Automating the maintenance of bi-directional requirements traceability

Introduction

Although the ever improving techniques in safety- and mission-critical software development and test are proven to yield significant improvements in software quality, they come to naught if the resulting application fails to perform as expected by the stakeholders – not just functionally but also with adequate regard for safety.

Small wonder then that depending on the criticality of the application, requirements traceability is obligatory for certifiable, safety-critical applications to ensure that all requirements are implemented, and that all development artefacts can be traced back to one or more requirements.

When requirements are managed well, traceability can be established between each development phase. For example, the resulting bi-directional traceability demonstrates that all system level requirements have been completely addressed by high level requirements, and that all high level requirements can be traced to a valid system level requirement. . Requirements traceability also encompasses the relationships between entities such as intermediate and final work products, changes in design documentation, and test plans.

The principle of bi-directional traceability has been established in the avionics community since no later than 1992 when the DO-178B document[1] was introduced (since succeeded by DO-178C[2]), and the introduction of other functional safety standards such as ISO 26262[3] in the automotive industry, IEC 62304[4] in the medical device sector, and the more generic IEC 61508[5] has seen that principle embraced more widely.

Although it is both a logical and laudable principle, last minute changes of requirements or code made to correct problems identified during test put such ideals into disarray. Despite good intentions, many projects fall into a pattern of disjointed software development in which requirements, design, implementation, and testing artefacts are produced from isolated phases. Such isolation results in tenuous links between requirements, the development stages, and/or the development teams.

The answer to this conundrum lies in the “trace data” between development processes which sits at the heart of any project. Whether or not the links are physically recorded and managed, they still exist. For example, a developer creates a link simply by reading a design specification and using that to drive the implementation. The collective relationships between these processes and their associated data artefacts can be viewed as a Requirements Traceability Matrix, or RTM. When the RTM becomes the centre of the development process, it impacts on all stages of safety critical application development from high-level requirements through to target-based testing.

Beyond that, the nature of connectivity calls into question when the development process itself comes to an end. The advent of such as the connected car, interactive medical device and Industrial IoT means that requirements can change at

[1] RTCA DO-178B “Software Considerations in Airborne Systems and Equipment Certification” http://www.rtca.org
[2] RTCA DO-178C “Software Considerations in Airborne Systems and Equipment Certification” http://www.rtca.org
[3] International standard ISO 26262 Road vehicles — Functional safety
[4] IEC 62304 International Standard Medical device software – Software life cycle processes Consolidated Version Edition 1.1 2015-06
[5] IEC 61508-1:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems - Part 1: General requirements

any time – not just during the traditional development lifecycle, but after it has been completed and even after a product's production life is over. Any newly discovered vulnerability or actual compromise of a system implies an additional requirement to counter it, bringing with it a new emphasis on traceability even into the product maintenance phase.

Requirements are an ongoing commitment

How often is the requirements specification baselined and then never referred to again? Without constant reference, how can a development team be sure that it is delivering a system which meets those requirements? By constructing trace links between requirements and development components from the very beginning of a project, problems such as missing or non-required functionality will be discovered earlier and, thus, will be easier and less costly to remedy. Unfortunately there are many factors which make it difficult or tiresome to maintain reference to a project's set of requirements.

At the start of a contract, the stakeholder sets out their vision for what they want from the delivered application. The project team then works to represent that vision as a set of requirements from which development can begin. The requirements should act as a blueprint for development. However, all too often, the team's efforts diverge from this blueprint resulting in an application which does not align with the requirements. At best the stakeholder is disappointed. At worst, the company opens itself up to litigation and costly remedial work.

The key to preventing the emergence of this "requirements gap" is to place the requirements at the forefront of development. To achieve this successfully, the process should not be too intrusive and the aim should be for it to help all participants equally, avoiding bias towards any particular disciplines or development phases.

As a basis for all validation and verification tasks, all high quality software must start with a definition of requirements. Each high level software requirement must map to a lower level requirement, design and implementation. The objective is to ensure that the complete system has been implemented as defined - a fundamental element of sound software engineering practice.

Simply ensuring that high level requirements map to something tangible in the requirements decomposition tree, design and implementation is not enough. The complete set of system requirements comes from multiple sources, including high level requirements, low level requirements and derived requirements. As illustrated in Figure 1 below, there is seldom a 1:1 mapping from high level requirements to source code, so a traceability mechanism is required to map and record the dependency relationships of requirements throughout the requirements decomposition tree.

