FxCop Tool Evaluation

Project Report

Authors:
Ramesh Seela
Ryan Miller
Derek Chang
Ali Shojaeddini
Ankit Sengar

3/25/2008

Contents

1 Introduction

FxCop is a static code analysis tool that checks .NET managed code assemblies for code correctness and conformance to the Microsoft .NET Framework Design Guidelines. FxCop is primarily aimed at supporting the adoption of the .NET Framework design guidelines, establish best practices that minimize code defects and maintenance costs, and transfer expert knowledge regarding technical issues and common programming errors to developers. FxCop helps developers create more consistent APIs (critical in framework and library development,) performant code, and secure applications.

The current version of FxCop (1.36 Beta 2) uses over 200 rules to categorize the defects in the following areas:

1. Design
2. Globalization
3. Interoperability
4. Naming Conventions
5. Mobility
6. Performance
7. Portability
8. Security
9. Usage

FxCop analyzes all constructs in .NET Framework applications including resource files, assemblies, modules, types, properties, events, and exceptions.

2 How FxCop Works

FxCop uses the following features for code analysis:

- **Targets:** These are managed code assemblies, used for analysis.
- **Rules:** These are the checks either provided by FxCop or created by the developers executed on the targets.
- **Messages:** These are the feedback reported as XML based output based on application of Rules.

FxCop uses an analysis engine that deconstructs the assemblies using meta-data APIs [*Source: MSDN*]. The engine calls in the relevant rules for each target or each assembly. It manages the messages that result from analysis and ignores excluded messages.

The tool allows developers to define new rules and integrate them seamlessly to the existing rule set. Moreover, the rule sets can be customized to avoid the use of inapplicable rules and suppress messages generated by the tool. FxCop includes interface to define and add new rules that are specific to the standards and policies set forth by the project.

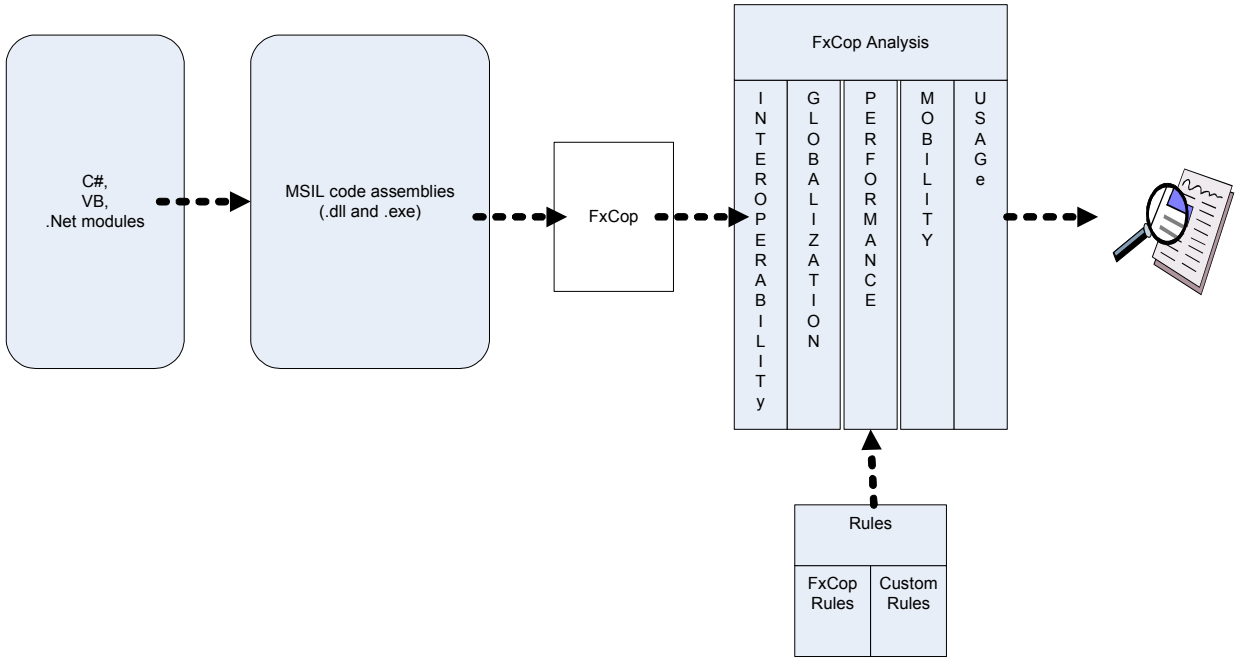


Figure 1: Program analysis using FxCop

2.1 Summary of Analysis Techniques

FxCop uses Reflection exposed type system [*Source: MSDN*] in the assemblies to disassemble Intermediate Language, build call graphs from assemblies, and generate control flow graphs from Methods. FxCop implements a combination of MSIL parsing, static analysis, and call graph analysis techniques to identify and report code defects.

MSIL parsing : FxCop includes an in-built parser to verify Microsoft Intermediate Language code that includes CPU-independent set of instructions for loading, storing, initializing, and calling methods on objects, as well as instructions for arithmetic and logical operations, control flow, direct memory access, exception handling, and other operations.

Static data analysis: FxCop analyses various methods in managed code assemblies by applying both pre-defined and customized rules defined for the project. Using this analysis, the tool identifies code defects that do not adhere to .NET Framework design guidelines. The informative messages help guide the developer understand the anomalies and make necessary corrections to the underlying code.

