Description

Company: Company Name
Program: Program Name
Language: C#
Last Version Complete: XXX

Revision History

Code Review Rev No.	Purpose/Description	Program Version	Reviewer	Start Date	End Date	Peer Reviewer	Peer Review Date

TEMPLATE Approver	Date	Rev
John Schweitzer	4/30/2008	05-02

File's Functions Line Count Consolidated Report for

Company Name, Program Name

On the Application level, no more than 50%
exceeding 60 lines, no more than 5%
exceeding 120 lines, and none exceeding 240

Code Review Supervisor: Replace example languages and insert lines as needed. When the reports are compiled, link the cells in the table below to the appropriate spreadsheet files.

Language	Less than 60+N/A	60 to 120	120 to 240	More than 240	Total
C#	#REF!	#REF!	#REF!	#REF!	#REF!
Total %	#REF!	#REF!	#REF!	#REF!	#REF!

VVSG Section									VVSG v.1: 5.2.2
General Description	**Review Date**	**Reviewer Name**	**Program Name**	**Program Version**	**TDP Date**	**File Path**	**File name**	**Class or Function name**	**Self-modifying code**
VVSG or Standard Description									Self-modifying, dynamically loaded, or interpreted code is prohibited
Language or Vendor Adjustments									

Total Accept	0
Total Reject	0
Total N/A	0
Total Blank	5
Total	5

VVSG v.1: 5.2.3.a	VVSG v.1: 5.2.3.b	Hart Verity Coding Std sec.: 3.2	VVSG v.1: 5.2.3.b 5.2.7.a	VVSG v.1: 5.2.3.c	VVSG v.1: 5.2.3.e	VVSG v.1: 5.2.3.e	VVSG v.1: 5.2.3.f	VVSG v.1: 5.2.4.a	VVSG v.1: 5.2.4.l	VVSG v.1: 5.2.4.ii
Specific function	**Module has unique name**	**Hart Naming Conventions**	**Module has header**	**Required resources**	**Single Entry Point**	**Single Exit Point**	**Control structures**	**Acceptable Constructs**	**Vendor Defined Constructs with Justificatio**	**Execution through Control Constructs**
Module performs a specific function	Uniquely and mnemonically named using names that differ by more than a single character	3.2.1 Use PascalCasing for classes enumerations, methods, functions, public properties, events, resources, and	Header describes purpose, other units needed, inputs, outputs, files read or written, globals, revision records (for modules greater than 10 lines) Header comments shall provide the following information: 1) The purpose of the	All required resources, such as data accessed by the module, should either be contained within the module or	Module has a single entry point	Module has a single exit point	Sequence, Conditionals, Top & Bottom tested loops, Switch..Case, For loops	Acceptable constructs are Sequence, If-Then-Else, Do-While, Do-Until, Case, and the General loop (including	If the programming language used does not provide these control constructs, the vendor shall provide them (that	While some programming languages do not create programs as linear processes, stepping from an initial condition.
			Hart uses /// comments for classes, properties, and methods							
									N/A	
									N/A	
									N/A	
									N/A	
									N/A	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	5	0
5	5	5	5	5	5	5	5	5	0	5
5	5	5	5	5	5	5	5	5	5	5