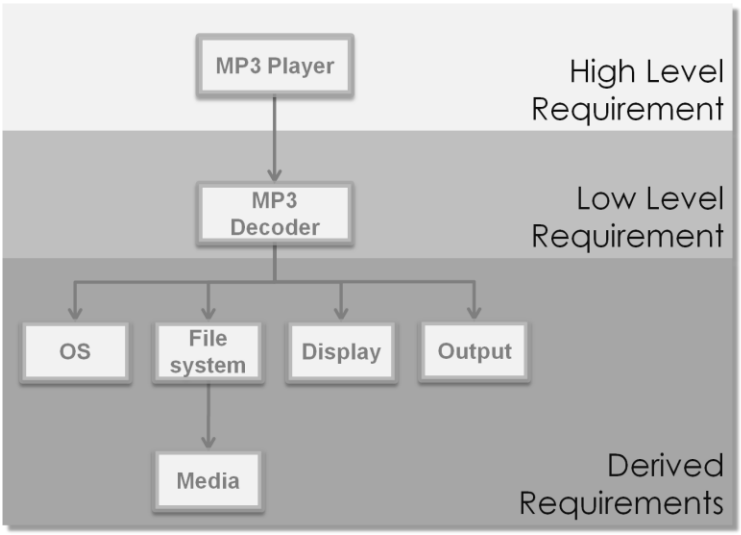

Figure 1 - Example of "1:Many" mapping from high level requirement through to requirements decomposition tree

Automating the maintenance of bi-directional traceability

To complicate matters further, each level of requirements might be captured using a different mechanism. For instance, a formal requirements capture tool might be used for the high level requirements while the low level requirements are captured in PDF and the derived requirements captured in a spreadsheet.

The Waterfall Process and other stories

With the initial requirements specified, development can proceed in accordance with the specified process for the project. It is useful to consider what impact the requirements have on the chosen process, and vice versa.
Back in the 80s and 90s the "Waterfall" process dominated software development. Waterfall development processes are generally broken down into four, distinct phases, as illustrated in Figure 2.

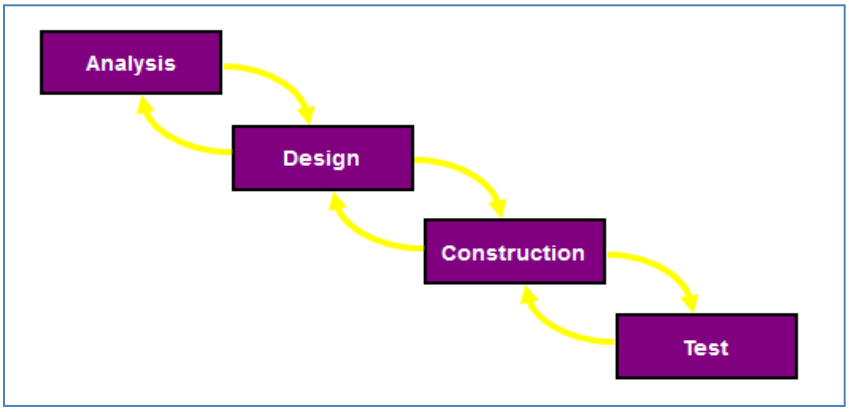

Figure 2 - The real- life implementation of a project is rarely as simple as the "Waterfall" process suggests

Each phase is performed in isolation with the output of one being the input for the next. The final output, in theory anyway, is a working system which passes all tests.

The purpose of the analysis phase is to refine the stakeholder's vision of the system and produce a list of requirements, with those for software being itemised in the Software Requirements Specification (SRS). However much a project manager may wish for the SRS to be error-free, it rarely is and the change log begins increasing in size until a new version becomes inevitable.

Contemporary software development processes and practices, such as the Iterative process, address many of the deficiencies found in the Waterfall process.