Call-graph analysis: FxCop internally generates graphs that represent calling relationships among various methods in the managed code assemblies. These call graphs are used to detect anomalies of program execution, violation of recommended guidelines, and possible code injection attacks.

The implementation details of algorithms implemented by FxCop are proprietary and are not published to developer communities.

2.2 Applicable Development Environments

FxCop is designed for analyzing code assemblies of .NET 1.x, .NET 2.0 and .NET 3.x components for conformance to the Microsoft .NET Framework. A project that is built using code that does not include .NET code assemblies does not benefit from this tool.

FxCop includes both GUI and command line versions of the tool and is geared for Windows platform that uses .NET framework version 2.0 or above. The tool support extends to operating systems such as Windows 2000, Windows XP, Windows Vista, Windows 2000 Server, and Windows Server 2003.

FxCop is highly scalable and can be used code bases containing millions of lines of code assemblies.

2.3 Process

FxCop application comes in two flavors: 1) GUI driven analysis tool (FxCop.exe) and 2) Command driven analysis tool (FxCopCmd.exe)

Before performing analysis of code assemblies, the developer needs to provide the information of target assemblies (.exe file or .dll file) and the applicable rules. The following steps can be used with GUI based tool to perform the analysis:

1. Launch FxCop application
2. Click “Project” Menu, Choose “Add Targets” and choose one or more .NET Assemblies
3. Enable Rules in the configuration pane and/or choose “Project Menu”, choose “Add Rules” and pick the file that includes custom FxCop Rule assemblies.
4. Click “Analyze” button on toolbar.

The application displays progress of analysis and the name of analysis engine used to perform the analysis. At the end of analysis, a summary of analysis is displayed that includes statistics and error messages corresponding to the completed analysis. The report also includes the messages, the number of message, start and end time of analysis, and problems encountered while performing the analysis.

2.3.1 Output and Graphical User interface

Following the completion of analysis, the FxCop application window displays the targets and rules included in a project, and the generated messages. The window is divided into three major areas: the configuration pane on the left, the messages pane on the right, and the properties pane at the bottom, as shown in the following screen shot.

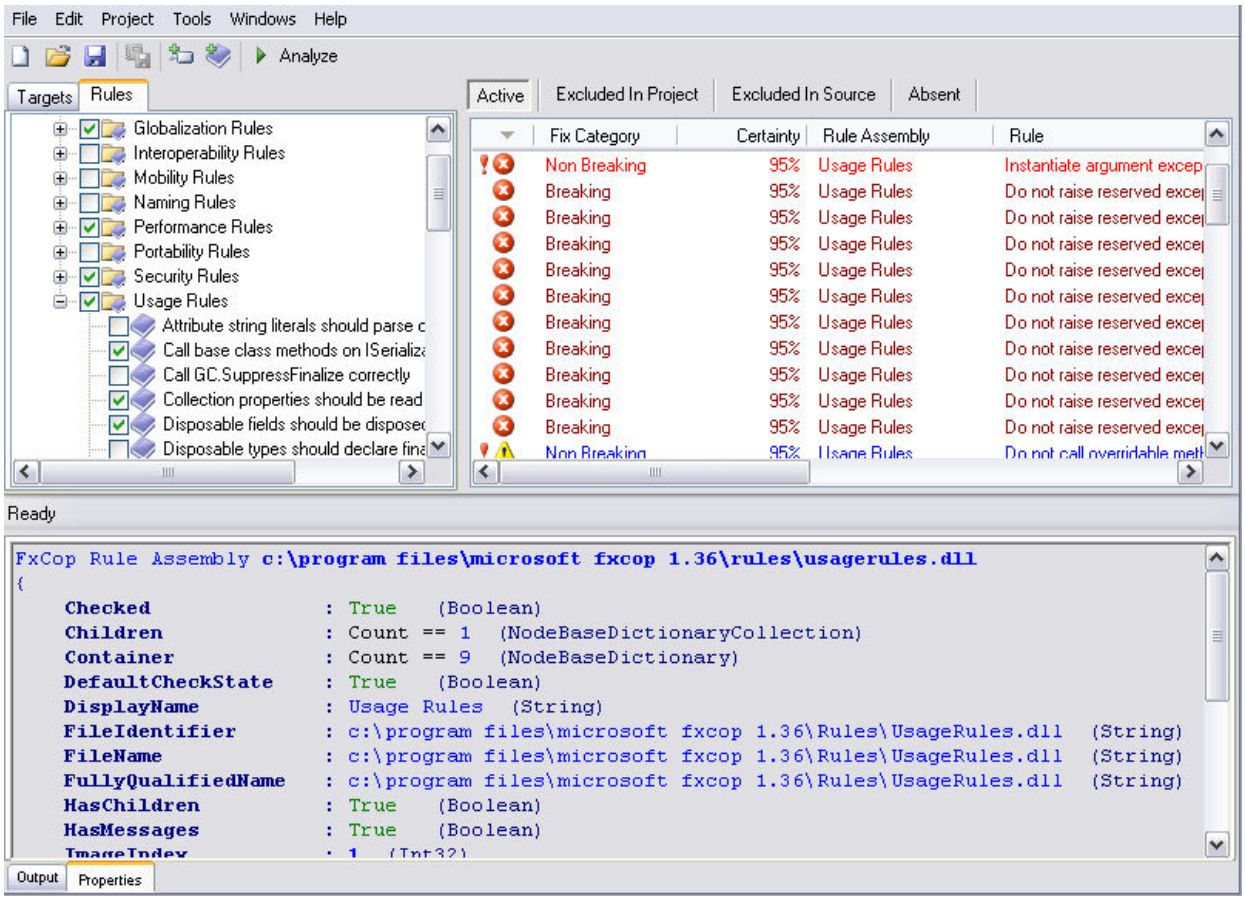


Figure 2 Screenshot of the FxCop 1.36 user interface

FxCopCmd can be used as a stand-alone tool, added to automated build processes, or integrated with Microsoft Visual Studio .NET as an external tool.

