Development Process for Production Ready Software based on R&D Projects

Martin Suesskraut, Jens Schmüdderich

2020

Preprint:

Development Process for Production Ready Software based on R&D Projects

Based on the Example Cooperative Risk Maps

Martin Süßkraut
SILISTRA SYSTEMS GMBH| KÖNIGSBRÜCKER STR. 124 | 01099 DRESDEN | GERMANY

Jens Schmüdderich
HONDA RESEARCH INSTITUTE EUROPE GMBH | CARL-LEGIEN-STR. 30 | 63073 OFFENBACH/MAIN | GERMANY

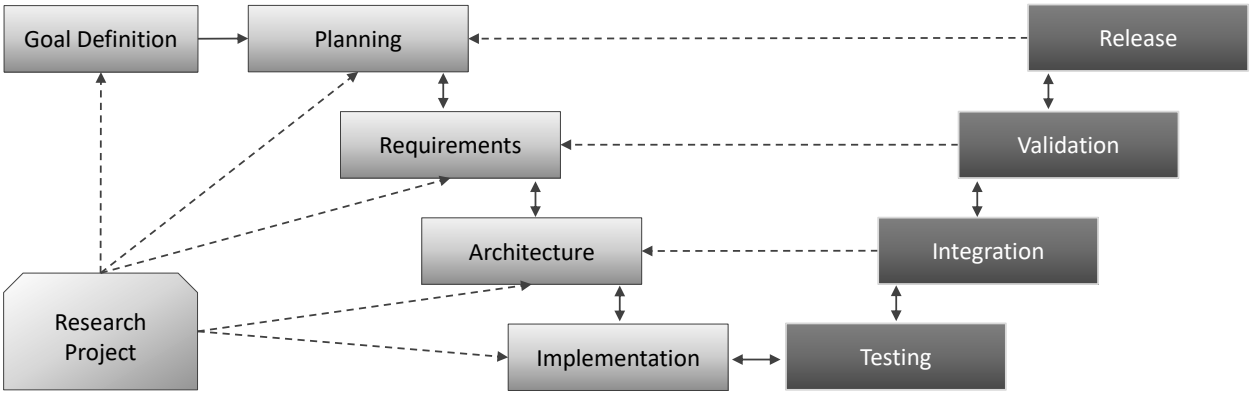
1 Executive Summary

The financial costs of software bugs do not require a motivation [51]. Especially in safety critical and cybersecurity critical systems the risk of failures and, hence, harm to people must be reduced to a negligible level. The standards for developing safety critical software and standards for cybersecurity critical software solve the problem of software bugs by requiring a structured software development process using state-of-the-art & proven-in-use software development approaches and techniques. The same solution also works for non-safety and non-cybersecurity critical software (called QM software development for quality managed software development). Because the earlier bugs are discovered, the lower the cost of fixing them [52].

This document presents a software development process for QM as well as safety & cybersecurity critical software. The process provides an interface for reusing artifacts from a research project that pre-dates the software development project.

The process introduced in this document focuses on application in the automotive industry based on the example project briefly introduced in Chapter 4, but can also be applied in fields with similar standards such as robotic and urban mobility. In either case, the software projects require a certain minimal size, as short-term projects with a length of just a few weeks and one (wo)man projects will not benefit from the outlined processes. Safety critical and cybersecurity critical projects, however, already require a structured development process which can neither be accomplished by one person nor realized in a few weeks for any meaningful projects.

The software development process of this document is based on the V-model [1]:



The left side of the diagram shows the creation phases from Goal Definition down to Implementation. Every phase towards the bottom becomes more concrete. Goal Definition defines the overall goal that the developed software should fulfill from a user's point of view. The Planning phase prepares the following phases and analyses the research project for its reusability in the following phases. In the Requirements phase the requirements engineering must be done to define the resulting software in detail on the technical level. In the Architecture phase the software architecture must be defined down to the SW units. Finally, these SW units must be specified and implemented in the Implementation phase.

The right side of the V-model is called V&V for Verification & Validation. This side verifies the corresponding phases on the left side. In Testing the SW units must be verified with automated unit tests, static & dynamic analysis, and code reviews. In the Integration phase the whole software (as a linked binary) must be tested on the target hardware. In the Validation phase the performed tests must be with against the requirements from the Requirement phase to determine: whether all requirements were tested successfully. From this analysis a release recommendation for management is derived. When management approves the release,

the final software together with all other work products of the software development process must be archived.

Chapter 5 describes the V-model for QM software development projects. Part of the Goal Definition is a Hazard and Risk Analysis to determine whether the software is safety critical and a Threat and Risk Analysis in case of cybersecurity critically, respectively. In either case, the software development process must adhere to the V-model of Chapter 6. This V-model is an extended variant intended for projects with safety and cybersecurity requirements, incorporating the requirements of the automotive safety standard ISO 26262 [2] and the automotive cybersecurity standard SAE J3061 [3]. Both standards have a holistic view on software and hardware. The extended V-model shares this view where necessary on the top most layer and focuses on the lower layers on software only. For pure software projects the safety element out-of-context approach of Section 6.1.1 is an alternative. In this case, the processes are based on assumed requirements from interacting components and hardware.

Chapter 7 introduces supporting processes that are required for QM, safety critical and cybersecurity critical software projects. The 5 support processes: Document Management, Configuration Management, Change Management, Tool Selection, and 3rd-party software qualification are required when applying the V-models of Chapter 5 and Chapter 6.

While the V-model already anticipates iterative development, its iterations are unstructured. The Scrum process presented in Chapter 8 gives these iterations structure and defines a process to learn from process failure and improve the process. According to Scrum, development is carried out in sprints. Each sprint lasts for a fixed period of time typically 2-4 weeks. A sprint starts with a planning meeting, where the work to be done in the current sprint is selected by the team based on the priorities and input of the project manager. At the end of the sprint the result is analyzed and compared with the initial planning. An additional retrospective afterwards identifies points in the process that require improvements. Section 8.2 explains how to iterate the V-model of the three previous chapters and how to map the roles of safety critical and cybersecurity critical software projects to Scrum.

Chapter 9 takes a broader view starting with the research project before the software development project up to support of users and customers when the resulting software is in the field. Chapter 9.1 gives recommendations on how to prepare a research project for reuse. in order to avoid a complete reimplementation. Chapter 9.3 defines a three-level support model. This model can give quick feedback to the user and customer in supported cases. And it only requires cooperation of from the experts from development when required.

The study finishes in Chapter 10 with a list of suitable tools to support the processes described throughout in this document. Tools are already mentioned throughout the document where the process that benefits is introduced. Chapter 10 gives are good overview of these tools for summary. Naturally, all mentioned tools can be replaced in part or completely with other appropriate ones. The presented selection is based primarily on the practical experience of this study's authors.

This study (incl. "Current Development Process at Honda Research Institute Europe") was commissioned by Honda Research Institute Europe and done by SIListra Systems GmbH from January to May 2020. The example use case from Chapter 4 was also provided by Honda Research Institute Europe.

2 Table of contents

3 Introduction and Scope

The scope of this document is the development of software projects which are safety and/or cybersecurity critical. This document also covers the processes required to determine the safety and cybersecurity criticality of a project. And, based on these criticality analyses it is shown how to derive the requirements of the software. The presented development process is based on the automotive safety standard ISO 26262 and the automotive cybersecurity standard SAE J3061, but also incorporates an iterative and incremental approach based on Scrum.