VVSG v.1: 5.2.4.iii	MS Std sec.: 2.2	Verity Std sec.: 2.1	MS Std sec.: 2.5	MS Std sec.: 2.5	Verity Std sec.: 3.8.4	MS Std sec.: 2.6	MS Std sec.: 2.7	MS Std sec.: 2.8	MS Std sec.: 2.8.1	MS Std sec.: 2.10
Program re-direction	**Do not use tabs**	**Line length**	**Local variables have minimum scope**	**Local variable declaration and initialization**	**Initialize pointer variables**	**Parameters ordered in groups**	**One statement per line**	**Use enums**	**Flag enums**	**Braces and indentation**
Logic that evaluates received or stored data shall not re-direct program control	All code should be written using four spaces for indentation.	Maximum line length is 150 characters for C#	Do declare local variables in the minimum scope block that can contain them, typically just before use if the language allows; otherwise, at	Do initialize variables when they are declared. Do declare and initialize/assign local variables on a single line where the language	Initialize pointers (when) they are declared (and) set them to NULL after freeing them.	Do order parameters, grouping the in parameters first, the out parameters last.	Do not put more than one statement on a single line	Do use an enum to strongly type parameters, properties, and return values that represent sets of values. Do not use an enum for	Do apply the System.Flags Attribute to flag enums in .NET. Do not apply this attribute to simple enums. Do use powers of two for the flags	Do use Allman bracing style. The style puts the brace associated with a control statement on the next line, indented to the same
		Hart allows 150 chars per line for C# and		Hart claims that variable initialization is automatic		Very similar to MS Std sec.: 4.9.4				
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5

Verity Std sec.: 3.7.3	MS Std sec.: 2.11.2	MS Std sec.: 2.11.6	MS Std sec.: 4.2	MS Std sec.: 4.4.1	MS Std sec.: 4.4.3	MS Std sec.: 4.5	MS Std sec.: 4.6	MS Std sec.: 4.7	MS Std sec.: 4.8	MS Std sec.: 4.9.1
Braces around single line conditionals	**File has header comments**	**TODO comments**	**File named for single contained public type**	**Meaningful names**	**No Hungarian notation**	**Constant fields**	**String operations**	**Array and Collection operations**	**Value types implement IEquatable <T>**	**Class instance fields are private and exposed**
Use braces around single line conditionals	Do have a file header comment at the start of every human-created code file. The header comment templates are as follows:	Do not use TODO comments in any released samples	Do not have more than one public type in a source file, unless they differ only in the number of generic parameters or one is nested in the	Do use meaning names for various types, functions, variables, constructs and types. Do not use underscores, hyphens, or any other non-	Do not use Hungarian notation (i.e., do not encode the type of a variable in its name) in .NET.	Do use constant fields for constants that will never change. Do use public static (shared) readonly fields for	* Do use overloads that explicitly specify the string comparison rules for string operations. Typically, this involves	* Do not use read-only array fields. The field itself is read-only and can't be changed, but elements in the array can be	* Do implement IEquatable< T> on value types.	Do not provide instance fields that are public or protected. Public and protected fields do not version well and are not protected by
				Hart allows undescore characters in names.						Hart allows protected instance fields
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5

MS Std sec.: 4.9.2	MS Std sec.: 4.9.3	MS Std sec.: 4.9.4	MS Std sec.: 4.9.4	MS Std sec.: 4.9.6	MS Std sec.: 4.9.10	MS Std sec.: 4.11.1	MS Std sec.: 4.11.2	MS Std sec.: 4.12	MS Std sec.: 4.12.1	MS Std sec.: 4.12.2
Set-only properties are not allowed	**Do not call virtual members on an object**	**Out parameters follow all of the pass-by-value and**	**Validate arguments to public, protected, or explicit**	**Member overloading**	**Abstract types' constructors**	**Throw specific exceptions. Do not return error codes.**	**Catch only specific errors**	**Do not force garbage collections**	**Do not use catch blocks for cleanup code.**	**Basic Dispose Pattern**
Do not provide set-only properties. If the property getter cannot be provided, use a method to implement	Do not call virtual members on an object inside its constructors. Calling a virtual member causes the most-derived	Do place all out parameters after all of the pass-by-value and ref parameters (excluding parameter arrays), even if this	Do validate arguments passed to public, protected, or explicitly implemented members. Throw System.ArgumentException, or one	* Do use member overloading rather than defining members with default arguments. Default arguments are not CLS-compliant * Do not	* Do not define public or protected-internal constructors in abstract types. * Do define a protected or an internal	* Do report execution failures by throwing exceptions. Do not return error codes. * Do throw the most specific (the most derived) exception that makes sense. For example, throw ArgumentNullException and not its base type	Do catch only specific errors that the code knows how to handle. You should catch a more	Do not force garbage collections with GC.Collect.	Do use try-finally blocks for cleanup code and try-catch blocks for error recovery code. Do not use catch	* Do implement the Basic Dispose Pattern on types containing instances of disposable types. * Do extend the Basic
		Very similar to MS Std sec.: 2.6								Hart allows "using" blocks as alternate to
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
5	5	5	5	5	5	5	5	5	5	5
5	5	5	5	5	5	5	5	5	5	5

