



## **White paper**

**Improving software quality with  
static source code analysis**

### COPYRIGHT NOTICE

© Copyright 2011 Atollic AB. All rights reserved. No part of this document may be reproduced or distributed without the prior written consent of Atollic AB.

### TRADEMARK

**Atollic, Atollic TrueSTUDIO, Atollic TrueINSPECTOR, Atollic TrueVERIFIER and Atollic TrueANALYZER** and the Atollic logotype are trademarks or registered trademarks owned by Atollic. ECLIPSE™ is a registered trademark of the Eclipse foundation. MISRA and "MISRA C" is a registered trademark of MIRA Ltd, held on behalf of the MISRA Consortium. All other product names are trademarks or registered trademarks of their respective owners.

### DISCLAIMER

The information in this document is subject to change without notice and does not represent a commitment of Atollic AB. The information contained in this document is assumed to be accurate, but Atollic assumes no responsibility for any errors or omissions. In no event shall Atollic AB, its employees, its contractors, or the authors of this document be liable for any type of damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

### DOCUMENT IDENTIFICATION

ASW-WPSA                      July 2011

### REVISION

First version

#### **Atollic AB**

Science Park  
Gjuterigatan 7  
SE- 553 18 Jönköping  
Sweden

+46 (0) 36 19 60 50

**E-mail:** [sales@atollic.com](mailto:sales@atollic.com)

**Web:** [www.atollic.com](http://www.atollic.com)

#### **Atollic Inc**

115 Route 46  
Building F, Suite 1000  
Mountain Lakes, NJ 07046-1668  
USA

+1 (973) 784 0047 (Voice)  
+1 (877) 218 9117 (Toll Free)  
+1 (973) 794 0075 (Fax)

**E-mail:** [sales.usa@atollic.com](mailto:sales.usa@atollic.com)

**Web:** [www.atollic.com](http://www.atollic.com)

# Contents

Abstract .....	1
Introduction.....	2
Finding errors earlier is cheaper.....	3
Static source code analysis.....	4
Coding standards compliance .....	5
Source code metrics.....	6
Embedded tools for static source code analysis .....	8
Summary.....	10

# Tables

**No table of figures entries found.**

## ABSTRACT

As embedded systems contain a lot more software today than only a couple of years ago, projects are exposed to an increased likelihood of distributing products with software problems.

It is possible to debug software to understand why a problem happens (which is a requirement to correct it), but a bug cannot be fixed unless you are aware of it. Some problems might be found during the testing phase, others are found by unhappy customers after your product is released.

A good strategy to improve the situation is to avoid problems in the first place, or at least fix them before the testing phase starts. This can be done automatically using static source code analysis and by gathering code metrics.

By deploying methods and tools outlined in this white paper, you and your team can deliver higher quality software with less effort.

# INTRODUCTION

Almost all software products contain errors. If you think that your product is an exception to that, it is most likely that you are just not aware of the bugs yet.

As modern 32-bit microcontrollers can contain hundreds of kilobytes, or even a megabyte or more of memory, there is a massive amount of software that goes into embedded products these days.

As code size increase, so do complexity and the number of software problems. There are various strategies to fight software quality issues, such as:

- Static source code analysis
- Code reviews
- Manual testing
- Automated unit testing
- Test quality measurement
- Etc

This white paper will discuss static source code analysis, including source code metrics, and present tool solutions that enable embedded developers to find potential problems automatically, thus enabling them to deliver higher-quality software with a minimum of effort.

## FINDING ERRORS EARLIER IS CHEAPER

Finding the cause of errors using a debugger is often necessary, but to fix a bug, it must first be detected. It is far cheaper to find the bug before the test phase is started, not to mention before the product is delivered to customers. Development teams should thus strive to find and correct bugs as early as possible in the development cycle.