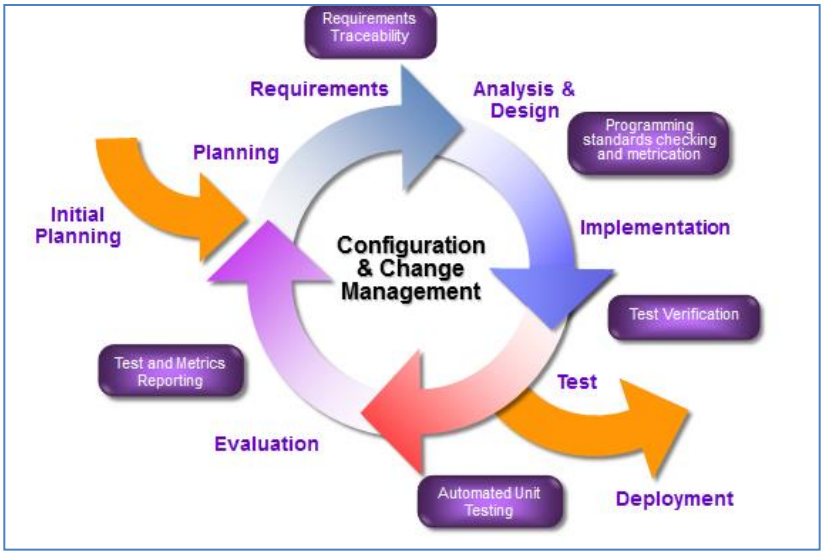

Figure 3 - The "Iterative" process is just one example of how the "Waterfall" process has been refined to reflect the dynamic nature of projects and their requirements

Requirements will change during the life of a project, whether due to stakeholders altering their vision or requested features proving to be unfeasible. Iterative processes (Figure 3) embrace this fact by splitting development into a number of phases (iterations) and considering only a subset of requirements during each iteration. Thus, the number of requirements subject to revision through feedback is significantly reduced; meanwhile development and refinement of those requirements not yet marked for implementation may proceed and benefit from any quality improvements applied to those requirements being implemented.

Iterative processes retain the Waterfall phases, perhaps with altered names and additional disciplines added. With reference again to Figure 2 and Figure 3, the 'Requirements' discipline is analogous to the 'Analysis' phase. However, the key difference is that, although most effort is invested during early iterations as we would expect, effort continues over the life of the project (albeit at a gradually reducing level).

Iterations can be thought of as mini Waterfall projects. A subset of the envisioned system is selected for implementation and then taken through the phases of analysis, design, construction and test. At the end of the iteration, the subset is expanded to include additional features and a new mini Waterfall begins. This process ensures that the requirements are continually being revisited and refined, keeping them in focus and using them to drive development.

The Art of Requirements

Requirements need to be high quality. If they are too complex or cannot be easily understood, in later phases they will be difficult to follow and a lot of time will be wasted requesting modifications and refinements. Confidence in the requirements needs to be kept high otherwise the willingness to work with them will diminish, risking divergence from the stakeholder's vision of the application. If the starting point of a project is of poor quality, then low-quality software is sure to follow.

Given the overwhelming complexity of many projects, if all stakeholders are to share a common commitment to requirements then they must be understandable, unambiguous and precise. Such an environment will help alleviate scope creep & requirements churn, and will ensure that the delivered solution meets the stakeholder needs. It will also provide a mechanism to ensure adherence to any applicable software and industry standards.

Textual Specifications

Textual specifications remain a popular way to capture requirements, and although they can be highly effective there are some disadvantages. For example, the stakeholder may prefer layman's language whereas the contractor naturally leans towards technical jargon and their plans for the implementation. Furthermore, in conversational form, spoken language is inherently imprecise and prone to ambiguity.

Automating the maintenance of bi-directional traceability

However, if a high degree of rigour is applied then such pitfalls can be overcome. One approach is to apply rules when writing requirements in much the same way as the MISRA standards are applied to C and C++ code; for example:

- Use paragraph formatting to distinguish requirements from non-requirement text
- List only one requirement per paragraph
- Use the verb “shall”
- Avoid “and” in a requirement
    - Consider refactoring as multiple requirements or specifying in more general terms
- Avoid conditional language such as “unless” or “only if”
    - Such terms are likely to lead to ambiguous interpretation

The use of such key words also helps if some members of the development team are less fluent in the chosen requirements language than others.

Use Cases

Use Cases[6] or User Stories offer another way to organise requirements, and to reduce ambiguous or imprecise specification. The example in figure 4 clearly shows what is expected to happen under a particular set of circumstances. The reduced dependence on natural language is particularly beneficial to international companies that do not share a common spoken language. Graphical representation of requirements switches the angle of analysis from a line-by-line, itemised list of desired features (perhaps spreading over many pages) to a user-focused view of how the system will interact with external elements and what value it will deliver.