Out of scope for this document is the hardware development of safety and/or cybersecurity critical systems. Maintenance is only briefly introduced. Also, the required organizational security structure is not covered by this document. The authors assume that appropriate organizational and technical security measures are assured.

4 Example Use Case

The requirements of the development process presented in this document were extrapolated from the HRI Project "Cooperative Risk Maps". The goal of this project is to do behavior planning of autonomous vehicles based on a quantitative risk evaluation and risk analysis.

The goal of this document is to present a universally applicable process. Consequently, it is not directly tailored towards any specific project, but draws the following conclusions from the "Cooperative Risk Maps" project:

- Application domain: automotive
- Safety relevant
- Security relevant

While an autonomous vehicle is always safety-critical, security becomes relevant as soon as any part of the vehicle is accessible by any malicious 3rd party. In the absence of better knowledge, the security implications must be analyzed and evaluated at the beginning of the development process.

The rest of the document does not address this example project any further, but takes application domain, safety, and security into account.

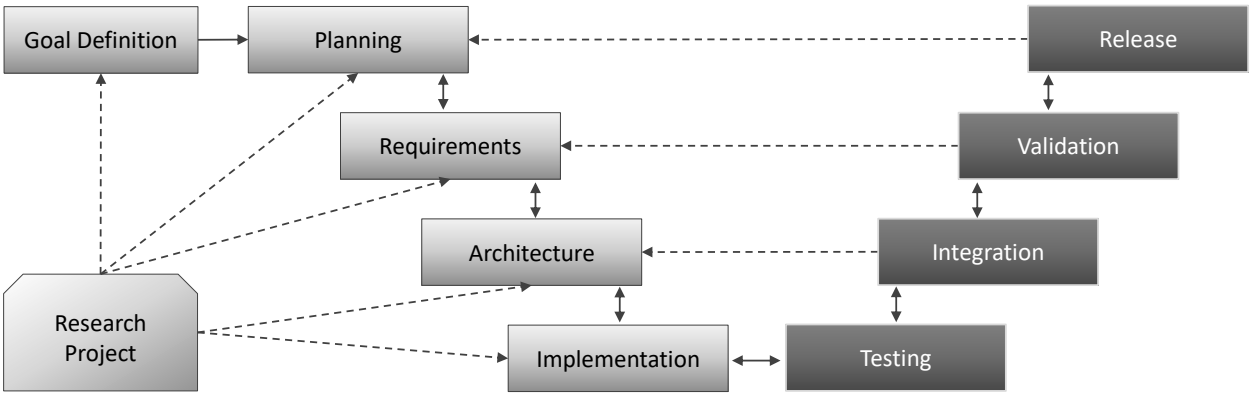
5 V-Model Without Safety and Security Requirements

This chapter describes a V-model for software development without any safety and cybersecurity requirements. The V-model is also used in the safety and security standards and will be the basis for the software development process with safety and cybersecurity requirements of the following chapter.

Although the V-model defines work products and their dependencies it is a rather "static view" on a software development project. It only defines what must be produced and in what order. But it does not define how to produce its work products. Therefore, this document will later introduce as a "dynamic view" an interactive incremental process based on Scrum.

The V-model consists of phases. Each phase consists of activities, that result in work products. The generic term work product refers to artifacts such as documents, analysis, reviews, and source code. An activity is always performed by a human (e. g. developer, project manager, reviewer, test engineer, safety specialist) typically with the help of appropriate tools. A work product produced or updated in a phase is an output of its phase. Subsequent phases require work products from previous phases as input.

The described development process assumes there is a previous research project with artifacts (publications, source code, and documentation) upon which the software to develop should be based. However, the development process makes no assumptions about the reusability of those artifacts, but includes activities to review the artifacts of the research project in order to determine their reusability.



The diagram above shows the generic V-model. The left side is the creation. The first phase on the left side is the goal definition and the final most concrete phase is the SW Unit Design & Implementation. These phases build on each other and reviews (dashed lines) ensure that a follow-up phase was correctly derived from its predecessors.

The right side is the verification counterpart. Each phase except the goal definition is mirrored by a corresponding verification phase (dotted lines). Sources for this V-model are [1] not taking safety and security into account, but also the safety standard ISO 26262 [2] and the cybersecurity standard SAE J3061 [3].

In the following each phase is described with its inputs, activities, and output work products.

Projects without any safety and cybersecurity requirements are called QM (quality managed) projects.

5.1 Goal Definition

- Input: Research project with
  - Documentation (e. g., publications, tech reports)

  - Source Code
- Activities:
  - Define goal of the project
  - Define HW assumptions for the SW project
    - Based on system level definition (see 6.5.2.3)
- Output: Goal definition (Item definition)
  - If safety or security requirements cannot be completely excluded, start over with safety/security enhanced process
  - HW definition

The goal definition documents the final product that should be developed in this software development project. For an automotive project, the item definition format (introduced in 6.5.1.1) can be used.

Note that, formally the decision, whether such a project is safety critical or cybersecurity critical, can only be derived from a hazard and risk analysis for safety (see 6.5.1.3) and a threat and risk analysis for cybersecurity (see 6.5.1.4) for automotive projects. Products which do not influence any safety critical actors are definitively QM.

The goal definition should not take the implementation of the research project into account. However, any relevant documentation of use cases from the research project can be reused for the goal definition, if applicable.

The goal definition must also contain assumptions about the HW target platform (server, desktop, embedded, mobile) as this decision influences the following phases.

5.2 Initiation of SW Development (Planning)

- Inputs:
  - Goal Definition
  - Research project
- Activities:
  - Gap Analysis with respect to rest of V-model
  - Planning
- Output:
  - GAP List with respect to the rest of the V-model
    - Artifacts to build completely new
    - Artifacts to extend/rework as part of the development
  - Project plan

The output of the planning phase is the project plan. This plan is a list of tickets in a ticket management system (e.g. Jira [40]), which define the next phases. In the planning phase, only a rough planning of the following phases is required. Any further detail tickets (e.g. about concreate SW Units) are derived from these phase planning tickets within the following phases, i.e. the project plan needs to be updated in the following phases. The following phase planning tickets are required:

- SW requirement engineering
- SW Architecture Design
- SW Unit Design & Implementation
- SW Unit Testing
- SW Integration Testing
- Verification of SW Requirements
- Releasing

For the activities documented in these tickets see the following phases. When planning the detailed tickets, role independence must be ensured, i.e. the implementer of a work product must not be the same person as the reviewer. Usually any work products must be reviewed whether it matches the requirements defined in its ticket. These review results should be documented in this ticket. Follow-up tickets (e.g. tickets for the SW Units derived from the Architecture) are also work products.

When creating the project plan the research project should be analyzed for artifacts that can be reused in the project. This reuse must be planned in the project plan. In the appropriate phase the reusable artifacts should be listed. For instance, source code should be assigned to the ticket for the SW Unit Design & Implementation.