MS Std sec.: 4.12.3	Hart Verity Coding Std sec.: 2.2.2	Discrepancy					Fixed Discrepancy		
Types finalizable	**Methods and Functions must be less than 240 lines**	**Disc. #**	**Description**	**Date Opened**	**TDP Date**	**Reviewer Name**	**Date Closed**	**TDP Date**	**Reviewer Name**
* Do make a type finalizable, if the type is responsible for releasing an unmanaged resource that does not have its own finalizer. When implementing	Hart's internal Verity standard has made this optional at Feb. 2014 by stating: Executable code in methods and functions should be less than 240 lines.								
	N/A								
	N/A								
	N/A								
	N/A								
	N/A								

0 0
0 0
0 5
5 0
5 5

Notes

Source Code Review Detail

Client Name:	Company Name
Report Date:	
Program:	Program Name
Version:	XXX
Programming Language:	C#

Total Modules Reviewed	0
Total Modules with Discrepancy	0

VSS Req.	Criterion	Definition		
Vol. 1 Section 4.2.2-Integrity			**Accept**	**Reject**
VVSG v.1: 5.2.2	**Self-modifying code**	Self-modifying, dynamically loaded, or interpreted code is prohibited	0	0
Vol. 1 Section 4.2.3- Modularity			**Accept**	**Reject**
VVSG v.1: 5.2.3.a	**Specific function**	Module performs a specific function	0	0
VVSG v.1: 5.2.3.b	**Module has unique name**	Uniquely and mnemonically named using names that differ by more than a single character	0	0

VVSG v.1: 5.2.3.b 5.2.7.a	**Module has header**	Header describes purpose, other units needed, inputs, outputs, files read or written, globals, revision records (for modules greater than 10 lines) Header comments shall provide the following information: 1) The purpose of the unit and how it works; 2) Other units called and the calling sequence; 3) A description of input parameters and outputs; 4) File references by name and method of access (read, write, modify , append, etc.); 5) Global variables used; and 6) Date of creation and a revision record;	0	0
VVSG v.1: 5.2.3.c	**Required resources**	All required resources, such as data accessed by the module, should either be contained within the module or explicitly identified	0	0
VVSG v.1: 5.2.3.e	**Single Entry Point**	Module has a single entry point	0	0
VVSG v.1: 5.2.3.e	**Single Exit Point**	Module has a single exit point	0	0
VVSG v.1: 5.2.3.f	**Control structures**	Sequence, Conditionals, Top & Bottom tested loops, Switch..Case, For loops	0	0
Vol. 1 Section 4.2.4-Control Constructs			**Accept**	**Reject**
VVSG v.1: 5.2.4.a	**Acceptable Constructs**	Acceptable constructs are Sequence, If-Then-Else, Do-While, Do-Until, Case, and the General loop (including the special case for loop); Note: This criteria will be "Accept" except in the case where the vendor defines code constructs.	0	0