On the other hand, a disadvantage to this approach is that precise written language is sure to be universally understood by anyone fluent in it. Not everyone involved in the project, particularly at its periphery, will have the inclination to learn the nuances of Use Case diagrams.

Each Use Case or User Story comprises several scenarios. The first scenario as illustrated in Figure 4 is always the “basic path” or “sunny day scenario” in which the actor and system interact in a normal, error-free way.

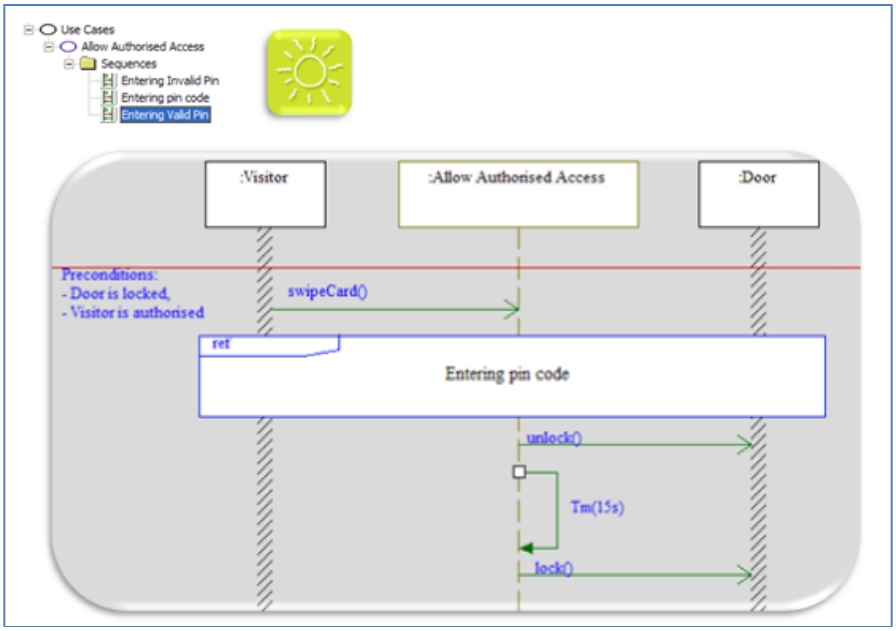

Figure 4 - This example of a “Sunny day” scenario from an “Allow Authorised Access” Use Case shows how a system is expected to behave when a valid key card is swiped

[6] TechTarget Definition: use case http://searchsoftwarequality.techtarget.com/definition/use-case

As the list of scenarios is established via this end-to-end analysis, the stakeholder's vision is rigorously exercised allowing ambiguities and problems to be ironed out. Each scenario is assigned a priority enabling the complete set to be ranked, allowing the project team to plan each iteration and select which subset of the system will be implemented.

Requirements Management and Traceability

However well the requirements are specified and their place in the development process established, a mechanism is required to ensure that they are reflected in the implementation of the project. Requirements traceability is widely accepted as a development best practice to ensure that all requirements are implemented and that all development artefacts can be traced back to one or more requirements. Like IEC 61508, ISO 26262 and IEC 62304 amongst others, the DO-178C standard requires bi-directional traceability and has a constant emphasis on the need for the derivation of one development tier from the one above it. Paragraph 5.5 c typifies this when it states:

"Trace Data, showing the bi-directional association between low-level requirements and Source Code, is developed. The purpose of this Trace Data is to:

1. Enable verification that no Source Code implements an undocumented function.
2. Enable verification of the complete implementation of the low-level requirements."

The level of traceability required by standards such as this vary, dependent on the criticality of the application. For example, less critical avionics applications designated DO-178C level (or "DAL") D are known as "black box", meaning that there is no focus on how the software has been developed. That means there is no need to have any traceability to the source code or software architecture. It is only required that the System Software requirements are traced to the High-Level Requirements and then to the test cases, test procedures and test results.

For the more demanding DO-178C levels B and C, the source code has been development process is considered significant and so evidence of bi-directional traceability is required from the High Level requirements to the Low Level Requirements and then to the source code.

Ultimately, for level A projects, there is a need to trace beyond the source code down to the executable object code. While bi-directional traceability is and always has been a laudable principle, last minute changes of requirements or code made to correct problems identified during test tend to put such ideals in disarray. Despite good intentions, many projects fall into a pattern of disjointed software development in which requirements, design, implementation, and testing artefacts are produced from isolated development phases. Such isolation results in tenuous links between the requirements stage and / or the development team.