When entering such a phase with listed reusable artifacts these artifacts must be considered, when planning the phase, i.e. added to the follow-up tickets. Then a GAP-analysis must be done, to derive if the artifact is either completely reusable or must be reworked (and to which extend). This GAP-analysis takes the previous phases into account (e.g. requirements and architecture or a test specification) and the artifact itself.

Beside the tickets the project plan must link to the documentation of the supported processes to use. This is especially important if there are several variants of a supporting process used in the company (e.g. different configuration management processes based on Subversion and git).

Also, the team members with their roles should be named in the project plan. At a minimum the roles for Scrum are required for a QM project (see 8.1.1). Having more specific roles like testers and architects is possible, but not necessary and would be against the Scrum principles. Small projects have usually developers which must fulfill all the specific roles. Dedicated experts with special skills can consult in the project.

In practice the project plan could be a wiki page with

- Links to the used supporting processes,
- A table of project members and their roles and
- A list of tickets (usually just a view into the ticket management system, e.g. with a project specific filter)

5.3 SW Requirements

- Input:
  - Goal Definition with use cases
  - HW definition
- Activity:
  - Requirement specification
- Output: Functional requirements

In this phase the software requirements must be specified. A requirement is defined as [4]:
A requirement describes what a user or customer of a product expects (condition, attributes, goal, utility).
A requirement is a documented representation of an attribute or condition, that either

- a user (person or system) requires to solve a problem or reach a goal or
- that a system or system component must fulfill to meet a contract, standard, specification or other formal document.

Requirements consist of a set of attributes:

- Unique ID
  - The ID must be unique for the whole requirement specification document.

    - Best practice is to use companywide unique ids or requirements.
    - Requirement ids must not be reused. Best is to just deprecate old requirements (see status below).
- Category
    - Safety, security, or function
    - Typical between QM and ASIL-D for safety
- Level
    - For QM only component level is required
    - For Safety & security:
        - Function level
        - System level
        - Component level
        - Unit level (done in SW Architectural Design)
- Status:
    - Draft
    - Approved
    - Deprecated
- Link:
    - Requirements derived usually from a source. That can be a use case, a safety goal or another requirement (e.g. a technical safety requirement is derived from a function safety requirement). This link has to be documented.
- Description:
    - Keywords [5]:
        - Must (required, shall)
        - Must not (shall not)
        - Should (recommended)
        - Should not (not recommended)
        - May (optional)
- Baseline:
    - The product version(s) for which this requirement applies. This attribute can also be used for managing variants of the product (e.g. for different operating systems).

Requirements must be testable/measurable. It does not matter whether the testing can be automated or not. A bad example: “The box must be beautiful.” – This description is subjective and can neither be tested or measured. A better specification would be “At least 20% of users must rate this box as beautiful.” This description can be measured asking users for feedback.

New requirements have always the status Draft. After a requirement was successful reviewed it enters the status Approved.

Requirements can be hierarchical. For example:

- High Level Requirements (HLR) are derived for use cases.
- Low Level Requirements (LLR) are derived from HLR.

It is also possible to derive LLRs from other LLRs to create additional hierarchy levels.

Overall, the requirements of a product must be a coherent document. It can be versioned with the help of the baseline attribute. All requirements to implement must be in status Approved.

In general, the requirements should be problem oriented and not solution oriented. The “what” is more important than the “how”. This helps to keep the solution space open.

One way to document requirements is to use a ticket management system and define each requirement as one ticket. Therefore, a ticket type “Requirement” should be created and assigned with the required attributes and an appropriate life cycle (sometimes called workflow).

5.4 SW Architectural Design

- Input:
  - Functional requirements
  - GAP Analysis
  - Use cases
  - Goal Definition
- Activity:
  - Design SW architecture model
    - Incl. UML diagrams
      - Structure diagrams: e.g. component, class diagrams
      - Activity diagrams: e.g. sequence diagrams, state machines diagrams
    - Map requirements for components, class, etc.
    - Refine requirements form the previous phase.
- Output:
  - SW architecture model
  - Refined requirements form the previous phase
  - Units with requirements
  - Refined HW definition if necessary

The architecture splits the SW product to implement into smaller SW components and defines the relationship between these components. The best practice is to model the architecture with the help of UML [20].

The architecture or parts thereof can come from a research project. If necessary, the architecture of the research project should be refined down to unit level (see below). The goal is to reuse as many units as possible from the research project.

Relationships between components can be modeled using structure and activity diagrams. Having at least structure diagrams for the most central components is best practice. Activity diagrams should be used for complex dynamic relationships. The design should be broken down onto the unit level for the next phase, i.e. components are broken down into sub-components until “atomic” units are reached. Units are typically a set of functions/methods that work on one common data structure, e.g. a unit in C++ or Python is usually one class.

All requirements from the previous phase should be mapped onto the sub-components and units of the architecture. If a requirement is implemented by more than one unit or sub-component, this requirement should be refined. Refinement means to derived two or more requirements (at-least one per implementation unit/sub-component). All these sub-requirements together implement the refined requirement. These relationships between requirements must be documented with the link attribute.

Also, the project plan must be reviewed.

If necessary, the HW assumptions need to be updated in this phase.

5.5 SW Unit Design & Implementation

- Input:
  - Planned Changes
    - Functional requirements
    - GAP Analysis
    - HW definition
    - Use cases

        - Goal Definition
- Activities:
    - Unit Specification (as ticket)
    - Unit Implementation in a new git branch
- Output:
    - Unit specification
    - Unit Implementation

The unit design can be done in tickets derived from the unit of the architecture of the previous phases and the requirements from the two previous phases. Per Unit one ticket should be created. This ticket should define a high-level interface description of the unit.

The implementation should be done on a separate branch, that is linked to the unit ticket. Each commit on this branch (for the unit implementation and unit tests) should also be linked to the unit ticket.

If possible, the unit implementation from the research project can be reused. Therefore, the implementation should be committed just as it would have been manually programmed. Of cause the unit implementation must be compared with the unit specification (incl. requirements). If necessary, the unit implementation from the research project must be updated to fulfil the specification.

5.6 SW Unit Testing

- Input:
    - Test plan
    - Planned Changes
        - Function requirements
        - GAP Analysis
        - HW definition
        - Use cases
        - Goal Definition
    - Unit specification & implementation
- Activities:
    - Program & run unit tests
        - Incl. dynamic analysis
    - Run static analysis
    - Code review
- Output:
    - Automated unit tests
        - Mocking and Stubbing [6]
        - Each test links to the requirement(s) it tests.
    - Unit test results

In this phase the unit tests shall be defined and implemented. The definition should happen in a separate ticket per unit. Hence, each unit ticket links to its unit testing ticket. The test specification (in the unit testing ticket) specifies the unit tests to implement for this unit. These tests should be derived from the requirements of this unit.

The commits of the test implementation shall be linked to unit testing ticket and committed to the same branch as the unit implementation. Unit tests run together with the static analysis on the build server (e.g. Jenkins). The test results (from running the unit tests) shall be linked to the unit testing ticket.

Like the unit implementation the unit tests could be reused from the research project.

There are several frameworks for writing unit tests:

- C/C++: GoogleTest (incl. GoogleMock) [18], Catch2 [21]
- Python: unittest [22] (incl. unittest.mock [23])

A useful technique is mocking to test units depending on other units' isolation [6]. Usually, unit level testing is done automatically.

