Synopsis of

“COMPARATIVE STUDY OF MISRA-C COMPLIANCE CHECKING TOOLS”

Independent research by TERA-Labs

(a research division of the Karel de Grote University College, Antwerp)

On May 9th 2012 TERA-Labs (a research division of the Karel de Grote University College, Antwerp, Belgium) published the final version of their research report, “A Comparative Study of MISRA-C compliance Checking Tools” [Ref 1].

This white paper comprises of two sections:

- **Section 1** provides an overview of the full, detailed 80-page TERA-Labs research. This summary - which has been generated by PRQA - comprises direct extracts of key content from the TERA-Labs report, and the authors of the original TERA-Labs research have confirmed that this is a fair and accurate summary of the original report.

- **Section 2** contains some of PRQA’s own observations and comments relating to the TERA-Labs report.

Keywords: MISRA, Static Code Analysis, Coding Standard, Compliance
SECTION 1 : Report Summary
This section contains a summary of the original detailed 80-page TERA-Labs report [Ref 1].

1.1 Introduction
TERA-Labs is a new research division of the Karel de Grote University College located in Antwerp, Belgium. The focus of TERA-Labs’ research is mainly on embedded systems across a wide range of applications/industries from automotive to distributed computing. This specific independent research on MISRA-C compliancy checking software tools was funded by IWT (agentschap voor Innovatie door Wetenschap en Technologie, www.iwt.be) and conducted over a 20-month period. A preliminary version of the report was issued October 6th 2010, and the final version was published May 9th 2012.

1.2 Scope and Objectives
The goal of the research was to assess how well software tools enforce the MISRA-C:2004 coding rules. The methodology involved presenting each of the tools with a series of test cases (“probes”) written in the C programming language by TERA-Labs engineers. Each of these test cases contained deliberate specific MISRA-C:2004 coding rule violations, and the team evaluated the effectiveness of each tool in identifying these violations.

1.3 Participants
TERA-Labs selected the following static analysis / code analyzer tools for the research. These tools were identified via an internet search, and those selected explicitly mentioned enforcement of MISRA-C:

1. Development Assistant for C (DAC) - RistanCASE
2. IAR embedded workbench
3. Klocwork Insight
4. LDRA Testbed
5. Parasoft C++test
6. QA-C – Programming Research/PRQA
7. PC-Lint - Gimpel
8. (Prevent - Coverity) *
9. Raincode **

(* TERA-labs subsequently dropped Coverity from the research once it became apparent that their tool only covered a minimal subset of the MISRA-C rules. There is therefore no data for Coverity in the results section of the report. ** Raincode were subsequently added to the project and are included in the results.)

Note also, due to legal requirements, in the final TERA-Labs report the tool vendors are made anonymous and referenced as Company 01 to 09. There is no relation between the order of the tools in the above list and the order of the charts and columns in the tables. However, in this summary whitepaper we are able to confirm that PRQA (QA-C) is Company 04.

1.4 Study Criteria
“Soft” and “hard” assessment criteria were considered during the study, as follows:

**Soft criteria**:
1. Ease of integration in the existing development environment.
2. Usability: the ease with which people can employ a particular tool.
3. Extensibility: can the user extend the program easily with extra functionality?
4. Quality of violation messages.
5. Command-line functionality (for automatisation).
6. Extra functionality, e.g. metrics?

**Hard criteria:**
7. Correctness: are the violations reported by the tool real violations? (true positives).
8. Completeness: are all violations present in the code, reported by the tool (no false negatives)?

It was not practical to test all MISRA-C rules, and the research therefore focused on a subset of 11 “important” and “representative” rules which were selected via a panel of industrial partners.

The rules were classified into three groups:
1. Crashing: rules which can cause a crash if violated
2. Maintainability: these rules decrease the probability that a programmer introduces an error when changing code
3. Portability: improve portability

The following 11 rules in the crashing and maintainability groups were assessed during the research:

<table>
<thead>
<tr>
<th>MISRA-C Rule</th>
<th>Rule Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.3</td>
<td>The character sequence /* shall not be used in a comment.</td>
</tr>
<tr>
<td>8.12</td>
<td>When an array is declared with external linkage, its size shall be stated explicitly or defined implicitly by initialization.</td>
</tr>
<tr>
<td>9.1</td>
<td>All automatic variables shall have been assigned a value before being used.</td>
</tr>
<tr>
<td>11.1</td>
<td>Conversion shall not be performed between a pointer to a function and any type other than an integral type.</td>
</tr>
<tr>
<td>12.4</td>
<td>The right-hand operand of a shift operator shall lie between zero and one less than the width in bits of the underlying type of the left-hand operand.</td>
</tr>
<tr>
<td>14.7</td>
<td>A function shall have a single point of exit at the end of the function.</td>
</tr>
<tr>
<td>15.2</td>
<td>An unconditional <strong>break</strong> statement shall terminate every non-empty switch clause.</td>
</tr>
<tr>
<td>15.3</td>
<td>The final clause of a switch statement shall be the <strong>default</strong> clause.</td>
</tr>
<tr>
<td>16.6</td>
<td>The number of arguments passed to a function shall match the number of parameters.</td>
</tr>
<tr>
<td>17.6</td>
<td>The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist.</td>
</tr>
<tr>
<td>19.10</td>
<td>In the definition of a function-like macro each instance of a parameter shall be enclosed in parenthesis unless it is used as the operand of # or ##.</td>
</tr>
</tbody>
</table>
1.5 Results - Soft Criteria