3 Quantitative Evaluation

This section explains the quantitative approach used for evaluating the tool and determining its strengths and weaknesses.

3.1 Sample Project Selection

Two projects were selected as sample targets for performing test analysis runs and collecting quantitative data on tool performance. These projects are introduced below.

- **QuickGraph**[1]
  The application provides .NET-based generic graph data structures and algorithms. This package seems ideal for an analysis because of its computation intensive nature.

[1] http://www.codeplex.com/quickgraph

Moreover, many easy to find defects are probably worked out (the product is on release 2.0,) and the tool can be tested for detecting obscure and hard to find bugs.

- **MSDN Reader**[2]
  This application provides offline/caching of MSDN articles and content annotation and sharing. It is based on the new Syndicated Client Experiences Starter Kit Beta SDK released by Microsoft. This project is chosen because it uses some of the .NET Framework technologies such as Windows Presentation Foundation that we intend to use in our Studio project.

3.2 Experiment Process

The following steps were planned and followed for testing tool performance and collecting quantitative data.

1. Initially, the tool is run on selected projects with all the rule categories enabled. This step is used to collect information on all supported rules and collect information to be used as the basis for customization in the following steps.

   In this step, the distribution of reported issues over different Rule Categories and Severity Levels is reported.
2. In this step, the tool is customized in two steps to exclude certain categories of rules as well as specific rules from remaining categories.
3. After rule customization, the analysis is performed again to collect the new warning distribution data.
4. Finally, a subset of warnings from included rules are inspected manually to:
   a. Determine accuracy of reported warnings and classify them as *True Positive*, *True Positive – Don't Care*, and *False Positive*.
   b. Project the number of warnings in each classification mentioned above according to sample data.

3.3 Empirical Findings

3.3.1 Issue Distribution before Customization

The information provided in this section is the outcome of the first step in the process outlined earlier.

Table 1 and Table 2 present the summary of reported issues for the *MSDN Reader and QuickGraph* projects respectively.

[2] http://code.msdn.microsoft.com/msdnreader

Table 1 Summary of reported issues for the *MSDN Reader* project

Severity Level	Design	Globalization	Interoperability	Mobility	Naming	Performance	Portability	Security	Usage
Critical Error	39	0	0	0	8	0	0	7	4
Error	29	20	0	0	49	0	5	0	10
Critical Warning	30	0	0	0	56	8	0	0	8
Warning	84	0	5	0	0	79	0	0	29
Total	**182**	**20**	**5**	**0**	**113**	**87**	**5**	**7**	**51**

Table 2 Summary of reported issues for the *QuickGraph* project

Severity Level	Design	Globalization	Interoperability	Mobility	Naming	Performance	Portability	Security	Usage
Critical Error	5	0	0	0	0	0	0	0	7
Error	42	45	0	0	26	0	0	0	1
Critical Warning	7	0	0	0	132	0	0	0	0
Warning	7	0	0	0	0	17	0	0	2
Total	**61**	**45**	**0**	**0**	**158**	**17**	**0**	**0**	**10**

According to an investigation of rules supported by each category and the empirical information presented above, the following rule categories were excluded according to the first planned customization step:

- **Naming**: adherence to organization and project naming conventions may override those of the .NET Framework Design Guidelines
- **Globalization**: applicability of these rules depends on project needs and may not apply to all products
- **Interoperability**: interacting with COM clients may not be required or desirable in many projects
- **Mobility**: these rules support efficient power usage which is applicable only to certain projects

After performing intermediate test runs (without the excluded categories,) the following criteria were used to exclude specific rules according to individual rule importance, severity level, and accuracy level[3]. The following list provides brief description of the strategy used for two major categories:

- For design issues, we skip rules that result only in warnings with low certainty percentage and low severity level. Examples of such rules include "*Avoid Namespaces with Few Types*", "*Use Properties Where Appropriate*", or "*Indexers Should Not Be Multidimensional*."

[3] The severity and accuracy levels are provided by the rule as part of the reported issues.

- For performance, security, and usage warnings, we inspect all rules to ensure no issues are overlooked. For example, we include the "*Review Visible Event Handlers*" rule, even though it results in low-certainty warnings.

3.3.2 Issue Distribution after Customization

The information presented in this section reflects the results of executing step 3 in the process outlined above.

Figure 3 and Figure 4 demonstrate the distribution of reported issues for the two sample projects.