Processes like the waterfall and iterative examples show each phase flowing into the next, perhaps with feedback to earlier phases. Traceability is assumed to be part of the relationships between phases; however, the mechanism by which trace links are recorded is seldom stated. The reality is that, while each individual phase may be conducted efficiently thanks to investment in up-to-date tool technology, these tools seldom contribute automatically to any traceability between the development tiers. As a result, the links between them become increasingly poorly maintained over the duration of projects.

The Requirements Traceability Matrix (RTM) provides the solution to this problem and represents the logical extension of the required traceability between different phases. The links between phases can be ignored, or they can be acknowledged and properly managed. Either way, are critical.

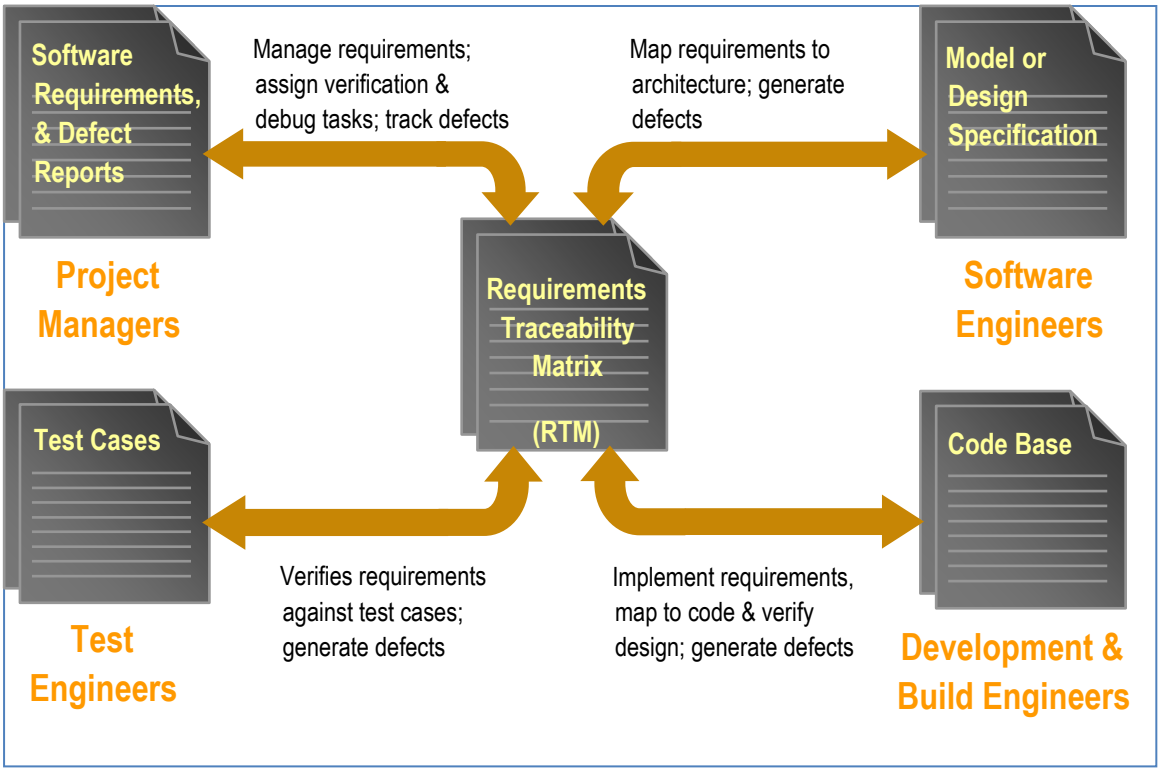

Figure 5 - RTM sits at the heart of the project defining and describing the interaction between the design, code, test and verification stages of development.

Figure 5 illustrates this alternative view of the development landscape, reflecting the importance that should be attached to the RTM. Due to this fundamental centrality, it is vital that project managers place the same priority on RTM construction and maintenance as they do on requirements management, version control, change management, modelling and testing.

The RTM must be represented explicitly in any lifecycle model to emphasise its importance as illustrated in Figure 6. With this elevated focus, it becomes the centre of the development process, impacting on all stages of design from high-level requirements through to target-based deployment.