The TERA-Labs report contains nine tables which have been summarized in the single table below:

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Comp 01</th>
<th>Comp 02</th>
<th>Comp 03</th>
<th>Comp 04 (PRQA)</th>
<th>Comp 05</th>
<th>Comp 06</th>
<th>Comp 08</th>
<th>Comp 09</th>
<th>MAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switching Rules From the Command Line</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>Switching Rules From the GUI</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>n/a</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Analyzing the Project and Excluding Files</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Quality of Error Messages &amp; Warnings</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>Integration</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>OS Supported</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Automation</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Additional Features</td>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Technical Support</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Total Score</td>
<td>15</td>
<td>18</td>
<td>22</td>
<td>25</td>
<td>16</td>
<td>20</td>
<td>21</td>
<td>18</td>
<td>30</td>
</tr>
</tbody>
</table>

(Note: The nine tables in the full report contain either ‘Yes’ or ‘No’ responses with several questions awarding ratings of up to three stars. In the condensed / simplified table above a ‘Yes’ has been scored as 1 and each star rating also scored as 1. The ‘MAX.’ column on the far right shows the maximum score available for each category.)

1.6 Results - Hard Criteria

The core of the TERA-Labs study shows the results of the hard criteria, and specifically the analysis of how effective each tool is in identifying violations of the 11 MIRSA-C rules. This effectiveness is measured in terms of each tool’s ability to 1) find and report all the genuine rule violations in the test code and 2) to avoid reporting false positives (noise), i.e. reporting violations where no genuine violation exists.

The results are presented in a series of charts in terms of “precision” and “recall”. The y-axis (recall) indicates how effective a tool is in finding genuine rule violations. The x-axis (precision) indicates how noisy a tool is and its propensity to produce false positives.
Tools which perform best will have datapoints plotted in the top right corner for this chart - indicating that the tool successfully finds the violation, with no associated noise / false positives. The further to the left that a datapoint appears the noisier the tool (more false positives) and the closer to the bottom indicates that the tool is less likely to find genuine violations. Bottom left depicts the worst performance, indicating that a tool fails to find the violation and simply creates noise / false positives.

The following charts, which are Appendix L to S in the TERA-Labs report, show the performance of each of the 8 tools (excluding Coverity):
1.7 TERA-Labs Study Conclusions

The remit of the TERA-Labs briefing for this research was to present the analysis and results but specifically not to explicitly recommend the “best” tool.

The 6 conclusions in the TERA-Labs report are summarized as follows:

1. The differences between the tools are large
   - GUIs, the command line, installation procedures, licencing schemes, …. are very different
   - There are also significant differences between tools concerning the rules to which violations are produced
2. The large number of rules in MISRA will (for any tool) result in an unmanageable number of violation messages.
3. The expectation was to encounter a large number of false positives. Contrary to this expectation the team found more false negatives, i.e. several tools fail to find obvious violations.
4. The more expensive tools, generally give better results i.e more rules have precision and recall in the upper right corner.
5. Most tools do not give the option to obtain fully annotated source (source code + violation messages + line numbers) as an exported file. This limits the possibilities for the integration of the tool in existing software development processes.
6. An important problem encountered with all tools (some more than others) is the fact that the tools give no results if the code cannot be parsed completely. This is mainly problematic for large existing code bases.
SECTION 2 : PRQA’s Observations and Comments
In this section of the document we outline some key observations and comments from PRQA’s perspective. These comments also reflect additional direct discussions with the authors of the TERA-Lab report.

General
We are strongly of the opinion that this study represents a genuine, credible, independent assessment of MISRA-C compliance checkers. (Having said that we also recognize that there are a few minor inaccuracies in the report and we might have minor queries on a few of the interpretations.)

The TERA-Labs team stated that, “On paper (according to the marketing material) all the selected tools appeared equivalent and claimed to provide comprehensive MIRSA-C compliance checking - but the reality was very different.”

It took considerable time and effort for the TERA-Labs team to perform their analysis and reach an objective conclusion - more than 1 man year. No commercial company could reasonably justify this level of investment when selecting a tool.