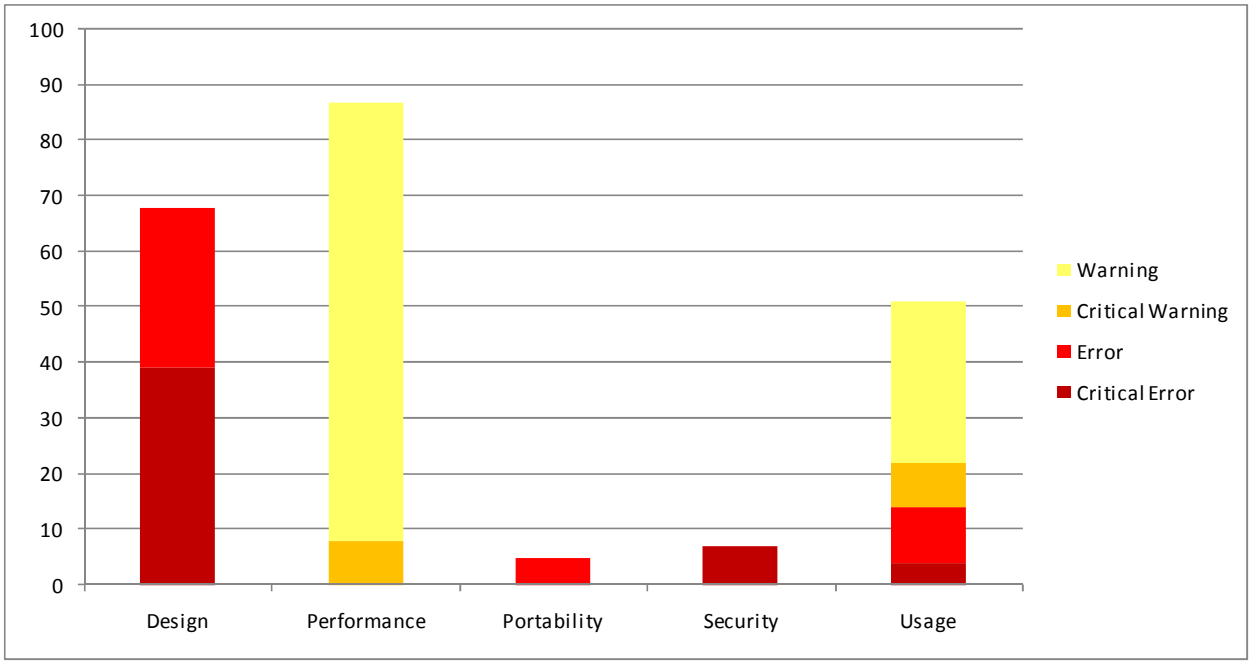


Figure 3 Distribution of reported issues for *MSDN Reader* after customization

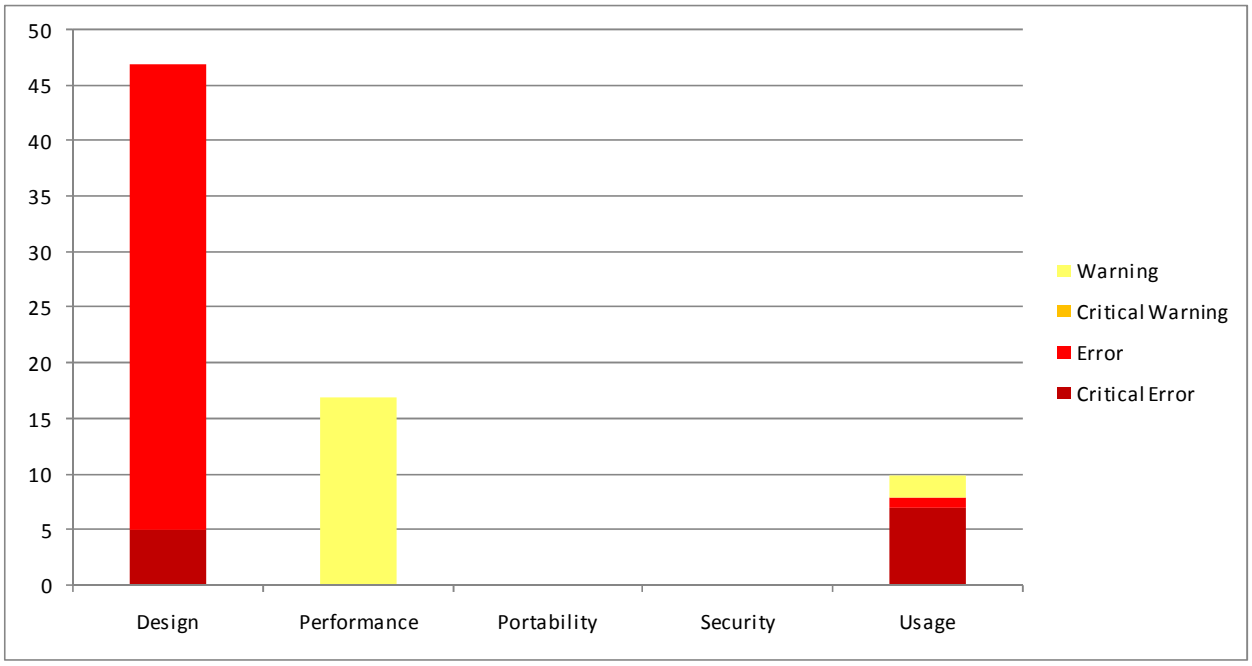


Figure 4 Distribution of reported issues for QuickGraph after customization

As demonstrated in these figures, the majority of reported issues fall in *Design*, *Usage*, and *Performance* categories. This is consistent with the original purpose of the tool, which is ensuring that the application or library complies with .NET Framework *Design* Guidelines. Figure 5 the percentage of issues reported in each category for both sample projects.