Each unit test must be linked to the requirements it tests. How this link is implemented depends on the unit test framework and the tooling to calculate the test coverage (of the requirements).

When running unit tests additional dynamic analysis techniques can be used. For example:

- Compiler based: Clangs sanitizer flags [24]
- Emulator based: Valgrind [25]
- Code coverage for C/C++: Bullseye [26], CTC++ [27]
- Code coverage for Python: coverage.py [28]

Code coverage is especially important because it gives feedback on how much code is tested with the unit tests. While a high code coverage is desirable, it is often not reached in practice, because some code paths are difficult to test (e.g. error handling). Furthermore, even 100% code coverage in unit tests does not guarantee the absence of bugs.

Static analysis is used to find bugs and mistakes in the code without running it. Tools for static analysis are:

- Compilers: modern compilers include a wide range of code analyses most notably warnings
- Formating: Clang-format for C++ [29]
- Linting for C++: cppcheck [30], clang tidy [31]
- Misra C/C++: Klocwork [33]
- Linting for Python: Pylint [32]

Configuration and setup of static analysis tools highly depends on the application to build and its requirements. Static analysis is prone to false positives, i.e. it might flag coding errors, that do no harm under the given setup. In such cases the exceptions can be annotated, e. g with special code comments. The static analysis tools ignore the error for the annotated parts of the code.

The code review should be done on the branch as pull request. Also, the unit tests must be reviewed, if the really implemented the specified tests. Code reviews should include:

- Does the unit implement its specification?
- Are specificied unit tests implemented?
- Are all unit tests implemented as specificed?
- Does the code follow the coding conventions?
- Are annotated static analysis exception justified?

Additionally, the reviewer must also check the test reports of the unit testing, static and dynamic analysis:

- Are all unit tests run and pass successfully?
- Is the code coverage acceptable?
- Does the static analysis find errors?
- Does dynamic analysis find errors?

This phase and the previous phase (SW Unit Design & Implementation) must be done iteratively. Any failures that are found in this phase must be addressed in the SW Unit Design & Implementation phase. Failures could be discovered in:

- Static analysis
- Dynamic analysis
- Testing
- Code Review

Discovered failures must be fixed on the original unit tickets. Hence, unit tickets may only be closed after the SW Unit Testing phase was successfully completed.

A good practice is to start writing tests before the unit implementation. This approach is called test driven development (short: TDD) [6].

5.7 SW Integration Testing

- Input:
  - SW Requirements
    - Each test links to the requirement(s) it tests.
  - Planned Changes
    - Function requirements
    - GAP Analysis
    - HW definition
    - Use cases
    - Goal definition
- Activity:
  - Link units to final software
  - Implement & run integration tests
- Output:
  - Integrated SW
  - Automated integration tests
  - Manual tests
  - Integration test results

In the SW Integration Testing phase, the final software must be linked together from the individual units (for compiled languages). This final software is then tested by the integration tests.

Integration tests are based on the requirements from the SW Requirements phase. Hence, integration tests for these requirements must be written. Integration tests may be either automated or manual. Even manual tests require a test strategy, e.g. a reviewed excel sheet with the individual test phases and test results, or directly documented in the test ticket based on a template (again with test phases and test results). Automated tests require a specification and an implementation (both to be reviewed). The implementation of automated integration tests can become quite large, especially, if they need to interact with other components and/or tools (e.g. simulations). Manual test should also look at the generated documentation, i.e. is the manual correctly generated (content is reviewed as part of code reviews), but main links, table of contents, version information, pictures etc. should be given a quick review. In practice, there is rarely a project that has no manual tests, because some manual testing is too difficult to automate (like the check of the user manual above). However, the aim should be to automate all tests:

- which should be repeatedly run,
- where automation is simpler than manual testing in the long run, and
- that are to error-prone to be done manually.

There are tools for supporting integration tests (automated and manual).

As with automated unit tests all integration tests must be linked to the tested requirements.

All integration tests must be run on the target hardware.

The research project may already contain some integration tests, that could be reused. Therefore, the GAP analysis for integration tests should be done before the specification of the integration tests. In this way it is possible to integrate existing tests from the research projects (if necessary, with improvements). Note, that most tests in research projects are integration tests.

5.8 Verification of SW Requirements

- Input:
  - Functional requirements
  - Unit test results
  - Integration test results
  - Planned Changes
  - Release Checklist
- Activity:
  - Is quality good enough for a release?
    - Analyze and evaluate test results with respect to requirements
    - Work through release checklist
  - Walk through release checklist (based on releasing ticket)
- Output:
  - Release recommendation
  - Integrated SW

The goal of the verification of the SW requirements is to decide whether the software can be released. Therefore, the release checklist (which is part of the releasing ticket) must be walked through. This check list should contain the following items:

- Version documented
  - Is there a version page with complete and correct information in the wiki?
    - Version information
    - Change log
    - Summary of changes
    - Links to all tickets planned for this version
    - Link to releasing ticket
  - Change log file complete (if any)?
- Legal
  - Is the licensing information complete?
  - Is all legally required documentation included?
- Release in configuration & change management
  - Is there a release branch with the correct name in git?
  - What is the test status on Jenkins of the release branch?
  - Is the version in Jira closed?
- Quality
  - Are all requirements covered by the tests?
    - If not, can the software still be released? And why?
  - Do all tests pass?
    - If not, can the software still be released? And why?
  - Are all tickets of the project plan closed?
- Archive
  - Is the integrated SW archived?

The points under quality are about the traceability. The answers to these questions highly influence the release recommendation for the management. There might be other points in it, e.g. for announcing the release, marketing or triggering a follow-up version.

The results of this phase are presented to the management, that must decide on the release. In safety and/or security projects this management decision is replaced with audits and assessments of the functional safety and security experts (see the next section about this topic).

5.9 Released SW

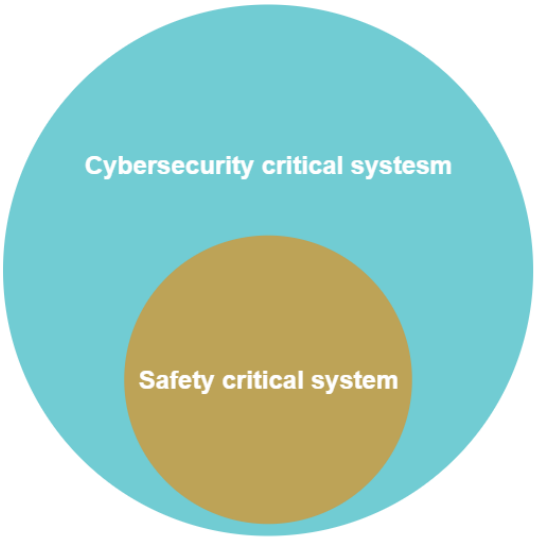
In this phase the management has decided about the release. From a software development point it must be ensured that all documentation about the software development and the resulting products are archived:

- Release recommendation
- Integrated SW
- Test results
- Goal definition
- Project Plan
    - Test results

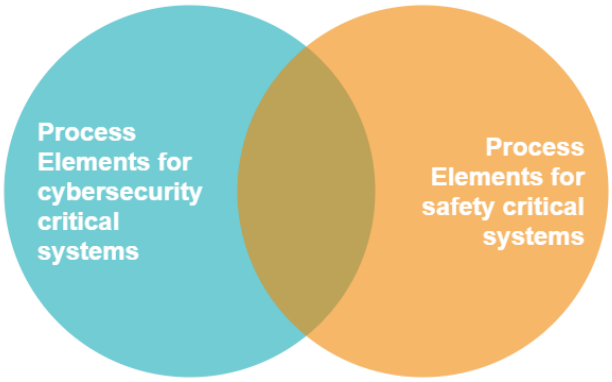
If everything is documented in Confluence, Jira, BitBucket (git) and Artifactory, then these tools must just be configured, that accidental deletion of old tickets, version information etc. cannot easily happen. For instance, closed versions and tickets should not be deleted by any user.