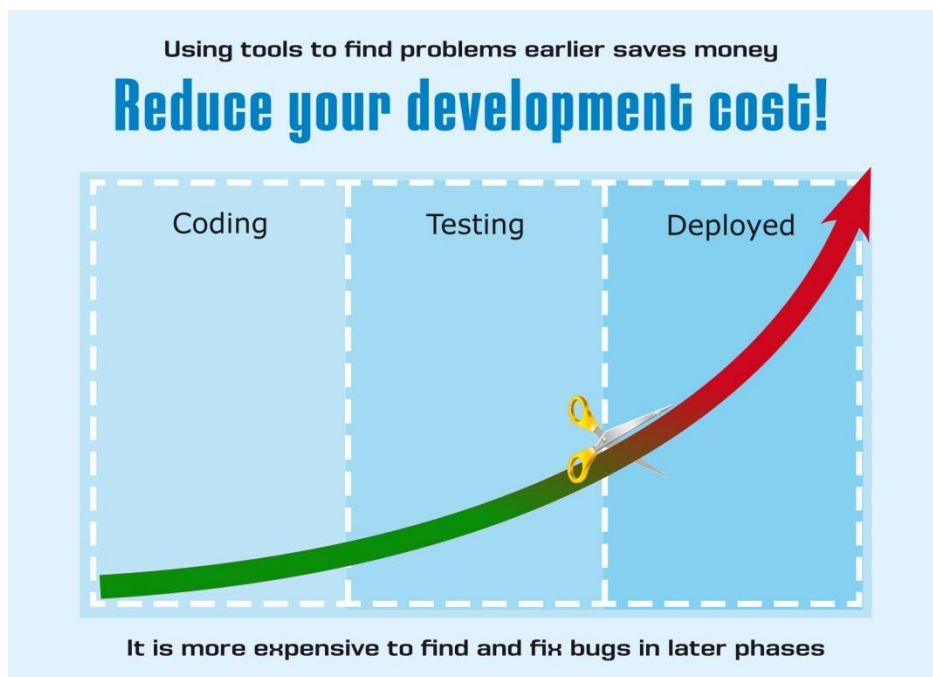


Figure 1 - It is cheaper to correct software problems early in the development process

It is clearly a good strategy for development teams to spend more efforts on software quality earlier in the development process, as any problem that is found and fixed during the development phase is cheaper than finding and fixing it during the testing or maintenance phase.

This can be done in different ways, such as using peer reviews or by deploying a tool for automated static source code analysis.

## STATIC SOURCE CODE ANALYSIS

Automated static source code analysis is the process where a software tool analyses the source code of an application, and automatically detects potential bugs or other types of problems in the source code.

The tool parses the source code of the application and analyzes how the source code is written. Static source code analysis is typically divided into two different areas:

- Coding standards compliance
- Source code metrics

Most tools that perform static source code analysis check the coding style versus a formal coding standard. Coding standards typically limits the programmer's flexibility and only permits using source code constructs that promote safety, reliability, maintenance and portability. In effect, potentially "dangerous" language constructs are avoided.

There are many different coding standards in use, and the most popular one in the embedded industry is currently MISRA<sup>®</sup>-C:2004, as explained in the next section.

Another important feature of some static source code analysis tools is the capability to provide code metrics, which essentially is statistics about the source code. Code metrics can for example present the percentage of lines that contain C/C++ comments, or information about the complexity level of each C/C++ function in the project.

Overly complex functions should be rewritten into a simpler coding style to reduce risk of bugs and to simplify maintenance. It is important to understand that code complexity in a C function has nothing to do with how many lines of code it contains; a short function can be contain very complex code and large functions can be written in simple manner.

Many companies have started to realize the quality benefits of following a coding standard like MISRA<sup>®</sup>-C, and may also have company guidelines that mandate that source code files must have a certain percentage of comment lines, that functions may not surpass a certain complexity value, etc.

Static source code analysis should be integrated in the C/C++ IDE in order to simplify its daily routine use, as is the case with the **Atollic TrueINSPECTOR<sup>®</sup>** tool that integrates into the **Atollic TrueSTUDIO<sup>®</sup>** IDE.