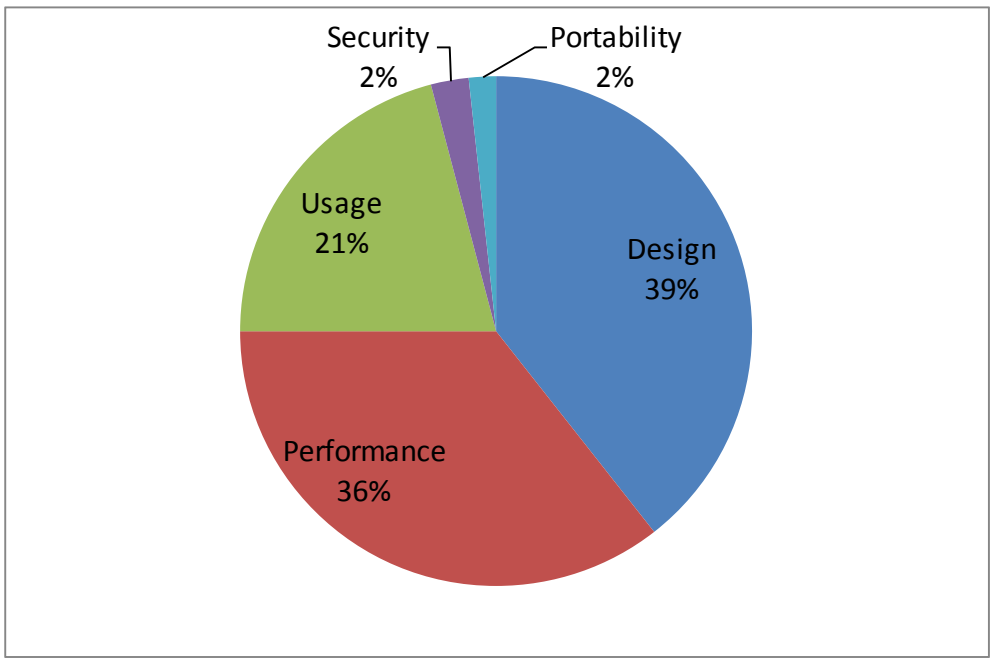


Figure 5 Combined distribution of reported issues over categories after customization

Of course, more sample projects from different application domains are needed before concluding on performance of the tool in each category. However, based on stated purpose of the tool, the number of important rules in the categories, and the empirical findings presented here, the following rule categories are the strongest within the tool: *Design*, *Usage*, and *Performance*.

3.4 Issue Classification as True Positive and False Positive

We believe that the certainty provided by the rule as part of the issue report provides a good measure of confidence in the accuracy of the report. Furthermore, inspecting every single issue in the applications that we are not familiar with is an infeasible and error-prone process. Therefore, we employed a sampling mechanism for efficiency—if the certainty was above 60% for the same rule, we inspected half of the reported issues randomly; if it was below 60%, we inspected all of them.

As a result of the process described above, we achieved the following statistical information for the rate of reported issues within each classification (relevant true positives, irrelevant true positives, and false positives.)

Table 3 True Positive and False Positive classification of issues for MSDN Reader

Severity Level	Design	Performance	Portability	Security	Usage	All Categories
True Positive – Relevant	56	65	0	0	16	137
True Positive - Irrelevant	12	18	5	7	35	77
False Positive	4	0	0	0	0	4
Total	**72**	**83**	**5**	**7**	**51**	**218**

Table 4 True Positive and False Positive classification of issues for QuickGraph

Severity Level	Design	Performance	Portability	Security	Usage	All Categories
True Positive – Relevant	37	10	0	0	2	**49**
True Positive - Irrelevant	10	7	0	0	8	**25**
False Positive	0	0	0	0	0	
Total	**47**	**17**	**0**	**0**	**10**	**74**

3.4.1 Sample True Positives – Relevant

MSDN Reader

Method 'MsdnStoryImageHyperlink.MsdnStoryImageHyperlink(Uri, Story)' passes parameter name 'navigateUri' as the 'message' argument to a 'ArgumentException' constructor. Replace this argument with a descriptive message and pass the parameter name in the correct position.

Warning Message

```
CriticalError, Certainty 95, for InstantiateArgumentExceptionsCorrectly
```

```
{
    Resolution    : "Method 'MsdnStoryImageHyperlink.MsdnStoryImageHyperlink(
                    Uri, Story)' passes parameter name 'navigateUri' as
                    the 'message' argument to a 'ArgumentException'
                   constructor.
                    Replace this argument with a descriptive message and
                    pass the parameter name in the correct position."
    Category      : Microsoft.Usage  (String)
    CheckId       : CA2208  (String)
}
```

The Code Fragment

```
public MsdnStoryImageHyperlink(Uri navigateUri, Story story) : base()
{
     if (navigateUri == null)
     {
         throw new ArgumentException("navigateUri");
     }

     if (story == null)
     {
         throw new ArgumentNullException("story");
     }

     NavigateUri = navigateUri;
     _story = story;
     _imageReference = null;
     _isImageReferenceLink = false;
}
```

Quickgraph

Do not declare static members on generic types(This is a critical issue and we cannot compile our program with generic types defined as static)

Warning Message

```
Error, Certainty 95, for DoNotDeclareStaticMembersOnGenericTypes
 {
     Resolution    : "Remove 'Edge<TVertex>.VertexType' from 'Edge<TVertex>'
                     or make it an instance member."
     Category      : Microsoft.Design  (String)
     CheckId       : CA1000  (String)
 }
```

