Series of articles Functional Safety, Part 3

FUNCTIONALE SAFETY

Product lifecycle management – but how!?

Toolchain – from requirement to test case; traceable, documented and verifiable

Motivation

In the past, software was merely an accessory that controlled the few functions of an actuator or sensor and prepared essential communication. In the age of Ethernet-based real-time protocols, more powerful processors and growing requirements, SMEs have started facing these challenges head-on with the help of software developers. However, having the tools is not enough. If there are no defined software development processes that are adhered to, software releases often prove hard to plan and errors pile up. Changes would often be implemented on demand and would only be understandable to the developers themselves.

Part 1 of this article detailed individual standards and the relationships between them, outlined the safety lifecycle, and explored the tension between flexibility, agility, and the formal requirements stipulated in the standards.

In Part 2, we looked at the early stages of projects and examined in more detail the emergence of the SRS (Safety Requirement Specification).

Part 3 discusses software development processes, product lifecycle management, and the documentation and development environment in which traceable and verifiable requirements should, in accordance with IEC 61508, result in a certifiable product. But what does this type of environment contain? Which tools are used? And how are these tools integrated?

Software development processes

Certain process models exist, such as the V-model, that are divided into phases. This is the model of choice if exact specifications are to be observed and verified. It also represents the standard model for functionally safe developments, external audits and certifications and public clients. It is usually combined with iterative models. Planned processes such as these are characterised by the fact that all activities are planned in advance and the progress of the project is measured against this plan.

Agile (evolutionary) processes are another option, tending to be more beneficial when customer requirements are imprecise or change over the duration of a project. User interfaces are a prime example: ‘ease of use’ as a requirement is vague and can be reinterpreted time and time again.

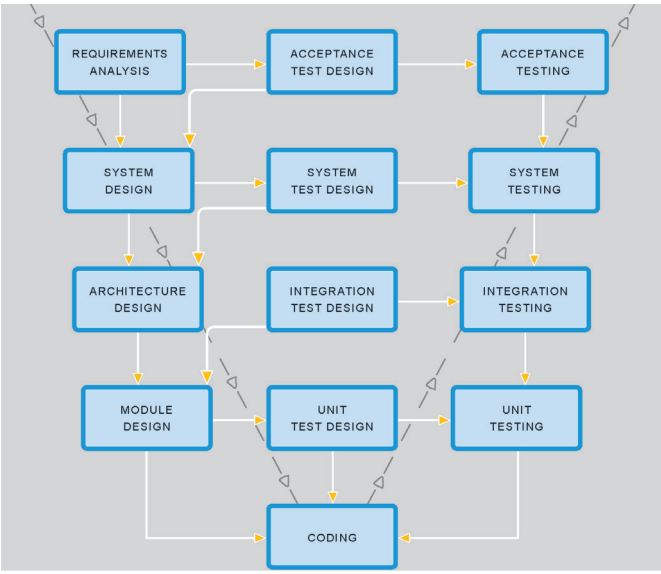

Figure 1: V-model

Agile approaches, such as Scrum, are highly effective in software development and are naturally also useful in projects to do with functional safety, in accordance with IEC 61508-3.

The following helps to shed some light on Scrum:

- Teams are motivated through effective communication, or ‘daily scrums’
- Teams are protected from unauthorised non-project access
- Capable and qualified product manager, or ‘product owner’
- Review phases during a project, or ‘sprint retrospectives’
- Metrics to measure project progress, ‘release/sprint burndowns’
- Prioritisation of requirements, or ‘product backlogs’

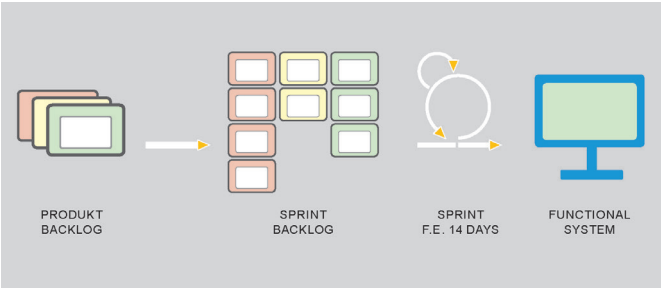

Figure 2: Scrum framework / agile approach

MESCO Engineering GmbH | Berner Weg 7 | 79539 Lörrach | Phone +49 7621 1575 0 | Fax +49 7621 1575 175 | info@mesco-engineering.com | www.mesco-engineering.com

