



DesignNews

Embedded Software Design Techniques

DAY 4 : Designing Quality into Embedded Systems

Sponsored by



© 2022Beningo Embedded Group, LLC. All Rights Reserved.

Webinar Logistics

- Turn on your system sound to hear the streaming presentation.
- If you have technical problems, click “Help” or submit a question asking for assistance.
- Participate in ‘Group Chat’ by maximizing the chat widget in your dock.
- Submit questions for the lecturer using the Q&A widget. They will follow-up after the lecture portion concludes.

Course Sessions

- Software Architectures 101
- Designing RTOS-based Applications
- Architecture Verification Techniques
- **Designing Quality into Embedded Systems**
- Software Configuration Management Techniques

1

What is Quality?

Software Quality

“Software Quality refers to two related but distinct notions:

1. Software functional quality reflects how well it complies with or conforms to a given design, based on functional requirements or specifications. That attribute can also be described as the fitness for purpose of a piece of software or how it compares to competitors in the marketplace as a worthwhile product. It is the degree to which the correct software was produced.
2. Software structural quality refers to how it meets non-functional requirements that support the delivery of the functional requirements, such as robustness or maintainability. It has a lot more to do with the degree to which the software works as needed.”

Software Quality

Functional Quality

- What management looks at
- Outside appearances
- Meets requirements

Structural Quality

- Low maintenance costs
- Scalable
- Great design
- Understandable
- Code is quality

Which software quality is most important to you?

- Functional software quality
- Structural software quality

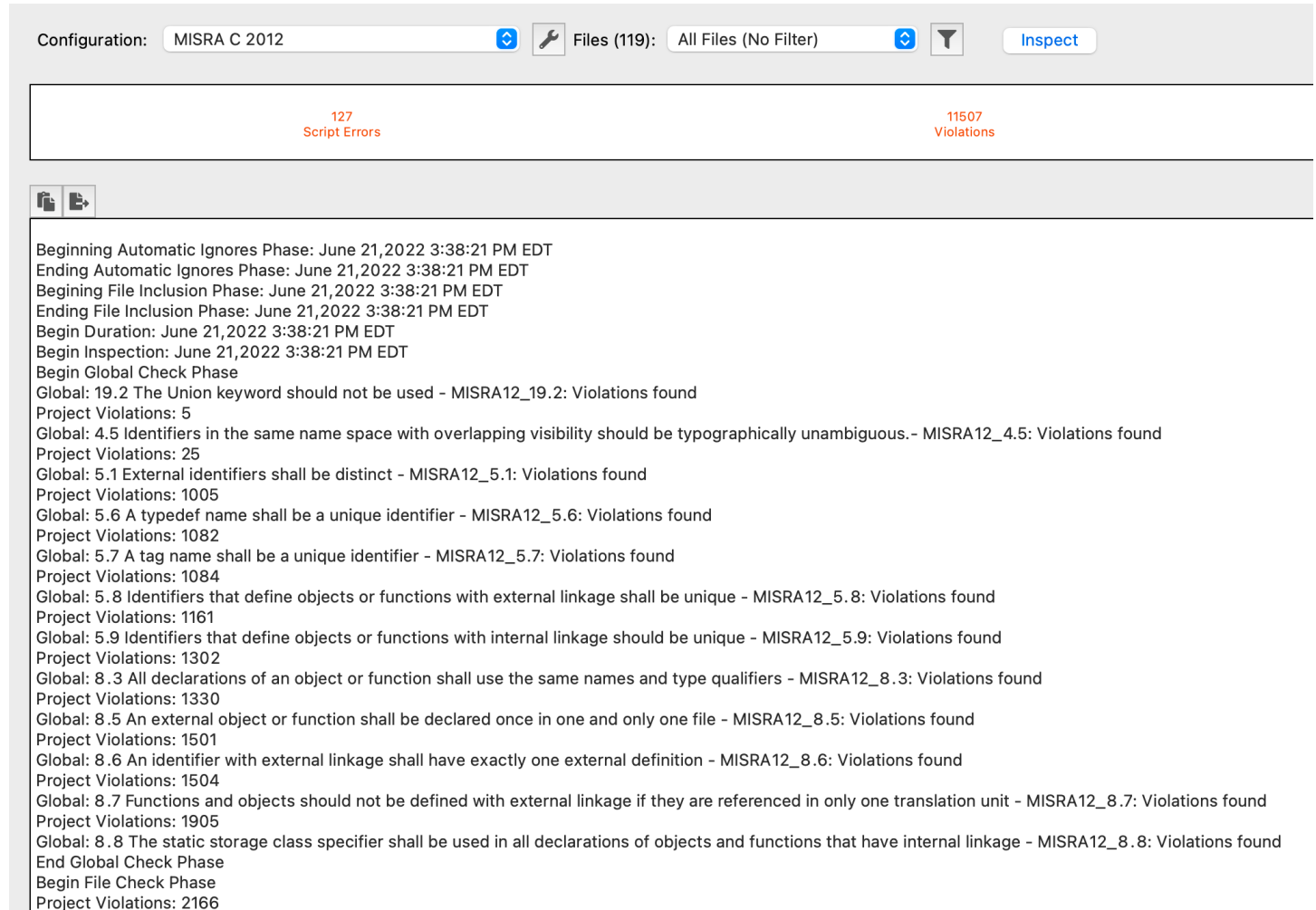
2 Code Quality

Defining Code Quality

- The code adheres to industry best practices and standards for the language
- The codes' function complexity is minimized and meets defined code metrics
- The code compiles without warnings and passes static code analysis
- The code unit test cases have 100% branch coverage
- The code has gone through the code review processes

