Static code analysis need not be limited to finding faults in manually produced code. When PRQA worked with long term customer Selex ES to integrate their QA·C++ source code analysis solutions with IBM Rational Rhapsody system modelling environment (auto-generated code), this improved code quality, reduced development times and resulted in a much more effective and robust software development workflow.

1. Manual Code

Selex ES, part of the Finmeccanica Group, has been using PRQA’s static analysis solutions since 2001, with the initial deployment of QA·C in Basildon, UK.

Four years later, Selex ES initiated a major re-evaluation of their complete tool chain, looking at best practices, identifying gaps and opportunities to improve productivity. The following key requirements were assessed during a rigorous evaluation of the commercially available code analysis tools:

- identify defects and dangerous language usage
- ensure coding standard compliance (to a MISRA based coding standard)
- ensure that the tool generates minimal false positives (“noise”)
- provide style enforcement (e.g. naming conventions and physical layout, to make code easier to use by other people)
- “phantom inspection” (ensuring that the tool gives developers visibility of other developers’ defects and accelerating the adoption of best practices)
- report metrics (providing management with analytics in support of code improvement)

Static Analysis - for Manual Code and Auto-generated Code

In Brief

- Aerospace and Defence Industry
- QA·C/QA·C++ widely adopted (since 2001) and proven to be highly effective when used to analyse manually generated code
- MISRA-based coding standard (sub-set of MISRA rules, with some company specific extensions)
- Rhapsody and auto-generated code identified as a critical, strategic initiative - and QA·C/ QA·C++ integration now providing an effective, pragmatic way to independently verify the quality of auto-generated code

www.programmingresearch.com
Selex ES also recognised the output needs of different stakeholders at different stages in the software development process, as set out in the table below.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Purpose</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design &amp; Implement</td>
<td>Provides an ongoing sanity check. Assists inexperienced engineers. Minimises defect leakage.</td>
<td>Developer</td>
</tr>
<tr>
<td>Test</td>
<td>Provides an automated inspection of generated code prior to dynamic testing</td>
<td>Tester</td>
</tr>
<tr>
<td>Approval</td>
<td>Required for work product / SCR (Software Change Request) closure</td>
<td>Design Authority</td>
</tr>
<tr>
<td>Release</td>
<td>Required for deliverable baseline</td>
<td>QA</td>
</tr>
</tbody>
</table>

And the outcome of this evaluation? In 2007 the decision was made to adopt QA·C/QA·C++ company-wide and these tools were subsequently rolled out across Edinburgh, Luton and to other Selex ES sites in Italy. Ian Anderson, Head of Software Engineering for Sensors Data Processing Solutions, confirmed, “This was an easy decision, QA·C and QA·C++ were already widely adopted by our development teams and already proven to deliver a highly effective, automated analysis of manually generated code.”

Over the following years the emphasis of Selex ES’s peer reviews has changed significantly, removing the need for detailed code walk-through, and allowing reviewers to concentrate on whether the code will work as designed. The results are much more consistent and efficient than manual code reviews that rely heavily on the knowledge and ability of the reviewers. Selex ES also observed that dynamic testing is now faster and easier, as QA·C/QA·C++ has improved the quality of the code and has already removed many defects before the unit testing stage is reached, thus minimizing the need for iterative rework. Additionally, the code is now stylistically consistent and much easier to maintain.

2. Model-Driven Development
One of Selex ES’s key strategic initiatives has been to increase the adoption of Model Driven Design (MDD), based on IBM Rational Rhapsody. As well as accelerating the overall development process, one of the key objectives is to leverage the re-use of auto-generated code across different projects.

Selex ES has found that auto-generated code from the UML model is between 60-80% of the code base on a typical project. The remaining 20-40% still needs to be created manually as code inserts (normally using C++), and these two types of code are interwoven within the one common codebase, generated from a model that incorporates both.

Diagrams 1a and 1b illustrate the relationship between the code in the model and the generated code:
Rhapsody model-generated code has widespread industry adoption, and this, coupled with its use over 15 years on around 40 live projects, gives Selex ES a high level of confidence in the robustness of this code.

Initially, manual code inspections were needed to provide independent verification, but this created major problems in terms of scalability, speed, consistency of analysis, and overall resourcing. QA/C/QA·C++ were already providing Selex ES with a highly effective verification solution for manually generated code. The challenge was to find a way to leverage this more fully in the Rhapsody environment. The crux of the problem was the need to differentiate between the two types of code:

a) Auto-generated code derived from the UML model artefacts:
For this type of code, Selex ES’s approach is “object orientated”, the focus of testing is more on functionality and is predominantly requirements-based or design-based with coverage measurements. Supporting this approach are three considerations. Model-generated code is more tightly constrained in use of language, due to inherent limitations in the model conversion and code creation process. Also, analogous to legacy considerations, generated code has already been field-proven through past usage, thus providing confidence-through-use verification. Finally, model-generated code cannot be altered by developers, only the model itself can be changed.

b) Auto-generated code derived from manual code inserts and model artefacts controllable by developers:
Code inserts are essentially manually created code, which Rhapsody interleaves with the code generated from model artefacts. Model artefacts controllable by developers are aspects such as function prototypes where control is available via the Rhapsody GUI. Both of these should be fully examined like any other manually generated code, to ensure compliance with best practices and coding standards. Since the developer is responsible for modifying this code, it is crucial that code analysis fits seamlessly within the software development workflow.