The Code Fragment

```
using System;
 namespace QuickGraph
 {
     [Serializable]
```

```
      public class Edge<TVertex> : IEdge<TVertex>
      {
          private readonly TVertex source;
          private readonly TVertex target;

          public Edge(TVertex source, TVertex target)
          {
              GraphContracts.AssumeNotNull(source, "source");
              GraphContracts.AssumeNotNull(target, "target");
              this.source = source;
              this.target = target;
          }

          public static Type VertexType
          {
              get { return typeof(TVertex); }
          }

          public TVertex Source
          {
              get { return this.source; }
          }

          public TVertex Target
          {
              get { return this.target; }
          }

          public override string ToString()
          {
              return String.Format("{0}->{1}", this.Source, this.Target);
          }
      }
  }

}
```

3.4.2 Sample True Positives – Irrelevant

MSDN Reader

We don't really care about the design issues because basically it is an RSS reader that the user will not requires or has any mechanism to write collection data back to the back-end server.

Warning Message

```
Warning, Certainty 75, for CollectionPropertiesShouldBeReadOnly
{
    Resolution    : "Change 'MainStoryControl.Stories' to be read-only
                    by removing the property setter."
    Category      : Microsoft.Usage  (String)
    CheckId       : CA2227  (String)
```

```
}
```

The Code Fragment

```
/// <summary>
/// The StoryCollection this control is currently binding to.
/// </summary>
public StoryCollection Stories
{
      get { return (StoryCollection)GetValue(StoriesProperty); }
      set { SetValue(StoriesProperty, value); }
}
```

Quickgraph

Remove unused locals(We can ignore errors related to performance because they do not break our application.)

Warning Message

```
 Warning, Certainty 95, for RemoveUnusedLocals
 {
     Resolution   : "'ImplicitEdgeDepthFirstSearchAlgorithm<TVertex,
                    TEdge>.Visit(TEdge, int)' declares a variable, 'c',
                    of type 'GraphColor', which is never used or is only
                    assigned to. Use this variable or remove it."
     Category     : Microsoft.Performance  (String)
     CheckId      : CA1804  (String)
 }
```

The Code Fragment

```
if (!this.EdgeColors.ContainsKey(e))
                {
                    OnDiscoverTreeEdge(se, e);
                    Visit(e, depth + 1);
                }
                else
                {
                    GraphColor c = this.EdgeColors[e];
                    if (EdgeColors[e] == GraphColor.Gray)
                        OnBackEdge(e);
                    else
                        OnForwardOrCrossEdge(e);
                }
```

3.4.3 Sample False Positives

MSDN Reader

The code actually implemented with the base type. We believe that they might be some compiling/building transformation causes the false positives. And actually four identified issues are also false positives.

Warning Message

```
Error, Certainty 50, for ConsiderPassingBaseTypesAsParameters
{
    Resolution   : "Consider changing the type of parameter 'e' in
                   'MsdnViewManager.OnImageHyperlinkRequestNavigate(object,
                   RequestNavigateEventArgs)' from 'RequestNavigateEventArgs'
                   to its base type 'RoutedEventArgs'. This method appears
                   to only require base class members in its implementation.
                    Suppress this violation if there is a compelling reason
                   to require the more derived type in the method signature."
    Category     : Microsoft.Design  (String)
    CheckId      : CA1011  (String)
}
```

The Code Fragment

```
 /// <summary>
 /// Static handler for image hyperlink's request navigate event
 /// </summary>
 /// <param name="sender"></param>
 /// <param name="e"></param>
 public void OnImageHyperlinkRequestNavigate(object sender, RoutedEventArgs
e)
 {
       // actuall code here….
       //……
       //……
       // actuall code here….

 }
```

4 Qualitative Evaluation

4.1 Report Content

The warnings reported by the tool contain comprehensive information that explain cause of the warning and guide the developers in fixing the issues. The following table lists some of the important items included in each warning.

Item	Description
Message Level	The importance of the issue that is identified by the rule (the four levels are Critical Error, Error, Critical Warning, and Warning)
Certainty	The estimate of the probability that the issue is detected correctly (an integer between 1 and 99)
Breaking Change	Whether the fix for a violation of the rule constitutes a breaking change

	Breaking change means that an assembly that has a dependency on the target that caused the violation will not re-compile with the new fixed version or might fail at runtime because of the change
How to Fix Violations	Explains how to change the source code to satisfy the rule and prevent it from generating a warning
When to Exclude Warnings	Describes when it is safe to exclude a warning from the rule

4.2 Usability

The user interface of the tool is designed in a way that can guide users in operating the tool without referring to any manual. Moreover, the main functionalities of the tool are accessible through main application view, and don't require exploring options and settings dialogs. Especially, rule customization and navigation can be easily done through the left *Rules* pane.

However, inspecting reported issues is not done as easiest possible way since the tool relies on external editors to direct the users to the origin of the reported issue. This is because FxCop handles .NET assemblies rather than source files. Therefore, tracing the issues back to source will require more effort from the users.

4.3 Customization and Rule Extension

As demonstrated in Figure 6, the tool allows entire categories or specific rules within categories to be excluded in the analysis. Rule customization is stored in the FxCop analysis project file. Therefore, it is possible to use various customizations for different projects according to their needs. The tool also allows specific rules to be excluded through inspection of issues reported by that rule.