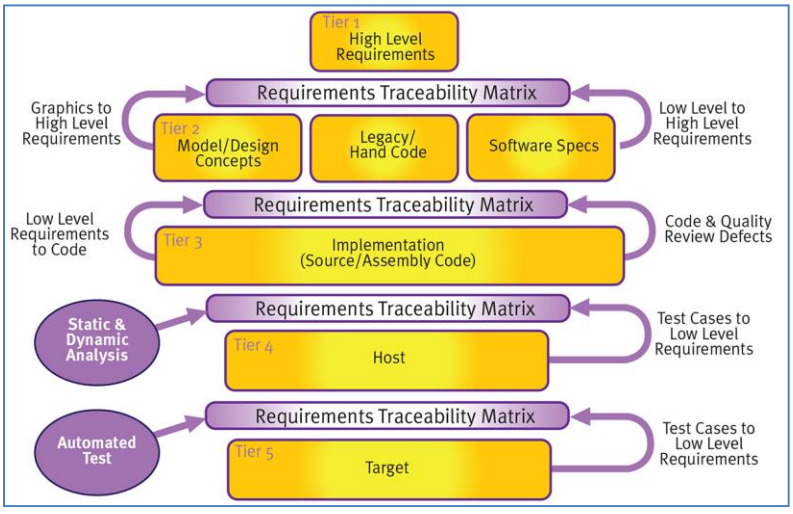

Figure 6 -The requirements traceability matrix (RTM) plays a central role in a development lifecycle model. Artefacts at all stages of development are linked directly to requirements matrix and changes within each phase automatically update the RTM.

Automating the maintenance of bi-directional traceability

At the highest level, Requirements Management and Traceability tools can initially provide the ability to capture the requirements specified by standards such as the DO-178C standard. These requirements (or “objectives”) can then be traced to Tier 1 – the application specific software and system requirements.

These Tier 1 high-level requirements might consist of a definitive statement of the system to be developed (perhaps a aircraft flap control module, for instance) and the functional criteria it must meet (e.g., extending the flap to raise the lift coefficient). This tier may be subdivided depending on the scale and complexity of the system.

Tier 2 describes the design of the system level defined by Tier 1. With our flap example, the low-level requirements might discuss how the flap extension is varied, building on the need to do so established in Tier 1.

Tier 3’s implementation refers to the source/assembly code developed in accordance with Tier 2. In our example, it is clear that the management of the flap extension is likely to involve several functions. Traceability of those functions back to Tier 2 requirements includes many-to-few relationships. It is very easy to overlook one or more of these relationships in a manually managed matrix.

In Tier 4 host-based verification, formal verification begins. Using a test strategy that may be top-down, bottom-up or a combination of both, software stimulation techniques help create an automated test harnesses and test case generators as necessary. Test cases should be repeatable at Tier 5 if required.

At this stage, we confirm that the example software managing the flap position is functioning as intended within its development environment, even though there is no guarantee it will work when in its target environment. DO-178C acknowledges this and calls for the testing “to verify correct operation of the software in the target computer environment”. However, testing in the host environment first allows the target test (which is often more time consuming) to merely confirm that the tests remain sound in the target environment.

In our example, we ensure in the host environment that function calls to the software associated with the flap control system return the values required of them in accordance with the requirements they are fulfilling. That information is then updated in the RTM.

Our flap control system is now retested in the target environment, ensuring that the tests results are consistent with those performed on the host. A further RTM layer shows that the tests have been confirmed.

Maintaining the Requirements Traceability Matrix

A Requirements Traceability Matrix is a laudable aim irrespective of whether a standard insists on it. However, maintaining an RTM in a set of spreadsheets is a logistical nightmare, fraught with the risk of error and permanently lagging the actual project status.

Constructing the RTM in a suitable tool not only maintains it automatically, but also opens up possibilities for filtering, quality checks, progress monitoring and metrics generation (Figure 7). The RTM is no longer a tedious, time-consuming task reluctantly carried out at the end of a project; instead it is a powerful utility which can contribute to its efficient running. The requirements becoming usable artefacts that are able to drive implementation and testing. Furthermore, many of the trace links may be captured simply by doing the day-to-day work of development, accelerating RTM construction and improving the quality of its contents.

Modern requirements traceability solutions enable the extension of the requirements mapping down to the verification tasks associated with the source code. The screenshot below shows one such example of this. Using this type of requirements traceability tool, the 100% requirements coverage metric objective can be clearly measured, no matter how many layers of requirements, design and implementation decomposition are used. This makes monitoring system completion progress an extremely straightforward activity.