VVSG v.1: 5.2.4.I	**Vendor Defined Constructs with Justification**	If the programming language used does not provide these control constructs, the vendor shall provide them (that is, comparable control structure logic). The constructs shall be used consistently throughout the code. No other constructs shall be used to control program logic and execution;	0	0
VVSG v.1: 5.2.4.ii	**Execution through Control Constructs**	While some programming languages do not create programs as linear processes, stepping from an initial condition, through changes, to a conclusion, the program components nonetheless contain procedures (such as “methods” in object-oriented languages). Even in these programming languages, the procedures must execute through these control constructs (or their equivalents, as defined and provided by the vendor);	0	0
VVSG v.1: 5.2.4.iii	**Program re-direction**	Logic that evaluates received or stored data shall not re-direct program control	0	0
Vol. 1 Section 4.2.5-Naming Conventions			**Accept**	**Reject**
MS Std sec.: 2.2	**Do not use tabs**	All code should be written using four spaces for indentation.	0	0
Verity Std sec.: 2.1	**Line length**	Maximum line length is 150 characters for C#	0	0
Vol. 1 Section 4.2.6-Coding Conventions			**Accept**	**Reject**
MS Std sec.: 2.5	**Local variables have minimum scope**	Do declare local variables in the minimum scope block that can contain them, typically just before use if the language allows; otherwise, at the top of that scope block	0	0

MS Std sec.: 2.5	**Local variable declaration and initialization**	Do initialize variables when they are declared. Do declare and initialize/assign local variables on a single line where the language allows it. Do not declare multiple variables in a single line.	0	0
MS Std sec.: 2.6	**Parameters ordered in groups**	Do order parameters, grouping the in parameters first, the out parameters last.	0	0
#REF!	**#REF!**	#REF!	#REF!	#REF!
MS Std sec.: 2.7	**One statement per line**	Do not put more than one statement on a single line	0	0
MS Std sec.: 2.8	**Use enums**	Do use an enum to strongly type parameters, properties, and return values that represent sets of values. Do not use an enum for open sets. Do not use Enum.IsDefined for enum range checks in .NET.	0	0
MS Std sec.: 2.8.1	**Flag enums**	Do apply the System.FlagsAttribute to flag enums in .NET. Do not apply this attribute to simple enums. Do use powers of two for the flags enum values.	0	0
MS Std sec.: 2.10	**Braces and indentation**	Do use Allman bracing style. The style puts the brace associated with a control statement on the next line, indented to the same level as the control statement. Statements within the braces are indented to the next level.	0	0

MS Std sec.: 2.11.2	**File has header comments**	Do have a file header comment at the start of every human-created code file. The header comment templates are as follows: /**********\ Module Name: <File Name> Project: <Sample Name> <Description of the file> **********/	0	0
MS Std sec.: 2.11.6	**TODO comments**	Do not use TODO comments in any released samples	0	0
MS Std sec.: 4.2	**File named for single contained public type**	Do not have more than one public type in a source file, unless they differ only in the number of generic parameters or one is nested in the other. Multiple internal types in one file are allowed. Do name the source file with the name of the public type it contains.	0	0
MS Std sec.: 4.4.1	**Meaningful names**	Do use meaning names for various types, functions, variables, constructs and types. Do not use underscores, hyphens, or any other non-alphanumeric characters.	0	0
MS Std sec.: 4.4.3	**No Hungarian notation**	Do not use Hungarian notation (i.e., do not encode the type of a variable in its name) in .NET.	0	0
MS Std sec.: 4.5	**Constant fields**	Do use constant fields for constants that will never change. Do use public static (shared) readonly fields for predefined object instances.	0	0

MS Std sec.: 4.6	**String operations**	* Do use overloads that explicitly specify the string comparison rules for string operations. Typically, this involves calling a method overload that has a parameter of type StringComparison. * Do use StringComparison.Ordinal or StringComparison.OrdinalIgnoreCase for comparisons as your safe default for culture-agnostic string matching, and for better performance. * Do use string operations that are based on StringComparison.CurrentCulture when you display output to the user. * Do use the non-linguistic StringComparison.Ordinal or StringComparison.OrdinalIgnoreCase values instead of string operations based on CultureInfo.InvariantCulture when the comparison is linguistically irrelevant (symbolic, for example). * Do use an overload of the String.Equals method to test whether two strings are equal. * Do not use an overload of the String.Compare or CompareTo method and test for a return value of zero to determine whether two strings are equal. * Do use the String.ToUpperInvariant method instead of the String.ToLowerInvariant method when you normalize strings for comparison.	0	0