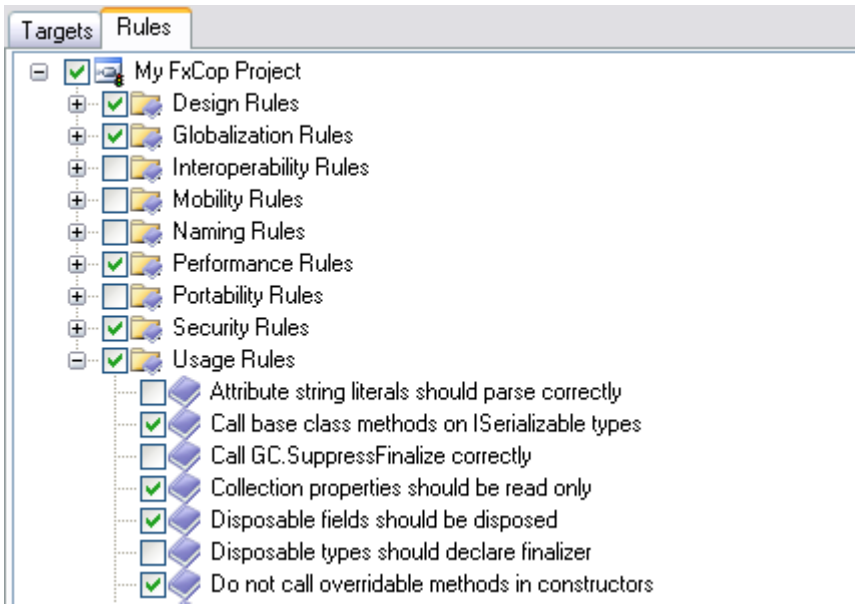


Figure 6 Rule customization in FxCop analysis projects

In addition, FxCop allows users to extend the tool by providing custom rule sets. All rule sets are stored in .NET assemblies, and the rule assemblies are loaded at run-time to access and execute the rules. Therefore, adding a new rule set involves developing a new .NET assembly containing the new rules and configuring the tool to load the assembly in addition to default rule assemblies.

The FxCop SDK contains the two following assemblies that must be referenced by developed rule sets:

- FxCopSdk.dll
- Microsoft.Cci.dll

After referencing these assemblies, the API will be accessible through the "**Microsoft.FxCop.Sdk**" namespace.

4.4 Documentation and Community Support

Microsoft developer network contains sufficient documentation and guidelines for using the tool. Moreover, the built-in rules are well-documented and accompanied by recommended ways to fix the issues and example violations of the rule in different .NET languages.

However, limited documentation is available on the internal methods and analysis techniques used by the tool. Similarly, limited documentation was available regarding development of custom rules by using the FxCop SDK, and we relied on third-party articles to gather information on how the tool can be extended. As a result, there is no large repository of community developed rules that can be downloaded and plugged in the tool.

5 Visual Studio Team System Integration

Starting with Visual Studio 2005, the FxCop analysis engine is integrated in the Team System edition of the Visual Studio product family. The following sections briefly explain the static analysis features available in Visual Studio Team System 2008.

Integration of the static analysis engine in Visual Studio enhances the usability and lifecycle integration of the tool. Static analysis can be configured directly in Visual Studio solution and project properties and enforced at check-in time when accessing code through supported source control products. Moreover, the errors and warnings are reported in the standard build output windows and code inspection does not require launching an external source editor.

5.1 Rule Categories

Visual Studio Team System 2008 supports all categories and rules available in FxCop 1.36 engine. However, two additional rule categories are introduced in Visual Studio. These rule categories are briefly explained below.

5.1.1 Maintainability Rules

These rules mainly rely on the new Code Metrics feature of Visual Studio 2008 to detect unmaintainable code. The Code Metrics features of Visual Studio are briefly introduced later in this section. The following is the list of rules under this category:

- Avoid excessive class coupling
- Avoid excessive complexity
- Avoid excessive inheritance
- Avoid unmaintainable code
- Review misleading field names
- Variable names should not match field names