6 V-Model for Safety & Security Software

All safety critical systems are also cybersecurity critical systems [3]. Whereas not all cybersecurity critical systems are also safety critical systems. The engineering methods for these two kinds of systems have some overlap, but are not 100% identical. The following figure illustrates the relationships [3]:



Releationship between cybersecurity critical systems and safety critical systems.



Releationship between engineering methods for cybersecurity critical systems and safety critical systems.

Because of the common overlap it makes sense to describe the SW development process for both safety- and cybersecurity critical systems together.

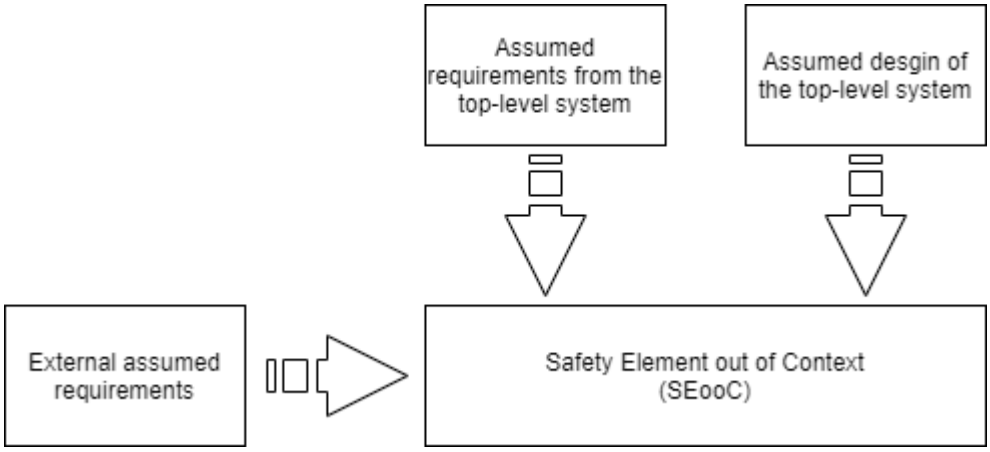
6.1 General comments about Safety Software

The safety elements of the process described in this section focus on the ISO 26262 2nd edition – the automotive safety standard. Safety (and security) do not only apply to SW alone. In most systems safety is delivered by the implementation of SW and HW. Hence, this section contains additional steps for HW and system design.

The overall goal of safety is to reduce the risk of systematic failures (i.e. "bugs" in requirements, design, SW and HW) to an acceptable level (i.e. negligibly) and to handle random HW failure. The risk reduction of systematic failures is done by a process that controls the source of such systematic failures. Random HW failures are mostly out of scope for this document which focuses on SW.

Hence, the process (especially its initial phases) are much broader than for QM software. The V-model starts with an item definition on a vehicle level and ends with a validation, which must also be done on the vehicle level.

6.1.1 Safety Element Out-of-context



- Solution for SW-only projects: SEooC [13]
  - SEooC = Safety element out-of context
  - SW-Project only for an specific SW-component of an item
    - From now on called "SW-component"
  - For the specific SW-project make assumptions about the surrounding item [12]
    - Incl. all elements (incl. their main HW- & SW-components) surrounding the SW-component to develop
  - Do the following analysis like you have to develop the whole item
  - Document the item and especially the assumptions
    - Incl. safety plan, safety concept, …
  - When using the SW-project in a specific item, the assumptions have to be re-evaluated and must be fulfilled
    - otherwise the SW-project must not be used (without further measures) for this specific item
- assumption for rest of section: one SW-component is developed as SEooC

6.2 General comments about Security Software

The automotive cybersecurity standard ISO 21434 is currently in development [14]. Hence, there is no established standard for automotive cybersecurity yet. The requirements for such a standard are:

- Applicable to the application domain (i.e. automotive)
- Works together with ISO 26262 or at least does not contradict ISO 26262 with its requirements.

The described security process in this chapter follows SAE J3061 [3]. This chapter merges the important steps of security and safety development. SAE J3061 is explicitly developed for projects that follow already the ISO 26262. However, security only projects (i.e. without safety requirements) are also possible.

Like with safety, security is a system wide non-functional property and hence cannot ensured by just looking at one SW component. However, in contrast to the ISO 26262 the SAE J3061 has no notion of a "Security element out of context". Nevertheless, the principles from 6.1.1 can be applied on a "Security element out of context": Anything that is required for deriving SW security requirements has to be based on reasonable assumptions. That are:

- Any external assumed cybersecurity requirements needed for other items.
- Assumed functional cybersecurity requirements from a assumed cybersecurity concept on system level.

- Assumed system level design with respect to the cybersecurity concept and requirements.

6.2.1 Status of SAE J3061

The ISO 26262 for automotive safety is much more detailed and concrete than the SAE J3061 for security. The main topics were SAE J3061 does not give concrete advice are:

- Thread and risk analysis (TARA):
  - The ISO 26262 describes exactly one approach to do a Hazard and risk analysis (HARA). The SAE J3061 introduces several approaches and technices for the TARA and does not favor any of them. Because these approaches all work differently they also produce different results that are not comparable with each other and do not produce a coherent security levels.
- Security Levels
  - Whereas the ISO 26262 knows QM and ASIL-A to ASIL-D, SAE J3061 does not introduce any security levels. The reason for this are the different TARA methods, which each producing their own unique levels.
- Methods
  - ISO 26262 recommends methods for reviews, implementation and testing based on the derived safety level (ASIL). Because SAE J3061 does not introduce a unified model for security levels, methods are just listed but a concrete recommandation based on security levels.

Hence, the recommendations in this chapter for security are much less concrete than for safety. After the release of the ISO 21434 this chapter should be reviewed and might need to be reworked. The release is planned for the end of 2020.

