https://doi.org/10.48009/1_iis_2014_1-10

Issues in Information Systems

Volume 15, Issue I, pp. 1-10, 2014

USING CODE ANALYSIS TOOL IN INTRODUCTORY PROGRAMMING CLASS

Abhijit Sen, Kwantlen Polytechnic University, abhijit.sen@kpu.ca

ABSTRACT

Code Review is one of the key components of the Software development process. It can be effectively used in a programming class to teach both novice and advanced students how to improve the program structure, and write cleaner code. Code Analysis will have significant positive impact on the quality of the code the students will produce. Code Review or Analysis can be done manually, however automated tools can be easily used to teach students better design principles and help improve productivity. In this paper the author attempts to explore methods to improve the teaching of computer programming by integrating code analysis techniques while teaching programming to the students with no, or minimal programming experience. The author suggests that the code analysis tool should be an integral part of teaching programming courses and discusses the author's experiences in teaching C# programming course using the Code Analysis feature of Microsoft Visual Studio.

Keywords: code analysis, code review, programming, and software development

INTRODUCTION

Teaching computer programming to students with very little background in computing is not an easy task. First year students have difficulties in learning the syntax and semantics of a particular programming language such as C# or Java. In addition students also have to learn appropriate design strategies, program decomposition and segmentation mechanisms, and to use professionally acceptable programming practices. In real world environment, program development team conducts code review and analysis to ensure that developers follow best practices and develop consistent code. However, in introductory programming courses code that is written for classes generally does not go through similar code review processes, because most problems that students tackle in these introductory courses are simple, and emphasis is placed on writing short, syntactically and structurally correct programs. In an academic setting it is also difficult to use existing commercial code review processes. Moreover, instructors don't have sufficient lecture time to teach methodologies to debug and write cleaner and easier-to-debug code. In this paper the author suggests how one can appreciably advance student learning of C# programming language and improve the quality of developed software by applying code analysis tools, such as Microsoft Visual Studio, in teaching C# language. The strategy of including code analysis concepts in teaching and developing applications will be of interest to people teaching both introductory or advanced level programming and application development courses.

LTERATURE SURVEY

The best way of teaching programming to students has been extensively discussed in literatures [1, 2]. The issues and challenges in teaching program concepts and structures have been articulated in [2]. The teaching approaches and learning tools to be included in the curriculum for motivating students have been proposed in [3]. A system for distributed code review in a classroom context is proposed in [4]. Students learning of a programming language could be significantly improved if one uses widely available automated tools to supplement traditional teaching methods. There are number of Code Analysis tools available for .NET framework such as dotTEST [5], CodeIt.Right [6], CodeRush [7] to name a few. These tools, although suitable for large scale software development, are not very easy to use and integrate in a class room teaching environment. However, all current versions of Visual Studio.NET IDE include Code Analysis tool. Code Analysis tool can help students to inspect the code quickly and easily, find common mistakes, and violations of standards. The code analysis is an important component of programming, and as such must be integrated into the course syllabus to enhance students' learning of a language.

METHODOGY OF CODE ANALYSIS

The Code Analysis is introduced in Program Structure & Design course, which is an introductory course for Computer Science and Information Technology students. The students learn C# programming language in this course using Microsoft Visual Studio.NET IDE. In this paper the author demonstrates with few simple experiments the result of running code analysis on simple code snippets. The author highlights the errors detected by the code analysis and shows how to fix these errors so that resulting program is validated by running through code analysis tool.

Code Analysis tool ensures that source code adheres to predefined design and style guidelines as well as best coding practices, and reports any programming and design rule violations. These rules are based on the Microsoft .NET Framework Design Guidelines [8]. In Visual Studio 2010 one can manage the list of rules that are executed against code using rule sets. Visual Studio comes with eight pre-defined rule sets. Each rule set defines variety of rules. Basic Correctness Rules for example defines set of 29 rules [9].