- Iterative development for measuring project progress and motivation
- Daily (and nightly) builds of complete software, or ‘continuous integration’, to:
  - Generate a working version every day
  - Retain control over ‘header’ and build mechanisms
  - Be able to carry out quick reviews of the ‘living object’
  - Interact with the customer by means of early ‘prototypes’

Ultimately, there is no such thing as a ‘right’ or ‘wrong’ process when it comes to software development; there is only the right process for the project.

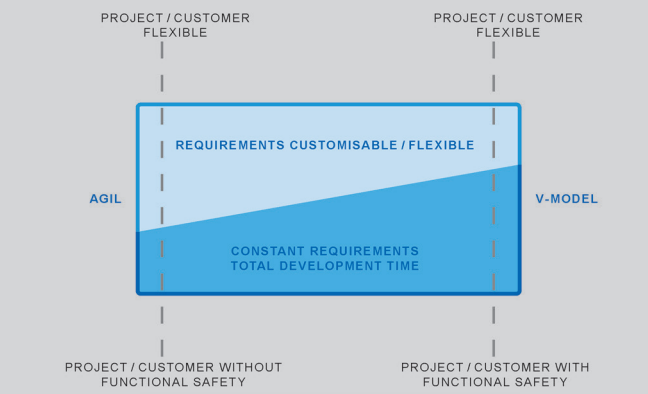

Figure 3: Agile approach vs. V-model

Software development processes are typically broken down into performance management processes (i.e. product, project and quality management), service provision processes (i.e. the actual core process), and support processes and methods.

An important component here is change management, or in other words, the way in which changes in requirements flow into the overall process. Traceability and documentation are also central elements here.

Support tools, toolchain

Requirement tracing tool

This tool allows you to write down, group, and link structured requirements. It typically also includes a module for the management and tracking of errors and anomalies and a module for the documentation and tracking of tests.

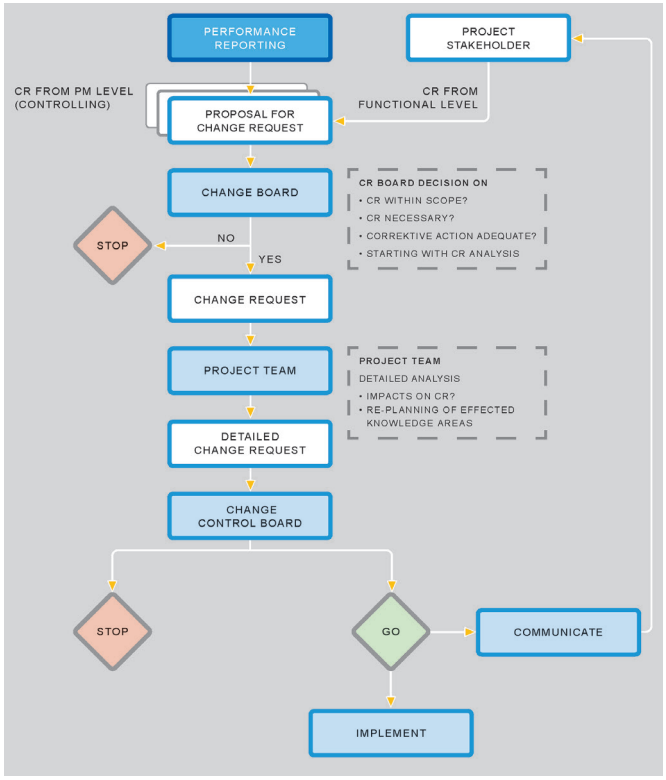

Figure 5: Change control / process

Links can be used to track whether each requirement has been processed, documented and tested. This system is the central hub of the project. Polarion (Siemens), Doors (IBM) and Enterprise Architect (Sparx Systems) are all examples of requirement tracking tools. In total, there are over 30 noted manufacturers of these systems, so it needs to be determined in advance which system characteristics are most important or useful for the respective company.