6.3 Roles

Here are the roles required to develop a safety critical or cybersecurity critical product:

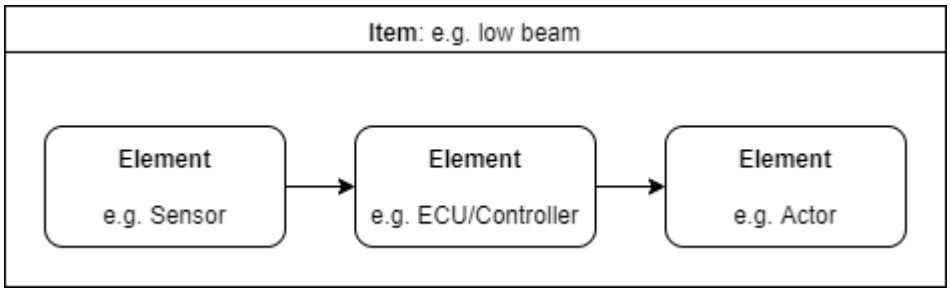
- Product manager/owner
  - Ensures safety activities are performed – compliance with ISO 26262
  - Ensures security activities are performed – compliance with SAE J3061
  - Ensures sufficient resources are available
- Safety Manager
  - May also be the product manager/product owner
  - Responsible for functional safety management
    - Safety plan
      - Creates, updates and monitors activities against plan
    - Plans and monitors safety activities in sub phases
    - Assigns safety activities to team
  - Should have safety experience and education
- Security Manager
  - Similar to Safety Manager
  - May also be the product manager/product owner
  - Responsible for cybersecurity management
    - Security plan
      - Creates, updates and monitors activities against plan
    - Plans and monitors security activities in sub phases
    - Assigns safety activities to team

    - Should have security experience and education
- Safety Engineer
    - "normal" developer maybe with some safety experience or education
    - Development
    - Testing
    - Documentation
    - Requirement engineering
    - Design
    - Maybe only some of these
- Security Engineer
    - "normal" developer maybe with some security experience or education
    - Similar to Safety Engineer
    - Development
    - Testing
    - Documentation
    - Requirement engineering
    - Design
    - Maybe only some of these
- Independent Safety assessor
    - Needs to be independent of department
    - Maybe extern or customer
    - Expert in Safety (education + experience)
- Independent Security assessor
    - Needs to be independent of department
    - Maybe extern or customer
    - Expert in Security (education + experience)

6.4 Whole Safety / Security Life Cycle of an item

The safety and security standards are always applied to a whole item.

6.4.1 Item and Element

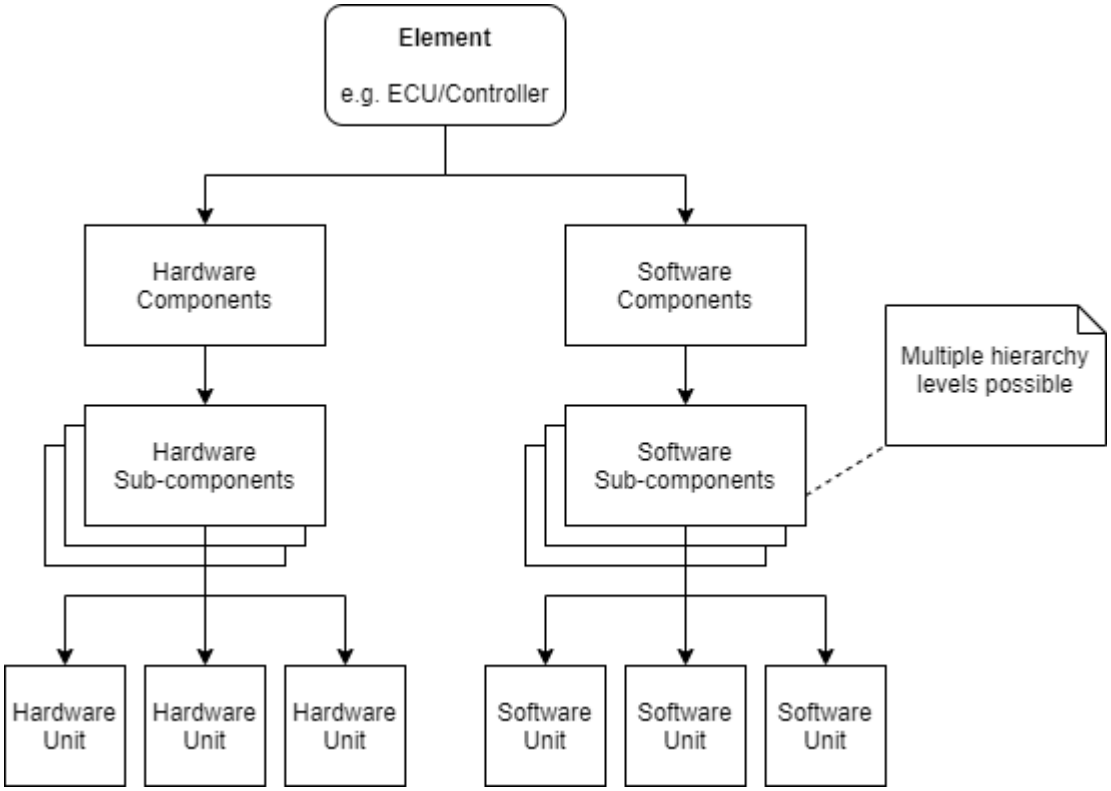


An item implements a function on the vehicle level (e.g. low beam, steering-column lock). The item is composed of elements – usually:

- Sensors
- Controllers
- Actors

Elements can be shared between items (e.g. low beam and windscreen wiper both may use the rain sensor). Elements that are not electrical/electronical/programmable (e.g. mechanical) are usually not relevant to this document. For example, for drive-by-wire-steering the steering wheel sensor would be an interesting item, but not the steering wheel itself.