Test Cases
Of the 11 selected rules, we would agree that these are indeed a good subset - important rules and representative of the MISRA-C coding standard.

The coding violations in the test cases created by TERA-Labs are relatively simple and straightforward. They did not represent complex code, difficult logic or unusual edge cases. There can be little doubt that tools which perform marginally when presented with basic examples of coding violations will perform much worse when confronted with larger complicated code bases.

Results - Hard Criteria Graphs
We completely agree that the ability to find true violations whilst minimizing noise/false negatives are absolutely the key criteria in assessing the performance and effectiveness of a tool, and that the graphical representations generated by TERA-Labs provide an excellent summary for each tool.

We are delighted that the performance of QA-C was rated so highly. It was clearly the best of all the tools tested, identifying all 11 rule violations without producing any false positives / noise.

We would point out the following:

- The x-axis (precision) indicates how noisy a tool is and its tendency to produce false positives. Datapoints plotted towards the left of the chart indicate that the tool generates more false positives. The obvious impact of false positives is the extra time/cost required for developers to assess, identify and eliminate these. This may not be a major issue if a limited number of rules are being tested in a small sample of code. However, as the number of rules and code size increases the implications (extra cost and time) of using a noisy tool become very apparent. Furthermore, note the very real danger that too many false positives will ultimately destroy the credibility of a tool and result in poor adoption by the development teams.

- The y-axis (recall) indicates how effective a tool is in finding genuine rule violations. Datapoints plotted towards the bottom of the chart indicate that the tool is failing to find genuine rule violations. This issue of false negatives is potentially more serious than false positives. We would suggest that a tool with too many false negatives might not be considered “fit for purpose” in demonstrating compliance to the coding standard. We also note that one of the key conclusions in the TERA-Labs report was that they encountered many more false negatives than they had anticipated. PRQA finds important, material defects that other tools miss.
Extrapolation of Results
We concluded, and the TERA-Labs team agreed, that if the scope of the research had covered all (~133) statically enforceable MISRA-C rules we would anticipate a similar relative performance from each tool, and an extrapolation of the results for the representative sample of the 11 MISRA-C rules. Indeed we could also anticipate a similar relative performance for the tool vendors in relation to MISRA-C++ compliance.

ROI (Return on Investment)
We note that the primary focus of the TERA-Labs research was on the technical performance of the tools. The study did not assess the time/cost that developers require to find and fix violations using each tool. Clearly much more time/cost will be required to eliminate false positives on noisy tools. Note that additional time/cost is also needed to find the false negatives - and significantly - catching these false negatives also requires additional supplementary testing methodologies. Unfortunately a cost-benefit analysis or ROI for each tool was outside the scope of the TERA-Labs research.

Process
During our discussion with TERA-Labs we both noted that more benefit can be derived from MISRA and MISRA compliance tools once development teams have structured software development processes in place.

Coding Standards
It is significant to note that while many development teams are specifically driving their code to be fully compliant to MISRA, other teams adopt key elements of MISRA (or an alternative coding standard), as a means to help them to develop more robust code - for example to focus on material/important defects such as undefined behavior and to make code reviews more effective. Their objective is not to have “MISRA compliant code” but to use MISRA as a means to generate more robust code.

One very valid / common approach, is to adopt a subset of key MISRA rules (eg start with the 11 rules per the TERA-Labs report), and later expand the number of rules as the value of the coding standard is recognized and acceptance of the coding rules grows within a development team. (Using PRQA tools it is very easy to select any subset of MISRA rules, and to complement the rule-set with any required company bespoke rules.)

Whether a rule originates internally or from an external standard it is important to recognize the role of static analysis tools in providing a highly effective, automated means to test the compliance to each rule.

Building on Strong Foundations
The strong performance of our tools reflects the fact that PRQA’s relationship with MISRA stretches back some 20 years. Major elements of both the MISRA-C and MISRA-C++ guidelines have been derived from our own coding standards and our technical experts have been - and will continue to be - key members of the working groups, writing the MISRA standards (including the new MISRA-C3 standard). It is these same language experts who design and develop our static analysis tools.

In Conclusion
The TERA-Labs study provides a genuine, credible, independent assessment of MISRA-C compliance checkers. We are delighted that the performance of QA-C was rated so highly and was shown to be the best of all the tools tested - finding important, material defects that other tools miss, and generating very few (no) false positives. We would urge companies who are selecting these tools to be very conscious of the impact (cost, time, quality, ROI) of deploying tools which perform poorly, in terms of false negatives and false positives/noise.


Contact Us: Email: info@programmingresearch.com Web: www.programmingresearch.com

All products or brand names are trademarks or registered trademarks of their respective holders.