Version Control Server

Typically, new functionalities are implemented in a follow-up software version to the version currently used. At the same time, the current version has to be updated with bug fixes and corrections. The challenge then is to incorporate fixes that were introduced to the current version into the next version and at the same time prevent the functionalities of the next version from being incorporated into the current version. If multiple different versions are in development at the same time, they must be strictly separated from each other.

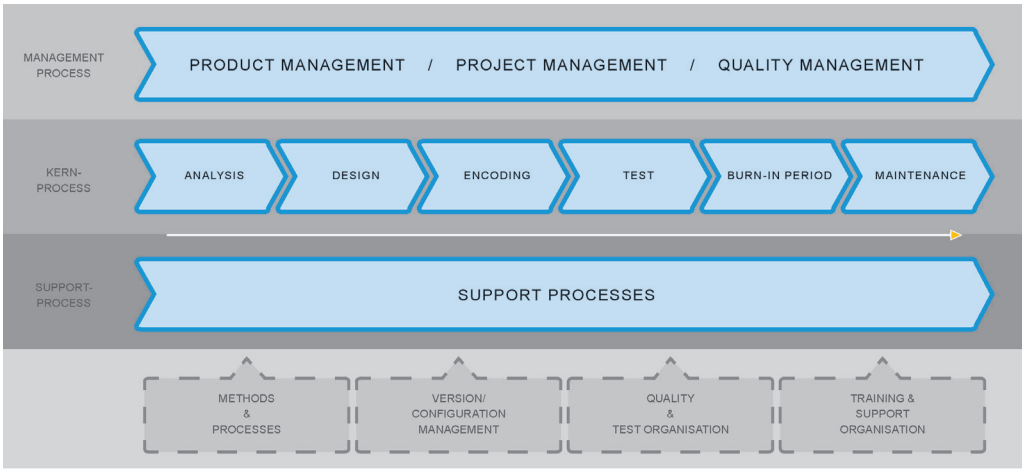

Figure 4: Overview of the development process

The second challenge is working together in a team on a piece of software. A developer may be responsible for all functionalities, for an entire object class, or just for a particular function or method.

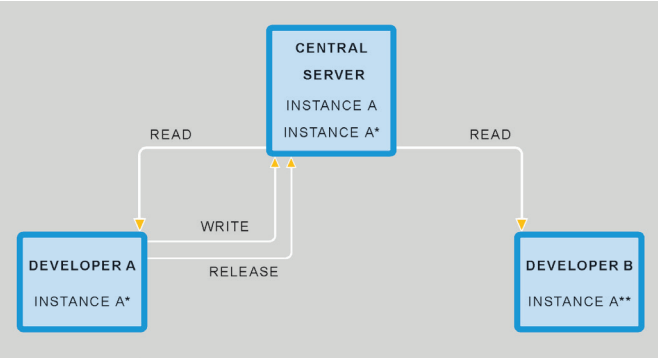

Figure 6: Version control with VCS and SVN

The compiler, linker and debugger are important components that are usually certified and approved by an auditing body for functional safety development. There are processor-specific libraries and settings, etc., some of which are made by processor manufacturers (ARM, Renesas) and others by specific workbench manufacturers (IAR, Keil). A TÜV certificate and corresponding report confirm that the specific compiler or workbench meets the requirements for tools in class T3 in accordance with IEC 61508-3. This allows customers to use the compiler or workbench for safety-related development up to SIL 3 (IEC 61508) or ASILD (ISO 26262) without the need for qualifications, provided the recommendations and conditions documented in the qualification kit are adhered to.

The functional safety standard IEC 61508 recommends that the C programming language only be used for the lowest Safety Integrity Level, SIL 1. For SIL 3 and SIL 4, C may only be used with the suitable coding standards and analysis tools. An important component for this is the programming standard MISRA-C. MISRA-C 2004 includes around 140 rules in 21 groups, including initialisation, data types, structures, and preprocessors. The aim of MISRA-C is to overcome the shortcomings of the C standard and prevent potential programming risks. Static code analysis tools such as PC-lint already support MISRA-C and, integrated into the toolchain, help meet the challenges of the C language for safety-related applications.