6.4.2 Element, Components, subcomponents, units



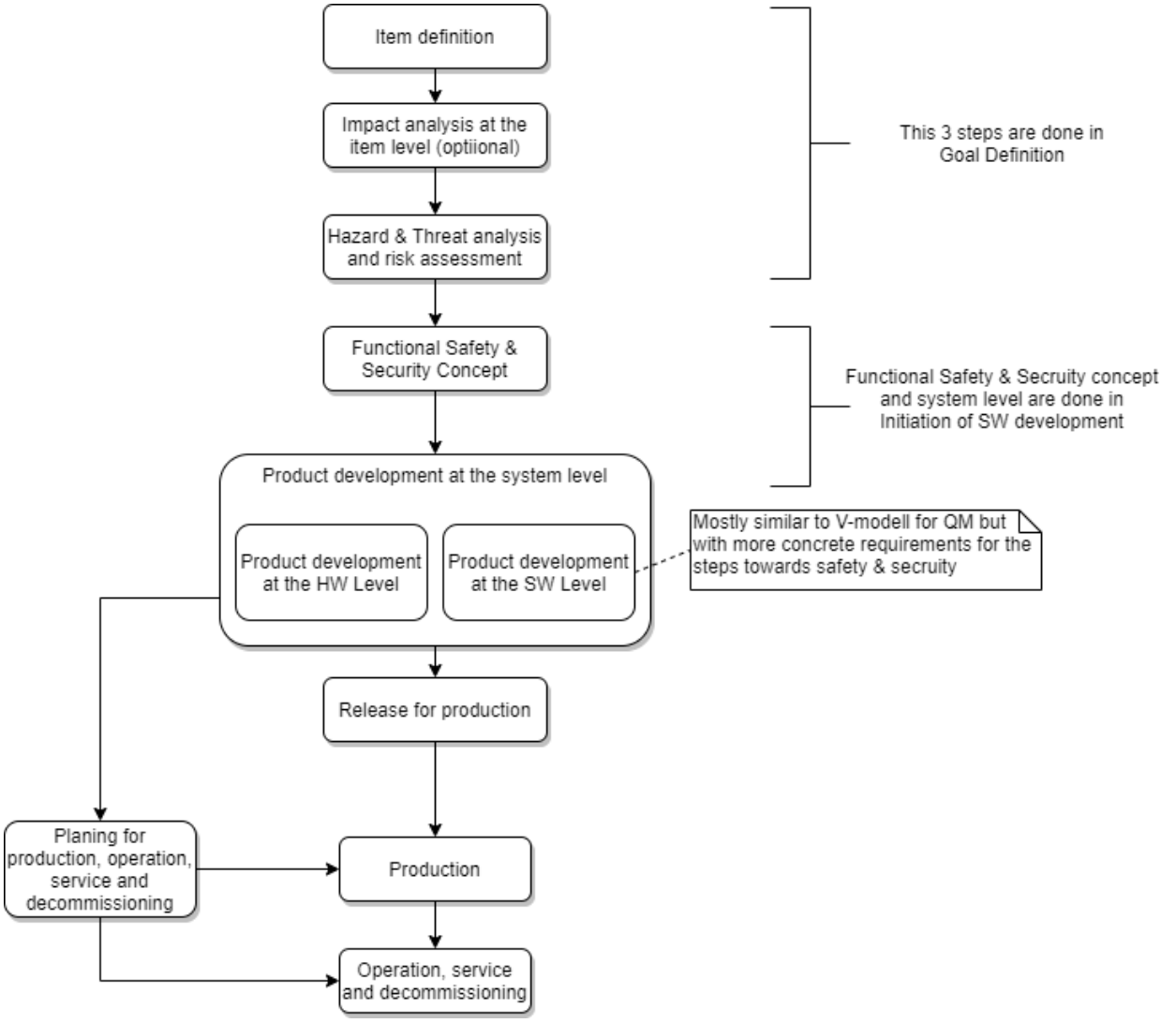
E/E Elements are composed of hardware components and most often also software components. Hardware only elements are not relevant for this document. A component is composed of sub-components (multiple levels of sub-components are possible). The sub-component on the lowest level is called unit.

6.4.3 Life Cycle

The safety and security life cycle is more extensive, than it is for QM projects. The goal is first to *systematically* derive the safety and security goals for an item in a hazard analysis and risk assessment (HARA) for the safety goals and a threat analysis and risk assessment (TARA) for the security goals. Both analyses require an item definition as input.

If a project builds on a previous project an impact analysis can be done after the item definition to identify the gaps between the previous project and the new item to build. With this gap analysis the following phases can be tailored by reusing as much as possible from the previous project. The impact analysis makes only sense if the previous projects already followed the process described in this chapter. For a research project that should be converted into production projects either start with the item definition and do the HARA and TARA or get the Safety and Security Goals from the user/customer.

Typically, an OEM does at least all phases until the product development at the system level and provides its Tier1 with the determined concepts, designs and requirements.



The goal of the operation, service and decommissioning phase is to do field observations with the goal to detect and handle defects (safety or security). Identified defects may lead to change requests (e.g. bug fixing). A best practice for this approach is described in 9.3.

6.4.4 Mandatory Reviews

All security & safety related work products require reviews by experts in the appropriate field (safety, security, but also requirements engineering, specific programming languages, specific hardware, …). The ISO 26262 defines confirmation reviews, that must be done in order to follow the standard. The person(s) who do(es) the confirmation review (e.g. the independent safety or security assessors – see Roles in 6.3) has to fulfil certain criteria depending on the highest safety level of all safety goals:

- I0: confirmation review *should*[1] be performed and the person who performs the review must not be the author of the reviewed work product (e.g. another developer)
  - Optional, but when done, the dual control principle is being followed
- I1: confirmation review *shall*[2] be performed and the person who performs the review must not be the author of the reviewed work product (e.g. another developer) – *dual conrrol principle*
  - Mandetory and following dual control principle
- I2: confirmation review *shall* be performed and the person who performs the review must be independent of the team that created the reviewed work product (e.g. author and review must not have the same direct superior)
  - Mandetory and 1 degree of independence between reviewer and author(s)
- I3: confirmation review *shall* be performed and the person who performs the review must be independent regarding management resources and release authority from the department that created the reviewed work product (not necessarly the customer)
  - Mandetory, with complete independence of the reviewer (could be done by a separate QA department, if organizational applicable)

The SAE J3061 does not define such a strict roll separation, mainly because it does not a define a set of security levels (see above). However, it defines independence like I2 and I3. Hence, it is recommended to follow the principles of the ISO 26262 for security and assuming I2 or I3 for the security reviews.

The following table lists all confirmation reviews required by the ISO 26262 with respect to the highest safety level applicable to the safety goals of the item. Security is a separate column. Of course, safety reviews must be done by a safety expert and security reviews by a security expert.

Confirmation review of …	Degree of independency:						Description
	QM	ASIL A	ASIL B	ASIL C	ASIL D	Security	
Impact Analysis	I3	I3	I3	I3	I3	I3	New item may require new safety/security goals.
HARA / TARA	I3	I3	I3	I3	I3	I3	Safety & security goals are only fixed after this review.
Safety & Security Plan	-	I1	I1	I2	I3	I2 or I3	Depending on highest ASIL
Functional Safety Concept	-	I1	I1	I2	I3	I2 or I3	
Technical Safety Concept	-	I1	I1	I2	I3	I2 or I3	
Implementation	-	-	-	-	-	I2 or I3	Only applicable for security
Integration and Test Strategy	-	I0	I1	I2	I2	I2	
Safety Validation Specification	-	I0	I1	I2	I2	I2	

[1] "Should" means here "can", but when it its done, then the rules must be followed.

[2] "Shall" means here "must". These reviews must be done and the given rules must be followed.

Safety Analysis and Dependent failure Analysis	-	I1	I1	I2	I3	-	Only applicable for safety
System Level Vulnerability Analysis	-	-	-	-	-	I2 or I3	Only applicable for security
Safety & Security Case	-	I0	I1	I2	I3	I2 or I3	
Safety Audit / Safety Assessment	-	-	I0	I2	I3	-	Only applicable for safety

One important note: the reviews in the table are only required confirmation reviews. However, the V-model requires additional reviews for documents and especially code reviews. These reviews are to be done following I1.

The individual work produces follow below.