- Basic Correctness Rules - focuses on logic errors and common mistakes in the usage of .NET framework
- Basic Design Guidelines Rules - focus on making code easier to understand and use
- Extended Correctness Rules - focus on logic and framework usage errors for specific application such as mobile application
- Extended Design Guidelines Rules - extends on the basic design guideline rules to maximize the usability and maintainability issues.
- Globalization Rules - focus on problems that might prevent data in your application from appearing correctly in different languages, locales, and cultures
- Minimum Recommended Rules - focus on the most critical problems in code, including potential security holes, application crashes, and other important logic and design errors.
- Security Rules – focus on addressing potential security issues in code
- All Rules- contains all rules for analyzing managed code

There are over 200 rules that grouped into different categories. These categories are then broken into different rule sets, targeting specific coding issues. The Table 1 summarizes some of the important rules from 'Code Analysis for Managed Code Warnings' [10].

Table 1: Summary of Pre-defined Rule Sets

Categories of Rules	**Purpose**	**Examples**
Naming Conventions	Enforce naming standards as described in the Design Guidelines	CA1719: Parameter names should not match member names [10]
Performance rules	Detect wasteful, redundant, or extraneous code that could be optimized for performance.	CA1820: Test for empty strings using string length [10]
Security rules	Identify insufficient or incorrect security practices	CA2121: Static constructors should be private [10]
Design Rules	Enforce proper implementation of interfaces and structure of code	CA1062: Validate arguments of public methods [10]
Globalization rules	Support the internationalization of code and focuses on formatting	CA1303: Do not pass literals as localized parameters [10]
Maintainability rules	Help to make code easier to maintain	CA1500: Variable names should not match field names [10]
Usage Rules	Support appropriate usage of the .NET Framework.	CA2241: Provide correct arguments to formatting methods [10]

The Error List window displays results of running code analysis on application, and lists details of violations - such as code file name, project name, line number etc. The violations of rules are displayed as warnings or compilation errors. One can either fix the violation or suppress the violation if that is not applicable for the project. Double clicking on the error list will redirect the user to the specific line where violation has occurred.

To access the static code analysis settings for a project in Visual Studio 2010,

- Right-click on the project in Solution Explorer
- Select Properties from the context menu.
- Select the Code Analysis tab to begin configuring how we want code analysis to work with this project. Figure 1 shows the layout for the Code Analysis tab.
- Enable Code Analysis on Build checkbox
- Using the drop down list box, you can select the rule set that you want to apply

Using the above steps we will demonstrate Code Analysis with the following rule set:

- All Rules
- Minimum Recommended Rules
- Custom Rules

For all cases below code analysis is performed using source code in Appendix II. For the purpose of demonstration only detected violations of custom rules are addressed in the subsequent section.

CODE ANALYSIS USING MICROSOFT ALL RULES

In this section we will show the results of executing Code Analysis using All Rules. As shown in Figure the Code Analysis has detected 25 potential issues of code violations as shown in Figures 1, 2 Appendix 1.

CODE ANALYSIS USING MICROSOFT MINIMUM RECOMMENDED RULES

In this section we will show the results of executing Code Analysis using Minimum Recommended Rules. As shown in Figure the Code Analysis has detected 2 potential issues of code violations as shown in Figure 3, Appendix 1.

CODE ANALYSIS BY CREATING OWN RULES

In this section we will show the results of executing Code Analysis using Custom Rules on source code in Appendix II. The rule set editor allows one to create own custom rule sets. The rule set configuration dialog and rule set editor can be accessed from the Project Properties \ Code Analysis tab. The following eight custom rules are created – CA1014, CA1062, CA1709, CA 1801, CA1802, CA1820, CA2210, and CA2241. Figure 6, Appendix 1 shows the selection of Custom Set Rules. For more information on using rule sets, refer to documentation 'Walkthrough: Configuring and Using a Custom Rule' [11].

When code analysis is done on the original application 16 violations are detected as shown in Figure 7, Appendix I. Each warning message details how the code has violated the specified rules. The table below shows the steps taken to fix the violations of the rules. All violations, except Warnings 3, 4, 5 are related to the program code of Test.cs and Program.cs. Warnings 3, 4 and 5 do not relate to specific codes, but indicate possible issues related to security or assembly. Table 2 shows the steps necessary to remove all the violations detected by Code Analysis.

The above fixes are implemented and shown in the source files Test.cs and Program.cs in Appendix III. When Code analysis is executed there are no Warnings issued for the violations that are corrected (See Figure 7). This indicates that the code has passed all the specified Code Analysis rules.