Automating the maintenance of bi-directional traceability

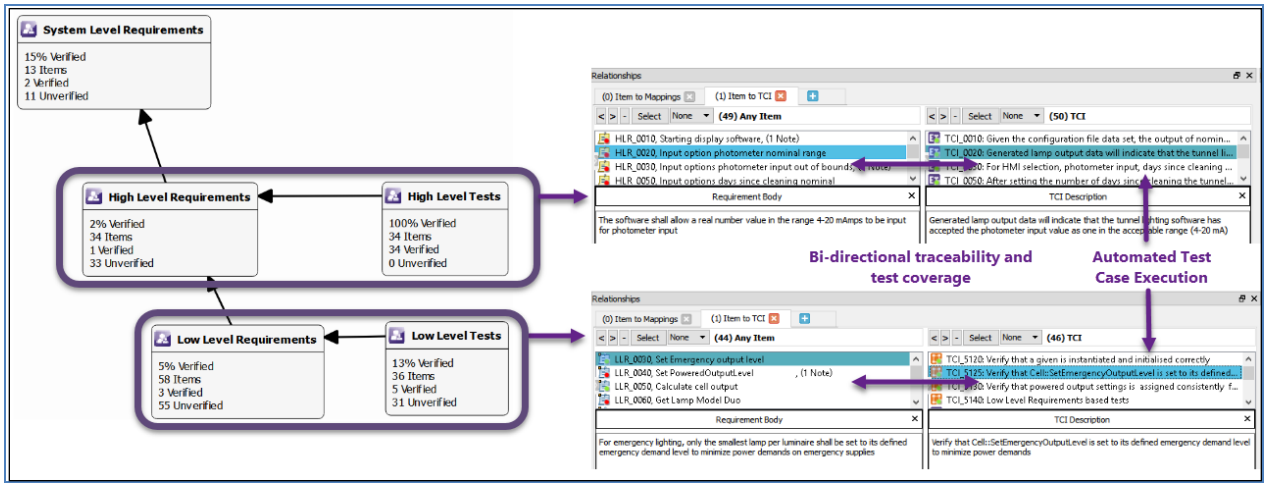

Figure 7 - Traceability from high level requirements down to source code and verification tasks.

Connectivity and the Infinite Development Lifecycle

During the development of a traditional, isolated system, that is clearly useful enough. But connectivity demands the ability to respond to vulnerabilities identified in the field. Each newly discovered vulnerability implies a changed or new requirement, and one to which an immediate response is needed – even though the system itself may not have been touched by development engineers for quite some time. In such circumstances, being able to isolate what is needed and automatically test only the functions implemented becomes something much more significant.

Connectivity changes the notion of the development process ending when a product is launched, or even when its production is ended. Whenever a new vulnerability is discovered in the field, there is a resulting change of requirement to cater for it, coupled with the additional pressure of knowing that in such circumstances, a speedy response to requirements change has the potential to both save lives and enhance reputations. Such an obligation shines a whole new light on automated requirements traceability.

Conclusion

The delivery of a Requirements Traceability Matrix (RTM) is often contractually imposed on suppliers. Even when not required, many development teams recognise that a RTM is an important ‘best practice’ for successful projects. However the creation of a useful and error-free RTM can only happen when the requirements are of sufficient quality and the process is taken seriously. This paper has outlined several areas which have the capability to limit or undermine the RTM and has proposed a series of solutions:

- Ensure that requirements embrace functional, safety and security related issues
- Accept that requirements will change over the life of the project
- Employ a development process which embraces and responds to change
- Manage the quality of requirements
- Let the requirements drive development
- Build a RTM from the start of the project
- Use the RTM to manage progress and improve project quality
- Use the RTM to respond quickly and effectively to newly-discovered security vulnerabilities after product deployment

Implementing these improvements undoubtedly takes effort but the end result will be a project that finishes on time, on budget, avoids any gap between the stakeholder’s vision of the application and what is ultimately delivered, and results in an effective support vehicle for deployed connected systems.

Speaker

Company Details

LDRA
Portside
Monks Ferry
Wirral
CH41 5LH

Tel: 0151 649 9300
Fax: 0151 649 9666
E-mail: info@ldra.com

Contact Details