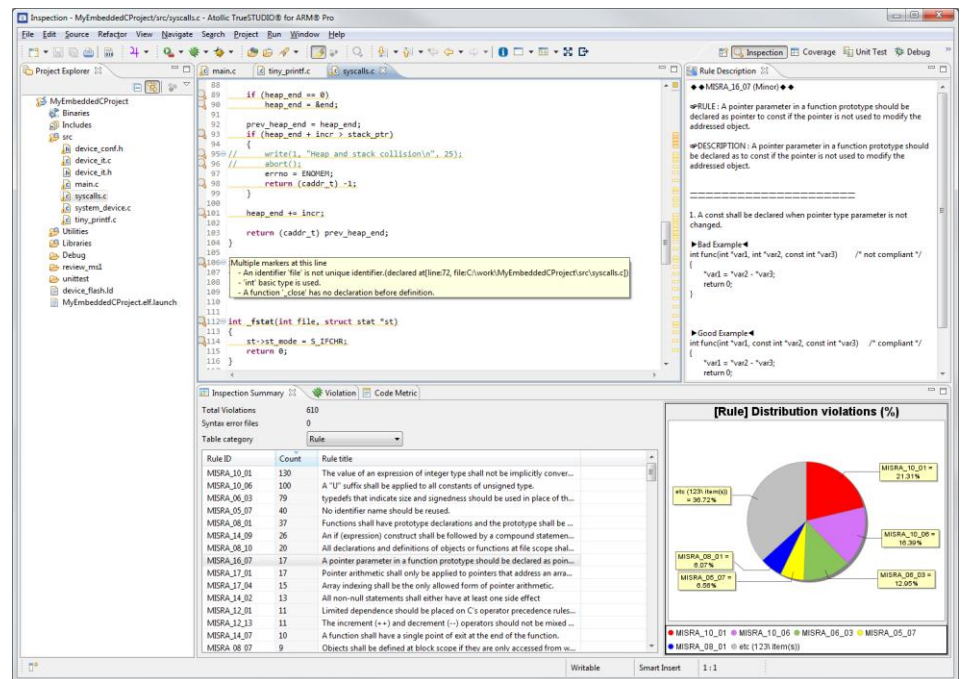


Figure 2 - Atollic TrueINSPECTOR® is integrated in the Atollic TrueSTUDIO® IDE

A common misconception is that a static source code analysis tool is only needed at the end of the project, where some violating code lines can be fixed at the end of the project.

In fact, nothing is further from the truth. No-one will start to rewrite code with thousands or perhaps tens of thousands of rule violations when the code is completed. The correct approach is to use this type of tool at least daily from the project start, to ensure a gradual and iterative development where all code additions are checked and fixed as they are added.

## CODING STANDARDS COMPLIANCE

While there are many coding standards in use, the most common one in the embedded industry is currently MISRA®-C.

MISRA® (The Motor Industry Software Reliability Association) was established as a collaboration between various vendors in the automotive industry. The purpose is to promote best practice in developing safety-critical systems in road vehicles and other types of embedded systems.

MISRA®-C is a coding standard for the C programming language, developed by MISRA®. The purpose is to identify a subset of the C language that improves safety, portability and reliability.

In 1998, the first edition of the MISRA® standard (MISRA®-C:1998, titled "Guidelines for the use of the C language in vehicle based software") was released. MISRA®-C:1998 have 127 rules, of which 93 are required and 34 are advisory.

The MISRA®-C:1998 standard was targeted towards automotive systems, and in 2004, a second edition (MISRA®-C:2004, titled "Guidelines for the use of the C language in critical systems") was released. MISRA®-C:2004 is more generic and better adapted for any type of embedded system, and have 141 rules of which 121 are required and 20 are advisory.

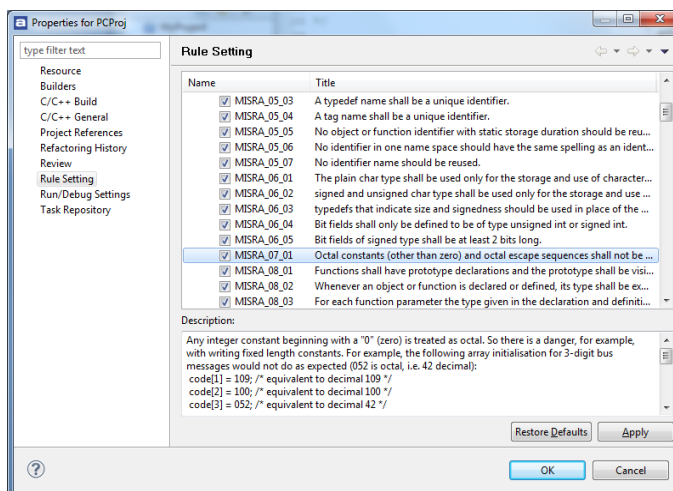


Figure 3 - Selecting MISRA®-C rules in Atollic TrueINSPECTOR®

By following the MISRA®-C coding standard, you ensure that potentially unsafe or unreliable coding constructs are not used in your software product, thus removing potential software problems and improving maintainability, and hence improving the overall software quality.