Table 2: List of Warnings and Corrections to remove violations

Warning	Warning Message	Corrections to remove violations
1	The field 'iacis.Program.msg' is assigned but its value is never used	Added lines 14,16 Program.cs
2	CA1709: Correct the casing of 'iacis' in namespace name 'iacis' by changing it to 'Iacis'.	Name changed to Iacis_NoViolation –line 6 Program.cs
3	CA2210: Sign 'iacis.exe' with a strong name key.	Sign the assembly using instructions on http://msdn.microsoft.com/en-us/library/ms182127(v=vs.100).aspx
4	CA1709: Correct the casing of 'iacis' in assembly name 'iacis.exe' by changing it to 'Iacis'.	Change the title line in AssemblyInfo.cs by: `[assembly: AssemblyTitle("Iacis_NoViolation")]`
5	CA1014: Mark 'iacis.exe' with CLSCompliant(true) because it exposes externally visible types.	Add the line: `[assembly:CLSCompliant(true)]` in `AssemblyInfo.cs` file
6	CA1801: Parameter 'args' of 'Program.Main(string[])' is never used. Remove the parameter or use it in the method body.	Parameter of Main() removed –line 12 Program.cs
7	CA1802: Field 'Program.msg' is declared as 'static readonly' but is initialized with a constant value 'Hello'. Mark this field as 'const' instead.	The field is marked as constant - line 10 Program.cs
8	CA1709: Correct the casing of 'display' in member name'Test.displayMessage1 (string)' by changing it to 'Display'.	display is changed to Display - line 11 Test.cs
9	CA1820: Replace the call to 'string.operator ==(string, string)' in 'Test.displayMessage1(string)' with a call to 'String.IsNullOrEmpty'.	Checked if msg is Null or Empty - line 13 Test.cs
10	CA1709: Correct the casing of 'display' in member name 'Test.displayMessage2(string)' by changing it to 'Display'.	display is changed to Display - line 18 Test.cs
11	CA1062 In externally visible method 'Test.displayMessage2(string)', validate parameter 'msg' before using it.	Checked if msg is NOTnull - line 20 Test.cs
12	CA1709: Correct the casing of 'display' in member name 'Test.displayMessage3(string)' by changing it to 'Display'.	display is changed to Display - line 30 Test.cs
13	CA1062: In externally visible method ' Test.displayMessage3(string)', validate parameter 'msg' before using it.	Checked if msg is NOTnull - line 32 Test.cs
14	CA1709: Correct the casing of 'display' in member name 'Test.displayMessage4(string)' by changing it to 'Display'.	display is changed to Display - line 42 Test.cs
15	CA2241: Method 'Test.displayMessage4(string)' calls 'string.Format(string, object)' and does not provide an argument for format item "{1}". The provided format string is: '"{0}: {1}"'	Removed format {1} - lines 45,47 Test.cs
16	CA1062: In externally visible method 'Test.displayMessage4(string)', validate parameter 'msg' before using it.	Checked if msg is NOTnull - line 44 Test.cs

RECOMMENDATION

As demonstrated with these examples it is easy to integrate code analysis along with teaching programming structure and design. Moreover depending on the level of target students one can easily develop customized rules. It is recommended that instructors teaching the course customize the rule sets appropriate for the target students. For the introductory programming students, simple rule sets may be generated that will address adherence to naming conventions, design issues, and code maintainability [10]. For intermediate or advanced level students more complex rule sets may be developed that will address security, performance and other issues [10].

CONCLUSIONS

This paper discusses what Code Analysis is and how it can be used in the classroom. It is difficult to replicate real-world code review in the class room. However, integrating widely available tools like Code Analysis of Visual Studio IDE with programming examples and assignments will give students exposure to good programming practices. Moreover, Code Analysis provides a systematic mechanism of reviewing of source code. Instructors can easily create customized rule sets and let the students test their codes against these customized rule sets. It will facilitate students to spot and fix mistakes overlooked during programming, thereby improving both the overall quality of software and the programmers' skills. Code analysis rule sets can easily be tailored to the target student's level of proficiency in programming. It is recommended that simple rule sets are introduced in the introductory programming course whereas the more advanced rule sets are discussed and taught in more advanced programming courses. It is suggested that beginner students use only a small subset of rule sets such as naming conventions, basic design, and code maintainability rules. However, the code review and analysis technique using many of the advance rule sets would be more appropriate for advanced students. Although there was no formal survey taken, the author feels that student's understanding of program structure, and syntax, will be greatly enhanced if Code Analysis is incorporated early in programming learning stage using simple rule sets. The students will have better understanding of codes, will be able to develop, implement consistent code, and will learn to detect common program defects early. Majority of the students feel that code analysis has improved their understanding of program structure and helped them writing cleaner code. Using code analysis tool, students will have the opportunity to create C# applications and practice developing professional code right from the beginning.