MS Std sec.: 4.7	**Array and Collection operations**	* Do not use read-only array fields. The field itself is read-only and can't be changed, but elements in the array can be changed. * Do use Collection<T> or a subclass of Collection<T> for properties or return values representing read/write collections, and use ReadOnlyCollection<T> or a subclass of ReadOnlyCollection<T> for properties or return values representing read-only collections. * Do not implement both IEnumerator<T> and IEnumerable<T> on the same type. * Do not return a null reference for Array or Collection.	0	0
MS Std sec.: 4.8	**Value types implement IEquatable<T>**	* Do implement IEquatable<T> on value types.	0	0
MS Std sec.: 4.9.1	**Class instance fields are private and exposed through properties**	Do not provide instance fields that are public or protected. Public and protected fields do not version well and are not protected by code access security demands. Instead of using publicly visible fields, use private fields and expose them through properties.	0	0
MS Std sec.: 4.9.2	**Set-only properties are not allowed**	Do not provide set-only properties. If the property getter cannot be provided, use a method to implement the functionality instead.	0	0
#REF!	**#REF!**	#REF!	#REF!	#REF!
MS Std sec.: 4.9.3	**Do not call virtual members on an object inside its constructors**	Do not call virtual members on an object inside its constructors. Calling a virtual member causes the most-derived override to be called regardless of whether the constructor for the type that defines the most-derived override has been called.	0	0

MS Std sec.: 4.9.4	**Out parameters follow all of the pass-by-value and ref parameters**	Do place all out parameters after all of the pass-by-value and ref parameters (excluding parameter arrays), even if this results in an inconsistency in parameter ordering between overloads.	0	0
MS Std sec.: 4.9.4	**Validate arguments to public, protected, or explicit members**	Do validate arguments passed to public, protected, or explicitly implemented members. Throw System.ArgumentException, or one of its subclasses, if the validation fails: If a null argument is passed and the member does not support null arguments, throw ArgumentNullException. If the value of an argument is outside the allowable range of values as defined by the invoked method, throw ArgumentOutOfRangeException.	0	0
Vol. 1 Section 4.2.7 -Comments			**Accept**	**Reject**
MS Std sec.: 4.9.6	**Member overloading**	* Do use member overloading rather than defining members with default arguments. Default arguments are not CLS-compliant * Do not arbitrarily vary parameter names in overloads. If a parameter in one overload represents the same input as a parameter in another overload, the parameters should have the same name. Parameters with the same name should appear in the same position in all overloads.	0	0
MS Std sec.: 4.9.10	**Abstract types' constructors**	* Do not define public or protected-internal constructors in abstract types. * Do define a protected or an internal constructor on abstract classes	0	0

MS Std sec.: 4.11.1	**Throw specific exceptions. Do not return error codes.**	* Do report execution failures by throwing exceptions. Do not return error codes. * Do throw the most specific (the most derived) exception that makes sense. For example, throw ArgumentNullException and not its base type ArgumentException if a null argument is passed. * Do not throw exceptions from exception filter blocks. * Do not explicitly throw exceptions from finally blocks. Implicitly thrown exceptions resulting from calling methods that throw are acceptable.	0	0
#REF!	**#REF!**	#REF!	#REF!	#REF!

#REF!	**#REF!**	#REF!	**Less than 60+N/A**	**60 to 120**	**120 to 240**	**More than 240**	**Total**
			#REF!	#REF!	#REF!	#REF!	#REF!
			#REF!	#REF!	#REF!	#REF!	#REF!