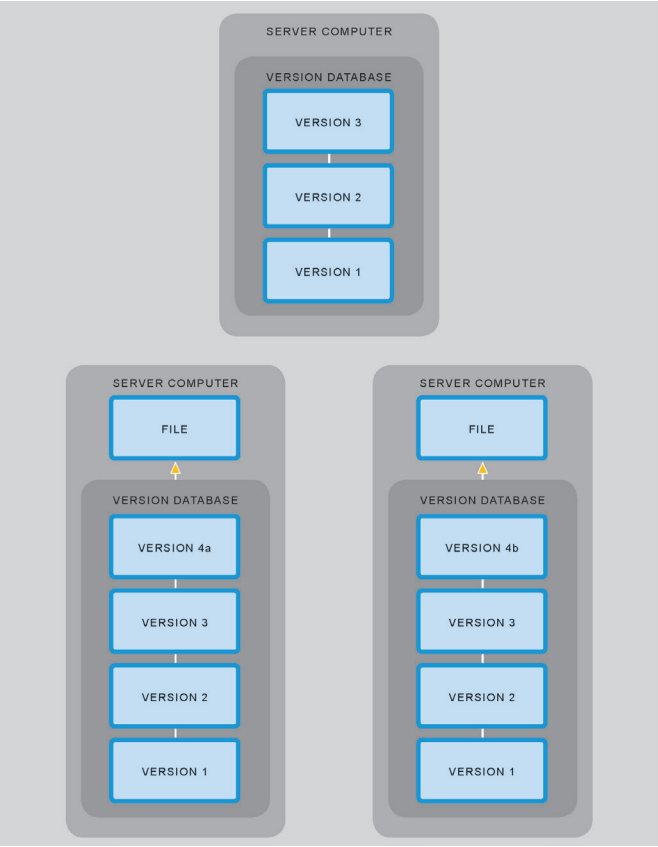

Figure 7: Git and Mercurial version control with local instances

The effective management of software artefacts, especially source code, is carried out by a version control server. Other important terms include source code configuration management (SCCM), revision control system (RCS), and version control system (VCS). These systems have been on the market for over 40 years now, on various platforms. Microsoft SourceSafe is one of the best-known systems for Windows, while CVS is well-known for open-source environments. CVS is widely used today, as is its derivative, Apache Subversion (SVN), which was developed as an alternative to CVS to eliminate some of its disadvantages.

Git is an entirely different variant, developed from Linus Torvalds' Linux environment. The biggest advantage of Git is its speed between dispersed sites when there is low bandwidth between them. However, for developers from the *classic* CVS environment, the learning curve is significantly larger, and the support for Windows servers is quite limited. An offshoot of this is Mercurial, which provides better Windows server support.

Again, there is no silver bullet and it ultimately depends on the developers, system administrators and project managers who will be using the systems.

Continuous integration server

The continuous integration server represents the central tool in which software is integrated, compiled and complete versions are made available for the build and test servers. Based on the integration server, statistical source code analysis tools are used frequently to influence the quality of the software at an early stage. Prominent examples of this include Jenkins and Phyton, both of which can be integrated into the aforementioned version control systems. With plug-in, these tools are highly powerful. From simple examinations of whether all variable objects have been initialised, to statistical statements about code coverage, there is something to make each software developer jump for joy.

Build and test servers

Build and test servers provide the validation and verification team with the latest trial versions and synchronise the hardware-in-the-loop (HIL) and device under test (DUT) tests.

