WriteLine("\n \nFunctional Coverage:" + (total / func) * 100); // func coverage.
```

Similarly , we can write a .xsl file for the xml which goes with the same above mentioned logic .Simple code but useful , isn't it?People who go for other commercial products like Ncover just because of the incapability of VSTS to process the functional coverage might find this post useful.

Think big. Think tank.
Srujan - The budding blogger &amp; a poet.