REFERENCES

1. Gomes, A., Mendes, A.J. (2007) 'An environment to improve programming education', Proceedings of the 2007 international conference on Computer systems and technologies, June 14-15, Bulgaria, pp. IV.19-1–IV.19-6.Available: http://ecet.ecs.ru.acad.bg/cst07/Docs/cp/sIV/IV.19.pdf
2. Butler, M., Morgan, M. (2007) 'Learning challenges faced by novice programming students studying high level and low feedback concepts ', Proceedings of ascilite 2007 conference ,Singapore, December 2-5, pp .99-107.Available: http://www.ascilite.org.au/conferences/singapore07/procs/butler.pdf
3. Matthíasdóttir, A (2006) 'How to teach programming languages to novice students? Lecturing or not? ', International Conference on Computer Systems and Technologies - CompSysTech'06, Bulgaria, June 15-16, pp. IV.13-1–IV.13-7.Available: http://ecet.ecs.ru.acad.bg/cst06/Docs/cp/sIV/IV.13.pdf
4. Tatarchenko, E (2012) 'Analysis of Performing Code Review in the Classroom', Thesis for Master of Engineering in Electrical and Computer Science, Massachusetts Institute of Technology. Available: http://groups.csail.mit.edu/uid/other-pubs/elena-thesis.pdf
5. Parasoft® dotTEST™ Available :http://www.parasoft.com/dottest
6. CodeIt.Right Available: http://submain.com/products/codeit.right.aspx
7. CodeRush for Visual Studio : https://www.devexpress.com/Products/CodeRush/
8. MSDN documentation -'Code Analysis Rule Set Reference for Managed Code' Available: http://msdn.microsoft.com/en-us/library/dd264925%28v=vs.100%29.aspx
9. MSDN documentation –'Microsoft Basic Correctness Rules Code Analysis Rule Set'. Available: http://msdn.microsoft.com/en-us/library/dd264935%28v=vs.100%29.aspx
10. MSDN documentation – 'Code Analysis for Managed Code Warnings'. Available: http://msdn.microsoft.com/en-us/library/ee1hzekz%28v=vs.100%29.aspx
11. MSDN documentation – 'Walkthrough: Configuring and Using a Custom Rule Set ' Available: http://msdn.microsoft.com/en-us/library/dd264949%28VS.100%29.aspx