It is almost impossible to ensure MISRA®-C compliance without tool support. To ensure compliance to a coding standard like MISRA®-C, a tool that automates the process by performing static source code analysis is needed.

**Atollic TrueINSPECTOR®** is an embedded systems tool that performs MISRA®-C:2004 checking, automatically verifying source code compliance, and pointing out any code lines that breaks any of the coding standard rules.

## SOURCE CODE METRICS

In addition to the benefits achieved by complying with a coding standard that enforce best-practice and defensive coding, code metrics can be used to further reduce the likelihood of software problems and to improve maintainability.

Source code metrics is essentially statistics on how the source code is written. Various metrics can be gathered, such as information on number of files, number of lines in files etc.

In practice, two metrics parameters tend to be useful when improving the coding style:

- Comment ratio
- Cyclomatic value of code complexity

The comment ratio gives a measurement on how well the source code files are documented. For example, your company may require that 30% of the code lines should contain a comment to be considered to be well documented enough. Obviously, files with too few source code comments are more difficult and expensive to maintain.

The cyclomatic value of code complexity gives a measurement on how complicated the software implementation is. Overly complex functions should be rewritten into simpler coding style in order to prevent bugs and simplify maintenance.

The cyclomatic complexity theory was developed by Thomas J. McCabe in 1976 and measures the number of linearly independent paths through program source code.

The complexity level accepted by a project or company may be different dependent on product, industry or company belief. But it has been suggested that no function ought to have a complexity value above 10, and for safety-critical products this limit may be 5.

The main point is that code metrics provides you with statistics on commenting level and function complexity, thus enabling you to rewrite the code into better coding style. This typically leads to less bugs and simplified maintenance.

# EMBEDDED TOOLS FOR STATIC SOURCE CODE ANALYSIS

**Atollic TrueINSPECTOR®** is an embedded systems code analysis tool that performs static source code analysis. It performs MISRA®-C:2004 coding standard compliance checking and code metrics measurements.

**Atollic TrueINSPECTOR®** performs static source code inspection with one mouse click, and provides various views that visualize the analysis results. For example, overall statistics is presented in list and chart form with various search and filtering options.

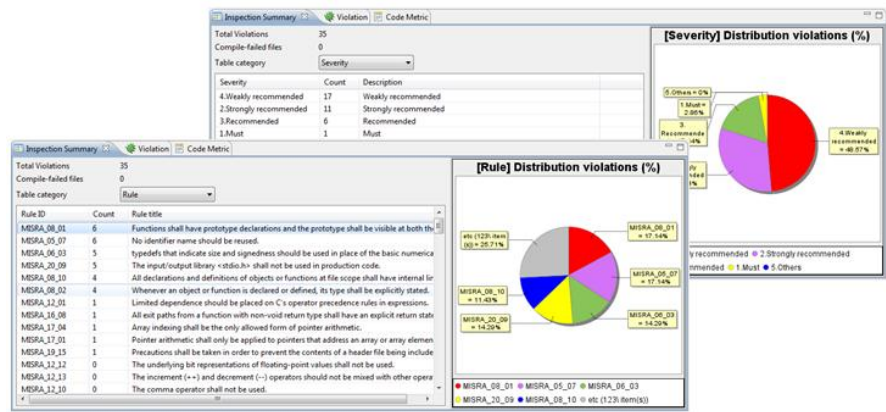


Figure 4 - Code analysis overview results in Atollic TrueINSPECTOR®

Developers might be more interested in the actual violations, and a view contains detailed information on specific violations with hyperlinks to the corresponding lines in the source code files. A rule description view provides detailed information on each coding rule, and display bad and good code examples as a teaching aid as well.