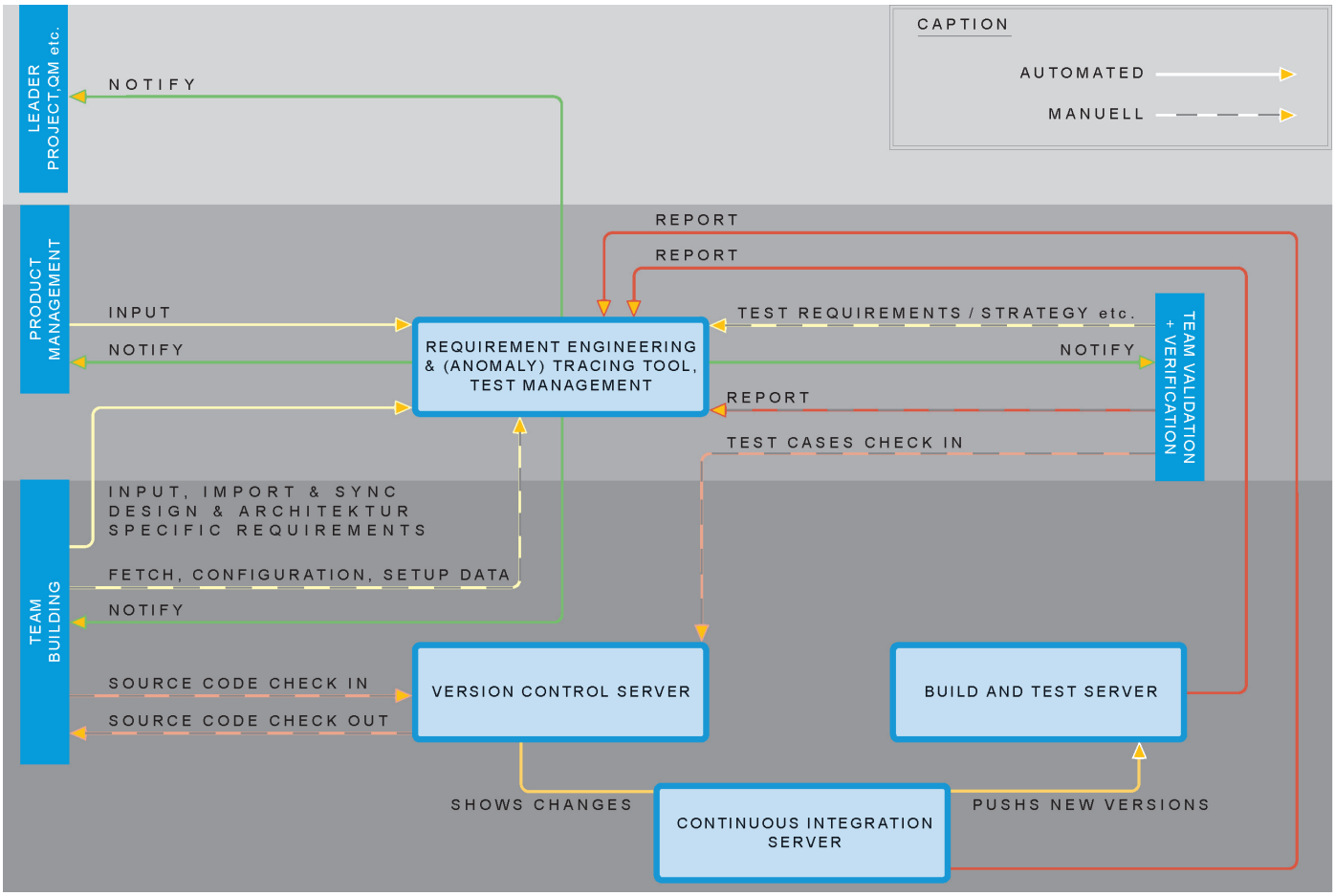

Figure 8: Overview of the software development toolchain

Overview and summary

The traceability from the requirement all the way up to the test and verifiable statistical statements results from the integration of the described tools, as shown in the following graphic.

The toolchain is a combination of processes and tools. Processes alone make it difficult to generate proof and traceability, and considerable effort is involved. Tools without integration and defined processes also come up short. The method of choice, therefore, is to integrate these tools intelligently and to combine them with smart processes that are adapted to the required Safety Integrity Level (from standard development without SIL requirements all the way up to SIL 3 requirements). However, setting up the processes and the toolchain cannot be done overnight. It requires a vision and the continuous implementation of individual steps until, after performing several phases of iteration – which can take years – the goal of process and tool integration is finally achieved.

Autor: Dr. Johann Pohany
Managing Director
Meditcine Consultants
www.meditcine.com

Autor: Armin Götzmann
Managing Director
MESCO Systems GmbH
www.mesco.de