Coding Standards

- GNU
- MISRA C / C++
- Cert C



Configuration: MISRA C 2012 Files (119): All Files (No Filter) [Inspect](#)

127 Script Errors 11507 Violations

Beginning Automatic Ignores Phase: June 21,2022 3:38:21 PM EDT
Ending Automatic Ignores Phase: June 21,2022 3:38:21 PM EDT
Beginning File Inclusion Phase: June 21,2022 3:38:21 PM EDT
Ending File Inclusion Phase: June 21,2022 3:38:21 PM EDT
Begin Duration: June 21,2022 3:38:21 PM EDT
Begin Inspection: June 21,2022 3:38:21 PM EDT
Begin Global Check Phase
Global: 19.2 The Union keyword should not be used - MISRA12_19.2: Violations found
Project Violations: 5
Global: 4.5 Identifiers in the same name space with overlapping visibility should be typographically unambiguous.- MISRA12_4.5: Violations found
Project Violations: 25
Global: 5.1 External identifiers shall be distinct - MISRA12_5.1: Violations found
Project Violations: 1005
Global: 5.6 A typedef name shall be a unique identifier - MISRA12_5.6: Violations found
Project Violations: 1082
Global: 5.7 A tag name shall be a unique identifier - MISRA12_5.7: Violations found
Project Violations: 1084
Global: 5.8 Identifiers that define objects or functions with external linkage shall be unique - MISRA12_5.8: Violations found
Project Violations: 1161
Global: 5.9 Identifiers that define objects or functions with internal linkage should be unique - MISRA12_5.9: Violations found
Project Violations: 1302
Global: 8.3 All declarations of an object or function shall use the same names and type qualifiers - MISRA12_8.3: Violations found
Project Violations: 1330
Global: 8.5 An external object or function shall be declared once in one and only one file - MISRA12_8.5: Violations found
Project Violations: 1501
Global: 8.6 An identifier with external linkage shall have exactly one external definition - MISRA12_8.6: Violations found
Project Violations: 1504
Global: 8.7 Functions and objects should not be defined with external linkage if they are referenced in only one translation unit - MISRA12_8.7: Violations found
Project Violations: 1905
Global: 8.8 The static storage class specifier shall be used in all declarations of objects and functions that have internal linkage - MISRA12_8.8: Violations found
End Global Check Phase
Begin File Check Phase
Project Violations: 2166

Metrics

- McCabe Cyclomatic Complexity
- Code churn (lines of code modified)
- CPU Utilization
- Assertion density
- Test case code coverage
- RTOS Task Periodicity (minimum, average, and max)

Does your code compile without warnings?

- Yes
- No
- N/A, my boss is looking over my shoulder

3

Cyclomatic Complexity and Code Coverage

Definition

McCabe Cyclomatic Complexity (Cyclomatic Complexity) is a measurement that can be performed on software that measures the number of linearly independent paths within a function.

- The minimum number of test cases required to test the function.
- The risk associated with modifying the function.
- The likelihood that the function contains undiscovered bugs.

Cyclomatic Complexity

Cyclomatic Complexity	Risk Evaluation
1 – 10	A simple function without much risk
11 – 20	A more complex function with moderate risk
21 – 50	A complex function of high risk
51 and greater	An untestable function of very high risk

Cyclomatic Complexity

Cyclomatic Complexity	Risk of Bug Injection
1 – 10	5%
11 – 20	20%
21 – 50	40%
51 and greater	60%

Code Coverage

PROBLEMS 9 OUTPUT TERMINAL DEBUG CONSOLE

```
35.71%  firmware/app/messaging/command.c
100.00%  firmware/app/controller/controller.c
100.00%  firmware/app/heaters/heater_sm.c
100.00%  firmware/app/messaging/crcGen16.c
100.00%  firmware/app/messaging/packet.c
100.00%  firmware/app/pump/a4964_config.c
mkdir -p gcov
mv *.gcov gcov
mv gcov_* gcov
See gcov directory for details
make[1]: Leaving directory '/home/app'
root@807a4a591a59:/home/app#
```

```
268      -: 264:*****
269      #####: 265:static void Command_Clear(uint8_t const * const Data)
270      -: 266:{
271      -: 267:    // This is just to use the passed parameters to remove static analysis error
272      -: 268:    // that we don't use the passed variable.
273      -: 269:    (void)Data;
274      #####: 270:}
```

Which of the following do you monitor?

- Cyclomatic Complexity
- Code Coverage
- Both
- Neither

4

Going Further

Thank you for attending

Please consider the resources below:

- www.beningo.com
 - Blog, White Papers, Courses
 - Embedded Bytes Newsletter
 - <http://bit.ly/1BAHYXm>
 - Embedded Software Design
 - <https://bit.ly/3PZCtNO>



From www.beningo.com under

- Blog > CEC – Embedded Software Design Techniques



DesignNews

Thank You

Sponsored by



© 2022Beningo Embedded Group, LLC. All Rights Reserved.