APPENDIX I: SCREEN SHOTS OF SELECTED EXAMPLES

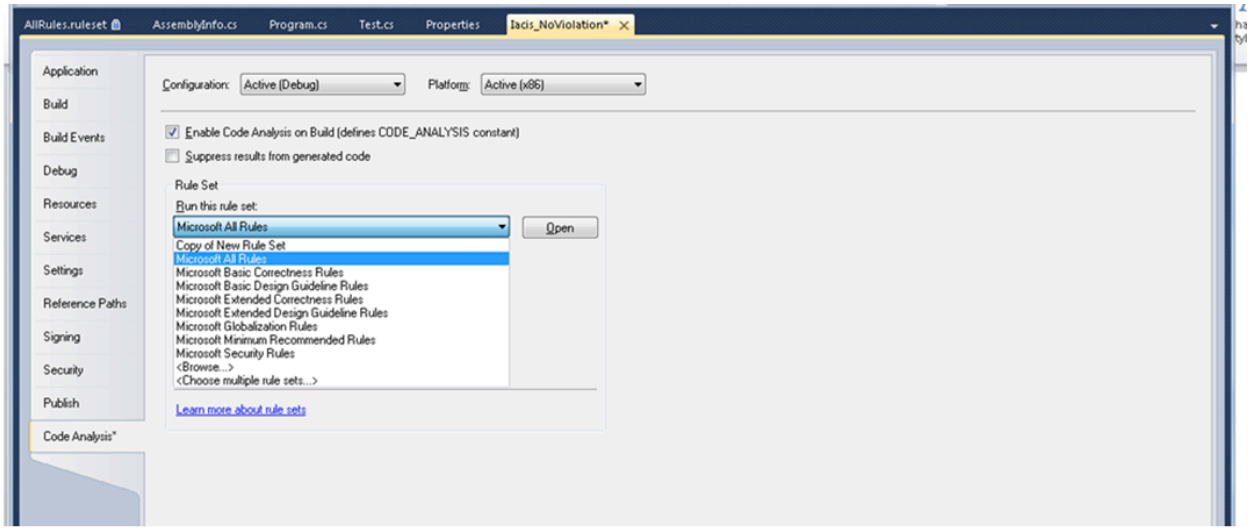

Figure 1: Rule Set Selection

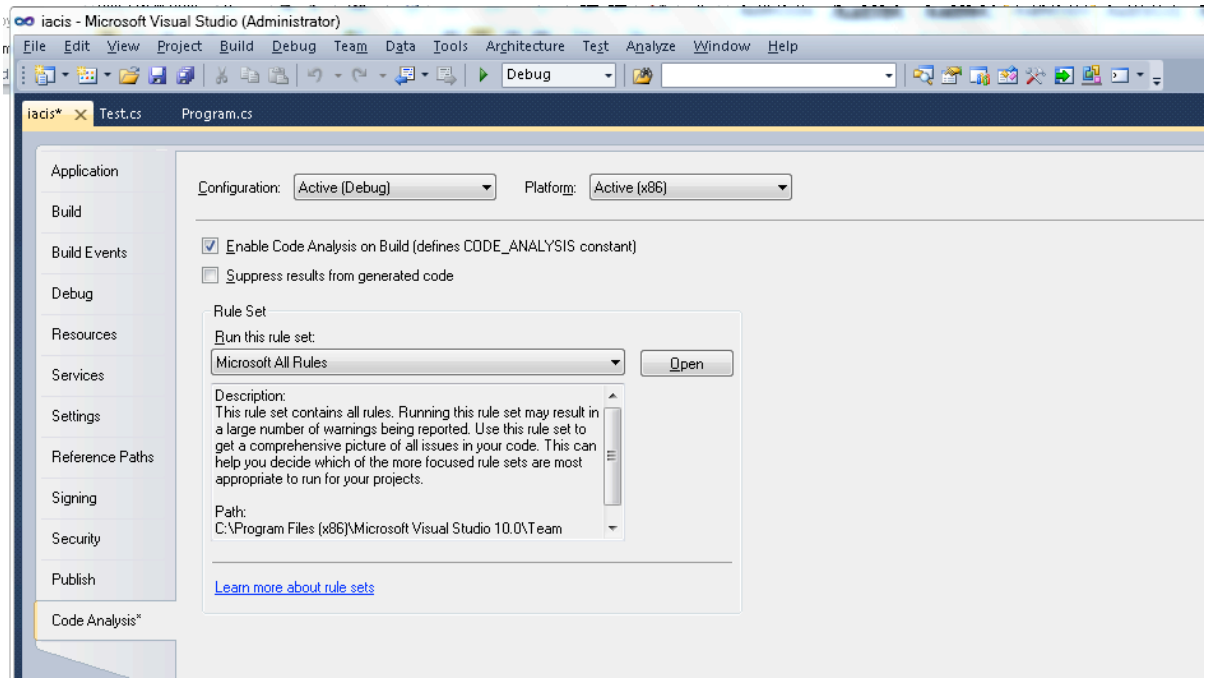

Figure 2: MicroSoft All Rules Selected using Code Analysis Tab of Visual Studio

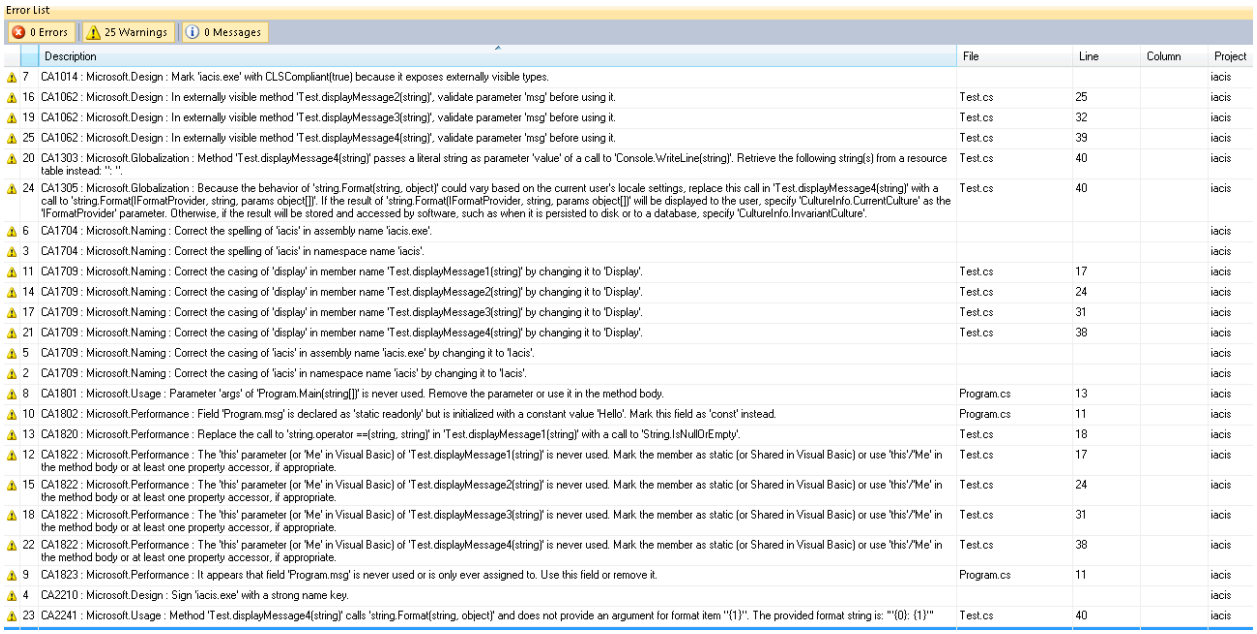

Figure 3: Warning Lists of Violations detected by Code Analysis on the example program when Maximum Rules

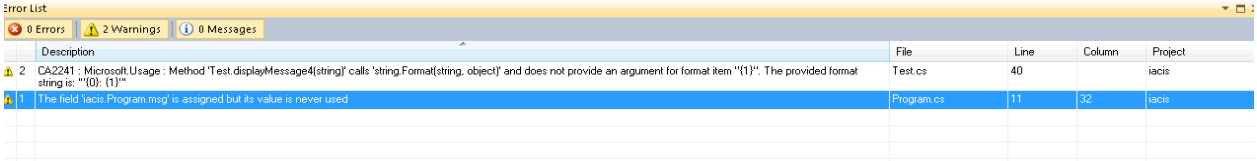

Figure 4: Warning Lists of Violations detected by Code Analysis on the example program when Minimum Rules

