Andrei Fangli, class 248

Software Quality - Automatic Code Review

Assignment

Choose a Code review tool and apply it for your app. Your task is to produce a report about the use of the chosen tool. Results for your app, advantages, disadvantages.

FxCop for automatic code review

FxCop is a free tool provided by Microsoft with Visual Studio that statically analyses code and determines when a code analysis rule (such as implementing IDisposable pattern, comparing strings without specifying any options for casing) is broken. The usefulness of the tool is that it determines common mistakes done by developers and thus reduces bugs and cleans code by suggesting common solutions.

To use FxCop simply go to *View -> Other Windows -> Code Analysis*, from the Code Analysis Window go to *Analyze -> Solution* to analyze all projects. **Note!** The solution must be able to be built.

Advantages

- **End user documentation**
  The end user documentation is complete. Rules are grouped into categories, each rule and category having a detailed description and the reason for which it has been added. For a full list of analysis rules see: https://msdn.microsoft.com/en-us/library/ee1hzekz(v=vs.140).aspx.
- **Many predefined rules**
  FxCop comes with not one but two handfuls of predefined code analysis rules for naming (e.g.: correct spelling, using Pascal case for type names), performance (e.g.: overriding Equals for value types, remove empty finalizers and so on), security (e.g.: using SQL command parameters to avoid SQL injection), reliability (e.g.: disposing objects before losing scope) and other categories.
- **Visual Studio Integration**
  FxCop comes with Visual Studio and it is fully integrated with the IDE. One can configure the rules that should be included in code analysis per project as well as whether to run analysis for a project after it has been built. After an analysis is ran a list of found issues is presented to the developer with the ability to navigate to the location where the issue is found.
  Some rules have exceptions, there are cases where the rule does not apply (e.g.: throwing argument exceptions in property setters using the property name as the argument name rather than “value” which does not tell which property throws the exception). When a rule is broken the developer can chose to suppress it from the code analysis result window either by a specifying it in a separate exception file or by using attributes in code (other developers will see that the rule is suppressed however this clutters code).
- **Configurability**
  Code analysis rules are configured per project. For each project is specified what rules to apply. The tool comes with a few predefined pre-defined profiles easing the configuration. A new code analysis profile can be created either with no selected rules (empty) or starting from one of the predefined ones. To automate even further the tool can be configured to run after each build.

Disadvantages

- **Lack of developer documentation**
  FxCop has little to none documentation for developers willing to define their own code analysis rules. Adding new rules is difficult and debugging them is even more difficult as the whole process of extending the current set of analysis rules is a trial and error process.
- **Introspection**
  FxCop scans the resulting assembly, Common Intermediate Language rather than source code to determine when rules are not followed. Source code is subject to compiler optimizations, the resulting assembly is not a 1 to 1 match with the source code allowing the latter to contain issues that are removed at compile time. One would say that it is an understandable compromise because if the result respects all rules then there is no practical issue, only that the source code is a bit unclean.
- **Performance**
  By using introspection the tool is forced to do mappings between source code files and CIL to determine the exact location from where the issue arises. This has an impact the overall performance as such mappings can take quite some time on large projects.

FxCop results for "Edesia" project

"Edesia" is my Bachelor of Computer Science degree project. It is an online shop for food (hence the name "Edesie" – roman goddess of feasting and food) that also is configurable for planning deliveries into tasks to delivery zones (specified by an administrator). For employees to be able to track their tasks the application also provides a basic workflow for each delivery task. The list containing all issues found using "Microsoft Managed Recommended Rules" code analysis profile:

- CA1063 **Implement IDisposable correctly**
  Provide an overridable implementation of Dispose(bool) on 'DeliveryRepository' or mark the type as sealed. A call to Dispose(false) should only clean up native resources. A call to Dispose(true) should clean up both managed and native resources.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 10)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Add(OrderDetails)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 112)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Update(Order)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 130)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetOrders(params OrderState[])' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 176)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetOrders(ApplicationUser, params OrderState[])' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 225)

- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Add(string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 246)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Remove(string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 268)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetStreets()' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 284)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetUnmappedStreets()' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 301)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Add(DeliveryZone)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 338)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Update(DeliveryZone, string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 387)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Remove(DeliveryZone)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 416)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetDeliveryZone(string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 442)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetDeliveryZones()' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 483)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Add(DeliveryTaskDetails)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 523)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Add(IEnumerable<DeliveryTaskDetails>)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 572)

- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.Update(DeliveryTask)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 603)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetDeliveryTask(int)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 628)
- CA2202 **Do not dispose objects multiple times**
  Object 'deliveryTaskDataReader' can be disposed more than once in method 'DeliveryRepository.GetDeliveryTasks(params TaskState[])'. To avoid generating a System.ObjectDisposedException you should not call Dispose more than one time on an object.: Lines: 667
  Andrei15193.Edesia - DeliveryRepository.cs (Line 667)
- CA2202 **Do not dispose objects multiple times**
  Object 'deliveryTaskDataReader' can be disposed more than once in method 'DeliveryRepository.GetDeliveryTasks(params TaskState[])'. To avoid generating a System.ObjectDisposedException you should not call Dispose more than one time on an object.: Lines: 667
  Andrei15193.Edesia - DeliveryRepository.cs (Line 667)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetDeliveryTasks(params TaskState[])' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 670)
- CA2202 **Do not dispose objects multiple times**
  Object 'deliveryTaskDataReader' can be disposed more than once in method 'DeliveryRepository.GetDeliveryTasks(Employee, params TaskState[])'. To avoid generating a System.ObjectDisposedException you should not call Dispose more than one time on an object.: Lines: 715
  Andrei15193.Edesia - DeliveryRepository.cs (Line 715)
- CA2202 **Do not dispose objects multiple times**
  Object 'deliveryTaskDataReader' can be disposed more than once in method 'DeliveryRepository.GetDeliveryTasks(Employee, params TaskState[])'. To avoid generating a System.ObjectDisposedException you should not call Dispose more than one time on an object.: Lines: 715
  Andrei15193.Edesia - DeliveryRepository.cs (Line 715)
- CA2002 **Do not lock on objects with weak identity**
  'DeliveryRepository.GetDeliveryTasks(Employee, params TaskState[])' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 718)
- CA1063 **Implement IDisposable correctly**
  Modify 'DeliveryRepository.Dispose()' so that it calls Dispose(true), then calls GC.SuppressFinalize on the current object instance ('this' or 'Me' in Visual Basic), and then returns.
  Andrei15193.Edesia - DeliveryRepository.cs (Line 724)

- CA2202 **Do not dispose objects multiple times**
  Object 'solutionEnumerator' can be disposed more than once in method 'DeliveryTaskController._GetDeliveryTasks(KeyValuePair<DeliveryZone, IEnumerable<Order>>)'. To avoid generating a System.ObjectDisposedException you should not call Dispose more than one time on an object.: Lines: 166
  Andrei15193.Edesia - DeliveryTaskController.cs (Line 166)
- CA2237 **Mark ISerializable types with SerializableAttribute**
  Add [Serializable] to 'DomainConstraintException' as this type implements ISerializable.
  Andrei15193.Edesia - DomainConstraintException.cs (Line 4)
- CA1063 **Implement IDisposable correctly**
  Provide an overridable implementation of Dispose(bool) on 'ProductRepository' or mark the type as sealed. A call to Dispose(false) should only clean up native resources. A call to Dispose(true) should clean up both managed and native resources.
  Andrei15193.Edesia - ProductRepository.cs (Line 9)
- CA2002 **Do not lock on objects with weak identity**
  'ProductRepository.GetProduct(string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - ProductRepository.cs (Line 41)
- CA2002 **Do not lock on objects with weak identity**
  'ProductRepository.GetProducts()' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - ProductRepository.cs (Line 57)
- CA2002 **Do not lock on objects with weak identity**
  'ProductRepository.Add(Product)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - ProductRepository.cs (Line 80)
- CA2002 **Do not lock on objects with weak identity**
  'ProductRepository.Remove(string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - ProductRepository.cs (Line 98)
- CA1063 **Implement IDisposable correctly**
  Modify 'ProductRepository.Dispose()' so that it calls Dispose(true), then calls GC.SuppressFinalize on the current object instance ('this' or 'Me' in Visual Basic), and then returns.
  Andrei15193.Edesia - ProductRepository.cs (Line 103)
- CA1063 **Implement IDisposable correctly**
  Provide an overridable implementation of Dispose(bool) on 'UserRepository' or mark the type as sealed. A call to Dispose(false) should only clean up native resources. A call to Dispose(true) should clean up both managed and native resources.
  Andrei15193.Edesia - UserRepository.cs (Line 12)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.Users.get()' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 77)

- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.GetEmployees()' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 93)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.ConfirmUser(string, string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 119)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.Add(ApplicationUser, string, string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 148)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.Update(ApplicationUser)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 194)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.Find(string)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 214)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.Find(string, string, AuthenticationTokenType)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 258)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.SetAuthenticationToken(ApplicationUser, string, AuthenticationTokenType)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 287)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.AddToShoppingCart(ApplicationUser, ShoppingCartEntry)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 305)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.UpdateShoppingCart(ApplicationUser, ShoppingCartEntry)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 322)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.RemoveFromShoppingCart(ApplicationUser, Product)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 340)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.Update(ShoppingCart)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 368)

- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.ClearShoppingCart(ApplicationUser)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 383)
- CA2002 **Do not lock on objects with weak identity**
  'UserRepository.GetShoppingCart(ApplicationUser)' locks on a reference of type 'SqlConnection'. Replace this with a lock against an object with strong-identity.
  Andrei15193.Edesia - UserRepository.cs (Line 403)
- CA1063 **Implement IDisposable correctly**
  Modify 'UserRepository.Dispose()' so that it calls Dispose(true), then calls GC.SuppressFinalize on the current object instance ('this' or 'Me' in Visual Basic), and then returns.
  Andrei15193.Edesia - UserRepository.cs (Line 409)
- CA2213 **Disposable fields should be disposed**
  'UserRepository' contains field 'UserRepository._hashAlgorithm' that is of IDisposable type: 'HashAlgorithm'. Change the Dispose method on 'UserRepository' to call Dispose or Close on this field.
  Andrei15193.Edesia - UserRepository.cs (Line 409)
- CA1001 **Types that own disposable fields should be disposable**
  Implement IDisposable on 'XmlApplicationUserRepository' because it creates members of the following IDisposable types: 'SHA256Managed'. If 'XmlApplicationUserRepository' has previously shipped, adding new members that implement IDisposable to this type is considered a breaking change to existing consumers.
  Andrei15193.Edesia - XmlApplicationUserRepository.cs (Line 15)
- CA2237 **Mark ISerializable types with SerializableAttribute**
  Add [Serializable] to 'XmlSchemaConstraintException' as this type implements ISerializable.
  Andrei15193.Edesia - XmlSchemaConstraintException.cs (Line 5)
- CA1063 **Implement IDisposable correctly**
  Provide an overridable implementation of Dispose(bool) on 'XmlTransaction' or mark the type as sealed. A call to Dispose(false) should only clean up native resources. A call to Dispose(true) should clean up both managed and native resources.
  Andrei15193.Edesia - XmlTransaction.cs (Line 5)
- CA1063 **Implement IDisposable correctly**
  Modify 'XmlTransaction.Dispose()' so that it calls Dispose(true), then calls GC.SuppressFinalize on the current object instance ('this' or 'Me' in Visual Basic), and then returns.
  Andrei15193.Edesia - XmlTransaction.cs (Line 46)