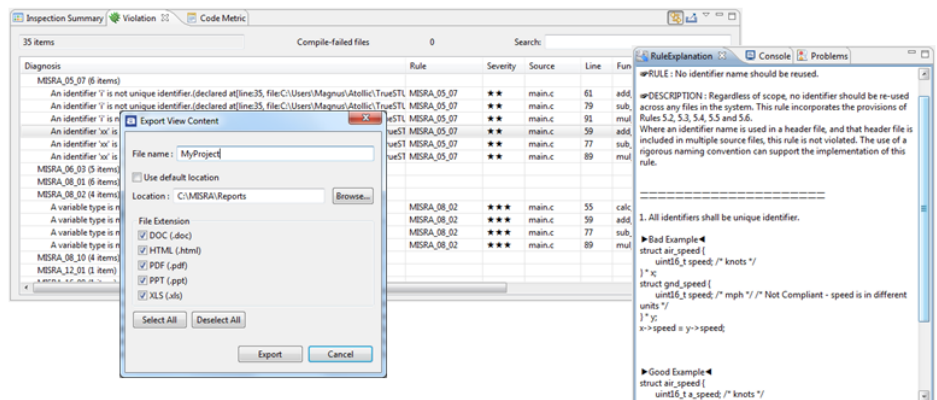
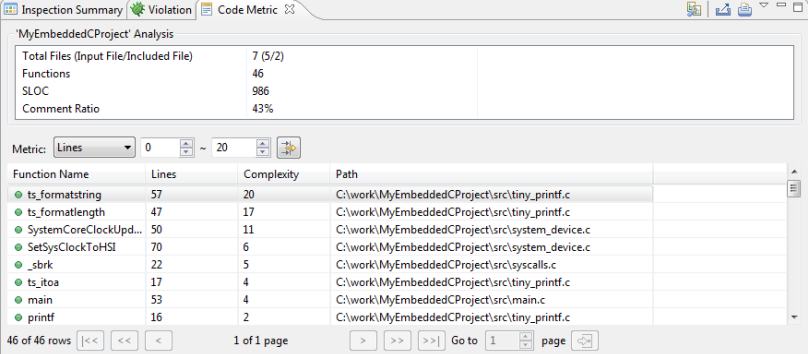


Figure 5 - Violations list and rule descriptions in Atollic TrueINSPECTOR®

**Atollic TrueINSPECTOR®** performs source code metrics on project, file and function level too. This enables developers to view statistics on parameters like number of lines in files, number of lines in functions, commenting level and function complexity.



The screenshot shows the 'Code Metric' window in Atollic TrueINSPECTOR. It displays a summary of metrics for the project 'MyEmbeddedCProject' and a table of function-level metrics. The summary includes: Total Files (Input File/Included File) 7 (5/2), Functions 46, SLOC 986, and Comment Ratio 43%. The function-level table lists functions with their names, line counts, complexity scores, and file paths. The table is sorted by complexity, with 'ts\_formatstring' having the highest complexity of 20.

Function Name	Lines	Complexity	Path
ts_formatstring	57	20	C:\work\MyEmbeddedCProject\src\tiny_printf.c
ts_formatlength	47	17	C:\work\MyEmbeddedCProject\src\tiny_printf.c
SystemCoreClockUpd...	50	11	C:\work\MyEmbeddedCProject\src\system_device.c
SetSysClockToHSI	70	6	C:\work\MyEmbeddedCProject\src\system_device.c
_sbrk	22	5	C:\work\MyEmbeddedCProject\src\syscalls.c
ts_ttoa	17	4	C:\work\MyEmbeddedCProject\src\tiny_printf.c
main	53	4	C:\work\MyEmbeddedCProject\src\main.c
printf	16	2	C:\work\MyEmbeddedCProject\src\tiny_printf.c

Figure 6 - Function level code metrics in Atollic TrueINSPECTOR®

Analysis results reports can be exported in PDF, HTML and Microsoft® Office® file formats too, for offline analysis or as technical proof of compliance for management or customers.

## SUMMARY

As complexity and code size grow for each year, so does the problem of releasing high-quality software. New embedded tools, like **Atollic TrueINSPECTOR®**, integrates seamlessly into the C/C++ development environment and provides automated code analysis that point out potential coding problems and enables improved software maintainability.

By checking for coding standards compliance, comment level and function complexity early and often, your product can be released with higher quality and with lower maintenance costs.

Atollic provides a family of well integrated tools for professional embedded systems development and debugging, static source code analysis, test automation and test quality measurement.

More information about Atollic, **Atollic TrueSTUDIO®**, **Atollic TrueINSPECTOR®**, **Atollic TrueANALYZER®** and **Atollic TrueVERIFIER™** products is available here:

[www.atollic.com](http://www.atollic.com)

[www.atollic.com/truestudio](http://www.atollic.com/truestudio)

[www.atollic.com/trueinspector](http://www.atollic.com/trueinspector)

[www.atollic.com/trueanalyzer](http://www.atollic.com/trueanalyzer)

[www.atollic.com/trueverifier](http://www.atollic.com/trueverifier)