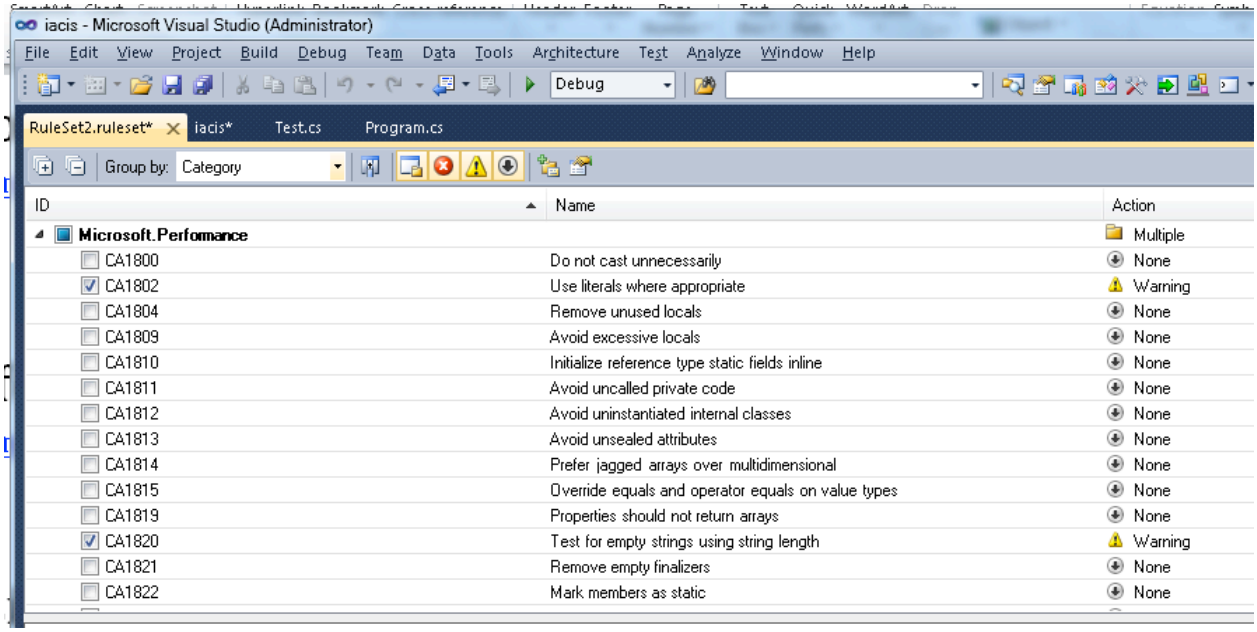

Figure 5: Shows sample Custom Rules

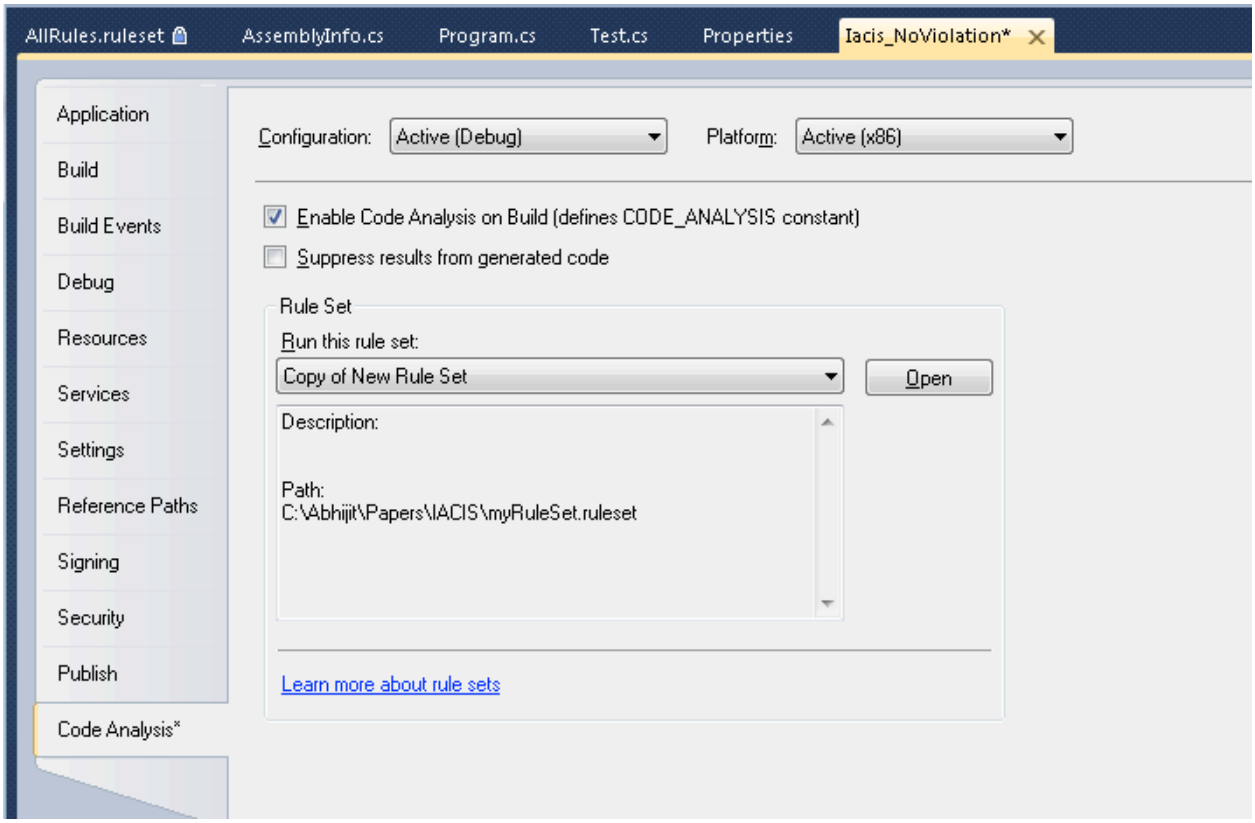

Figure 6: Select the Custom Rule

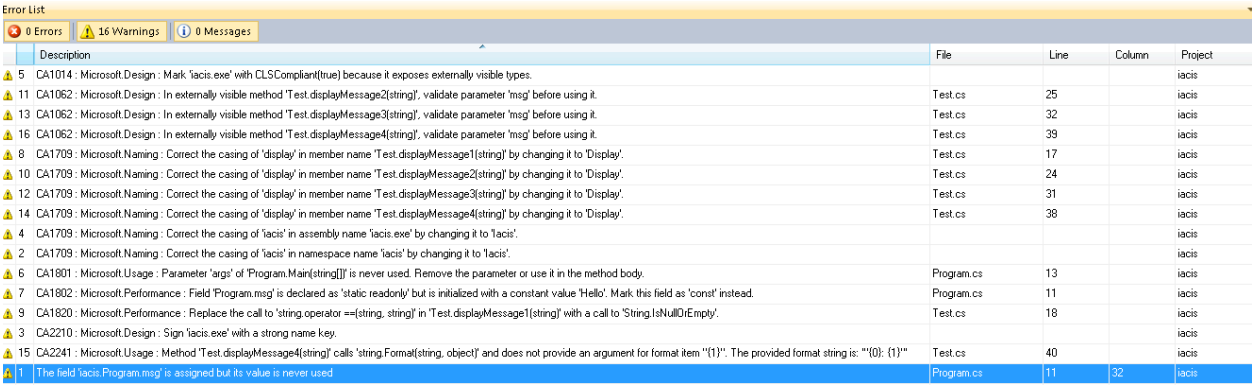

Figure 7: Warning Lists of Violations detected by Code Analysis on the example program using Custom

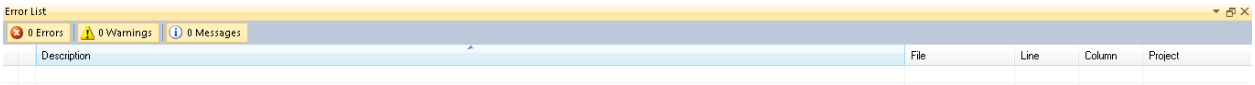

Figure 8: No Violations detected by Code Analysis on the example program after fixes

APPENDIX II: Original C# Source Code

Test.CS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace iacis
{
    public class Test
    {

    public void displayMessage1(string msg)
        {
            if (msg== "")
                Console.WriteLine(msg);
            return;
        }

    public void displayMessage2(string msg)
        {
            if (msg.Length != 0)
                Console.WriteLine(msg);
            return;
        }

    public void displayMessage3(string msg)
        {
            if (msg.Length != 0)
                Console.WriteLine(msg);
            return;
        }

    public void displayMessage4(string msg)
        {
            if (msg.Length != 0)

               Console.WriteLine(String.Form
               at("{0}: {1}", msg));
                return;
            }


    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;



namespace iacis
{

    class Program
    {
      static readonly string msg = "Hello";
      static void Main(string[] args)
        {


        }
    }
}
```

APPENDIX III: Final Version of C# Source Code

Corrected Program

Test.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Iacis_NoViolation
{
    public class Test
    {

    public void DisplayMessage1(string msg)
        {
           if (!string.IsNullOrEmpty (msg))
                Console.WriteLine(msg);
            return;
        }

    public void DisplayMessage2(string msg)
        {
            if (msg != null)
                Console.WriteLine(msg);
            else
            {

               Console.WriteLine("Empty
               String");

            }
            return;
        }

   public void DisplayMessage3(string msg,
string title)
        {
            if (msg != null)
                Console.WriteLine(msg);
            else
            {
                Console.WriteLine(title);
            }
            return;
        }

    public void DisplayMessage4(string msg)
        {
            if (msg != null)

               Console.WriteLine(String.Form
               at("{0}", msg));
            else

               Console.WriteLine(String.Form
               at("{0}", "Empty String"));
            return;
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Iacis_NoViolation
{
    class Program
    {
         const string msg = "Hello";

        static void Main()
        {
            Test myTest = new Test();

            myTest.DisplayMessage1(msg);
        }
    }
```