Rhapsody and QA/C/QA·C++
A joint PRQA and Selex ES workshop identified a number of key elements that were essential for a successful integration between Rhapsody and QA/C/QA·C++, in brief:

- Enabling developers to identify and focus their attention on the code that is within their responsibility - ie the manually generated code and model artefacts that are code-like (e.g. function parameter types). This means that it must be possible to identify and suppress any messages specific to the boilerplate auto-generated code derived from model artefacts.

- Enabling QA/C/QA·C++ analysis to fit seamlessly within the Rhapsody-based software development workflow, specifically:
  - operating with the full QA/C/QA·C++ functionality
  - easy navigation to static analysis diagnostics from within the Rhapsody IDE
  - providing efficient focus and analysis on each individual code change

The outcome of this exercise was the creation of a QA/C/QA·C++ integration for Rhapsody, which was subsequently rolled out across all Selex ES Rhapsody projects.

Working with the integration - at a more technical level
Selex ES uses Microsoft’s MSVC++ compiler and Wind River’s VxWorks RTOS, with GNU and DIAB compilers. The integration runs happily with all of these applications. Selex ES use Rhapsody as the ‘IDE’ and so code inserts are directly entered into Rhapsody and stored/configured as part of the model files: there are no source files containing handwritten code. Engineers only work with the model – the generated C++ files are merely artefacts on the way to an executable, much like object code. Selex ES are model orientated, this is the only item that is required to be stored in the repository. All the required code can be generated from the model.

After creating the model in Rhapsody, the integration, run from within Rhapsody, creates a QA+C++ project containing the generated C++ files and required analysis settings. The QA+C++ GUI automatically launches and loads the project ready for analysis. The developer can then choose which files to analyse. Analysis produces diagnostics describing the issues found in the code.
Within QA-C++ are over 1300 possible diagnostics, grouped by MISRA rules (or, if not using MISRA, ranked into 9 levels of importance).

After analysis, the messages are displayed in a message browser. This groups the messages according to severity, making it easy to differentiate the most critical issues (e.g. undefined behaviour) from the less important ones (e.g. code layout). Navigation is possible by file or by message or group. Diagnostics are interleaved with the generated source and every message generated includes explanatory help in HTML format.

Diagnostics associated with model artefacts are suppressed (hidden) by default, but can also optionally be made visible. On review of all other diagnostics, developers can then modify their code inserts within the Rhapsody IDE to resolve issues. Re-generation through Rhapsody and re-analysis through QA-C++ gives very fast confirmation that the model modification has fixed the issue and also not introduced other issues.

In conclusion

Selex ES have been successfully using QA-C/QA-C++ to automatically analyse manually generated code for more than a decade.

With the strategic adoption of Model Based Design, they were extremely keen to ensure that the proven benefits of QA-C/QA-C++ could continue to be leveraged in the Rhapsody environment. The solution to this challenge recognized the fact that two distinct categories of auto-generated code need to be accommodated - code derived from manual inserts and code derived from the UML model artefacts. PRQA provided Selex ES with an integration which:

- enables developers to identify and focus on the manually generated code inserts that they own and can be edited directly
- enables QA-C/QA-C++ analysis to fit seamlessly within the Rhapsody based software development workflow

“The rollout of Rhapsody and auto-generated code was a critical, strategic initiative for Selex ES” confirmed Ian Anderson, “QA-C and QA-C++ were already widely adopted by our development teams and already proven to be highly effective when used to analyse manually generated code. This integration was very important for us, and PRQA delivered an effective, pragmatic way to independently verify the quality of auto-generated code.”

Selex ES, a Finmeccanica company, is an international leader in electronic and information solutions for defence, aerospace, space, security, high-integrity surveillance, network management, information security and mission-essential services.

As a world leader in high technology systems and sensors with extensive experience across a range of sectors and domains, Selex ES is able to meet the diverse needs of customers who require first class solutions.

Within aerospace and defence electronics the company’s experience includes the design and development of tactical ISTAR systems, C4I infrastructures, electronic warfare equipment and mission critical systems for situational awareness, self-protection, wide-area surveillance and information dissemination.

Selex ES brings similar technologies and skill sets to the security, route management and mission-critical service sectors. These include the control and monitoring of air and maritime traffic, the surveillance and protection of green and blue borders, robust cyber security, secure communication networks, and the deployment of ‘smart’ solutions for managing complex infrastructures and ‘systems of systems’.

Alongside core domestic operations and corporate headquarters in Italy and the UK, Selex ES has an established industrial and commercial footprint in the United States, Germany, Turkey, Romania, Brazil, Saudi Arabia, India and the United Arab Emirates. With a workforce of over 17,000 people and total revenues in excess of €3.5 billion, Selex ES is entrusted by its customers and partners to deliver solutions for a safer, smarter and more secure society. www.selex-es.com

Contact Us

For further information regarding PRQA’s solutions, please contact your local sales representative, or directly at:
info@programmingresearch.com

© Programming Research Ltd 2014 v1.2  www.programmingresearch.com