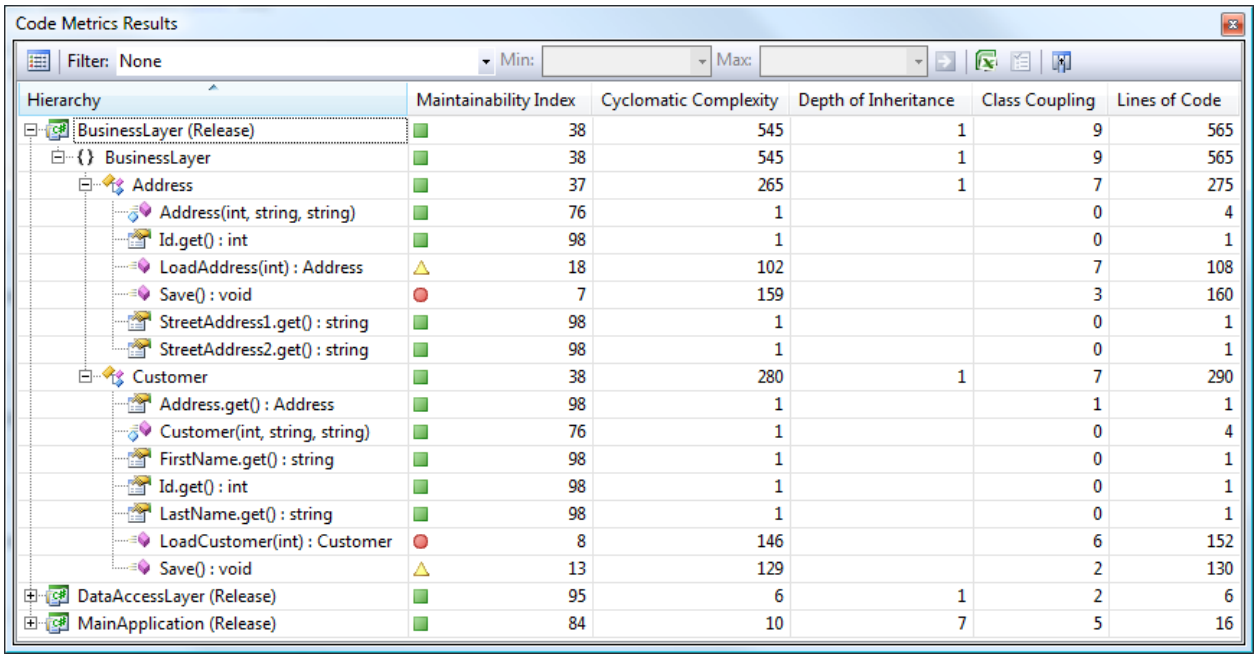
5.1.2 Reliability Rules

These rules support reliability of the library or application by ensuring correct memory management and thread usage. The rules in this category include:

- Avoid calling problematic methods
- Do not lock on objects with weak identity
- Do not treat fibers as threads
- Remove calls to *GC.KeepAlive*
- Use *SafeHandle* to encapsulate native resources

5.2 Code Metrics

As mentioned above, Code Metrics is a new feature introduced in Visual Studio Team System 2008. This feature is accessible through the “Code Metrics Results” window (Figure 7) and helps users detect complex and unmaintainable areas in the code. This feature is also the basis for the new Maintainability category of static analysis rules.


Code Metrics Results

Filter: None Min: Max:

Hierarchy	Maintainability Index	Cyclomatic Complexity	Depth of Inheritance	Class Coupling	Lines of Code
BusinessLayer (Release)	38	545	1	9	565
BusinessLayer	38	545	1	9	565
Address	37	265	1	7	275
Address(int, string, string)	76	1		0	4
Id.get() : int	98	1		0	1
LoadAddress(int) : Address	18	102		7	108
Save() : void	7	159		3	160
StreetAddress1.get() : string	98	1		0	1
StreetAddress2.get() : string	98	1		0	1
Customer	38	280	1	7	290
Address.get() : Address	98	1		1	1
Customer(int, string, string)	76	1		0	4
FirstName.get() : string	98	1		0	1
Id.get() : int	98	1		0	1
LastName.get() : string	98	1		0	1
LoadCustomer(int) : Customer	8	146		6	152
Save() : void	13	129		2	130
DataAccessLayer (Release)	95	6	1	2	6
MainApplication (Release)	84	10	7	5	16

Figure 7 Code Metrics Results tool window in Visual Studio Team System 2008

The five metrics calculated by the Code Metrics feature are listed below:

- **Class Coupling**
  Indicates the total number of dependencies that the item has on other types
- **Depth of Inheritance**
  Indicates the number of types that are above the type in the inheritance tree
- **Cyclomatic Complexity**
  Indicates the total number of individual paths through the code
- **Lines of Code**
  Indicates the total number of executable lines of code, which excludes white space, comments, braces and the declarations of members, types and namespaces
- **Maintainability Index**
  Indicates the overall maintainability index (0 to 100) of a member or type based on several metrics, including *Halstead Volume*, Cyclomatic Complexity and Lines of Code

6 Conclusion

Based on analysis, we conclude that FxCop is a simple, easy-to-use tool with rich features to catch possible code violations early in the development and build secure and performant code. The tool Moreover, the tool is available free and is enhanced on a regular basis by the leading company in software. The only drawback, limited documentation available for the tool, is alleviated through the use of online forums by the DotNet community.

6.1 Strengths

- Well suited for enforcing design and usage guidelines in .Net applications
- Self intuitive user interface offers adequate guidance to developers
- False positives reported by the tool are very minimal (based on our analysis)
- Messages and warnings reported during analysis are very informative
- Extensible and customizable rule sets
- Integrates seamlessly with Microsoft Visual Studio

6.2 Weaknesses

- Limited documentation on tool internals
- Insufficient guidance on creating customized rules
- Plugins for custom rules not available on MSDN communities

6.3 Recommendations

We recommend FxCop to teams developing software using .Net framework. As the tool integrates seamlessly with VS.net, the teams using Visual Studio IDE are advised to include the tool in development lifecycle and automate program analysis. In particular, the tool will prove useful to aid enforcement of best design practices in the teams that include less experienced developers.

6.4 Applicability to MSE Studio

The software development for our studio project relies on .Net framework to build the platform to support authoring and viewing cases. For a team constrained by human resource, FxCop comes very handy in ensuring best programming practices and design guidelines. The ability to integrate the tool with Visual Studio 2008 allows the team to catch the errors during code check-in and fix the code. Performance, Portability, Security, and Robustness are a few quality attributes that are critical to the success of the project. The team can use the rules offered by the tool along these categories and benefit from customizing the rule sets to the requirements of the project.

7 References

- The Visual Studio Code Analysis Team Blog (http://blogs.msdn.com/fxcop/)
- FxCop overview (http://msdn2.microsoft.com/en-us/library/bb429476%28VS.80%29.aspx)
- Code Correctness with FxCop (http://msdn.microsoft.com/msdntv/transcripts/20031204FxCopMMTranscript.aspx)