In the case of SEooC some of the confirmation reviews cannot be done by the company developing the SEooC. These confirmation reviews must be by the company which implements the whole item (e.g. the OEM). In a SW SEooC usually only the following work products must be reviewed:

- Safety&Security Plan,
- Implementation, and
- Integration and Test Strategy

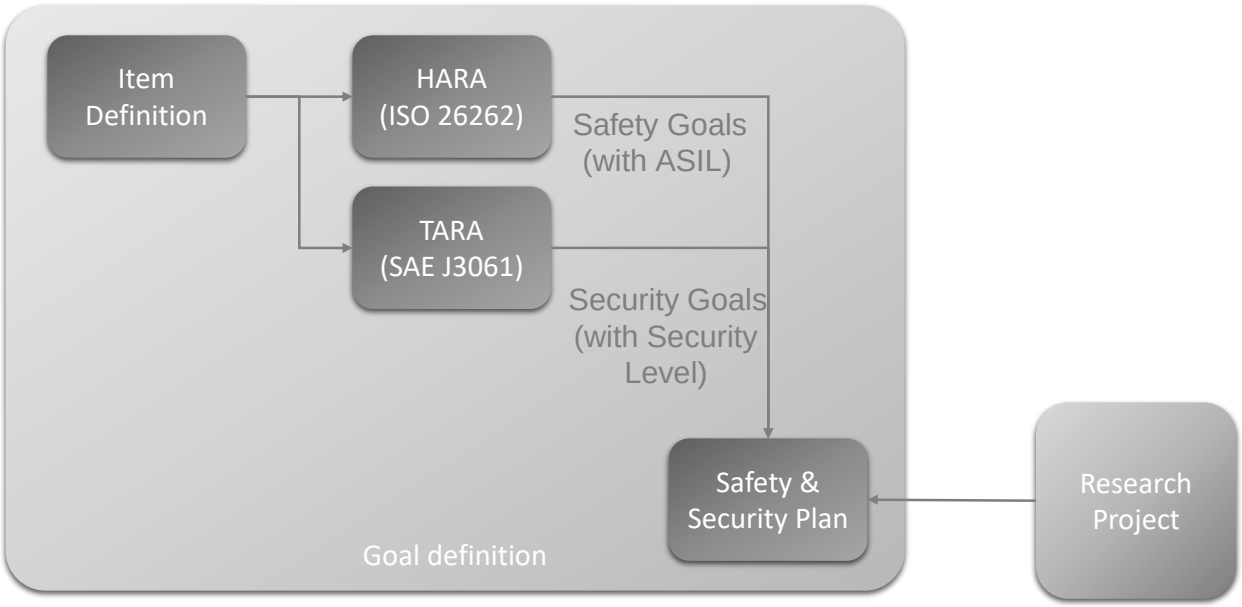
6.5 Extended V-model for Safety & Security Software

The extended V-model for Safety & Security Software follows the V-model for QM software. In an SEooC SW project most parts of the goal definition phase and of SW development (Planning) is replaced by assumptions (e.g. the HARA & TARA, functional safety & security concept, and technical safety & security concept). Compared to the QM software each phase is extended with additionally work products (e.g. safety analysis or formal reviews) and additional requirements of work products also existing in QM (e.g. unit tests).

The individual methods to choose for some of the activities depend on the safety level of the safety goal the work product applies to. In such cases the description refers to the appropriate tables in the ISO 26262. For security such relations between security level and methods are not defined in the SAE J3061. However, this document will refer to the methods listed in the SAE J3061. The more critical a security goal is the more complete and thorough these methods must be applied.

Additionally, all requirements must be categorized either as safety, security or neither (e.g. functional). All lower level requirements derived from a requirement A derive also the category of A. Hence, a function safety requirement (in the functional safety concept) must be implemented by the technical safety requirement (in the technical safety concept). It is not possible to implement a safety or security high level requirement with lower level requirements that are marked as function.

6.5.1 Extended Goal Definition



The safety & security process mostly replaces the informal goal definition of the QM process with a structured process, that starts with an item definition and goes to a function safety/security concept. All these phases have much higher requirements on qualification of the people doing them and on reviews (see 6.4.4).

The focus in this section is on SW development. But the goal definition phase must look at the whole item to derive assumptions for the SW development as SEooC. The look on the whole item continues also in the next phase (planning).

6.5.1.1 Item Definition

The item definition is the starting point of the development process. It defines the scope & elements of the item to develop on a very high level. In this context the item is a function on vehicle level (see 6.4.1). In the SAE J3061 this document is called Feature Definition. For SEooC projects, e.g. software only projects, the item definition is still useful. It might come either from the user/customer of the SEooC or on can define a generic item, that uses the SEooC component. The structure of the item definition is defined below in 6.5.1.1.1.

- Input:
  - No required input
  - Optional: the project proposal and/or documentation of the research project
- Activity:
  - Create Item Definition
- Output:
  - Item definition

6.5.1.1.1 Structure of an Item Definition

The following itemization lists the required topics, that an item definition must cover.

- General: version, status (draft or final), history, author, reviewer, release date

- Purpose of this document
  - Input for HARA & TARA
  - Define the item in the context of the vehicle
- Purpose of the item
- Function block diagram
  - Very high level (sensor(s) -> logic -> actuator(s))
- Boundary of the system responsibility and interfaces
  - Interaction of this item with other items
  - E.g. reuse of sensors and/or actuators of other items or providing a sensor/actuator also for other items
  - Based on function block diagram
  - Interface especially importation for security
  - Define external dependencies and assets
- Other sources of hazards, which influence security, safety and reliability of the item
  - Not E/E (electrical or electronics -> HW/SW)
- Function requirements
  - Define the function of the item
- Already known functional safety requirements
  - Should be derived from HARA, but (some) might be already known
- Already known cybersecurity requirements
- Other requirements
  - Possibly other environmental requirements
- Laws directives and standards
  - Which laws, (company) directives and standard have to be considered
- External measures to minimize risk
  - Which external measures can be taken to minimize the risk?

6.5.1.2 (Optional) Impact analysis

The impact analysis is an optional activity. It allows to reduce the effort for a complete V-model walkthrough when starting from an existing project. Not the Impact analysis requires all documents of a complete life cycle of a safety & security project developed according to this section or at least ISO 26262 and/or SAE J3061. For a research project the reuse-analysis is done in the beginning of the Planning phase (see below).

The goal of the impact analysis the allow minor updates of existing projects, where the update has no to big impact on the existing work products. The output is a Safety & Security Plan, with all the impacted work products derived from the Safety & Security Plan of the previous item release.

- Input:
  - Item definition, HARA, TARA, Function Safety & Security Concept of previous item release
  - Safety&security Plan of the previous item release
- Activity:
  - Does tailoring of the following phases at element and item level
    - Similar to the GAP analysis in the Planning phase
  - Does not replace the GAP analysis for research projects
    - Because GAP analysis starts with the a research project and not a previous version of an existing item
  - Requires a confirmation review of an independent safety & security assessor (see 6.4.4)
- Output:

- Safety&Security Plan (see below)

6.5.1.3 Hazard and risk analysis (HARA)

The HARA is the most important document for safety critical development. It derives the safety goals with assigned ASIL-levels in a defined and systematic way. These safety goals influence the safety activities in all follow-up phases. Hence, this document has the highest requirements from a confirmation review independent of the outcome of the HARA.

Please note, a HARA is required for any automotive project. If a HARA results in no safety goals, the follow-up phases can be done following the QM V-model from 5. For a SEooC the HARA could be done by the customer/user or safety goals incl. ASIL can be assumed, e.g. based on the generic item definition from 6.5.1.1.

Section 6.5.1.3.1 lists the required topics and the structure of a HARA. The TARA for security is defined below in 6.5.1.4.

- Input:
  - Item definition
- Activity:
  - Create HARA
- Output:
  - HARA
  - Confirmation report

6.5.1.3.1 Structure: Hazard and Risk Analyze

The following points list required topics of a HARA. More details about the HARA can be found in the ISO 26262. The HARA should be created by a team of experts, that have the formal education (e.g. Function Safety Engineer) and experience form previous safety projects.

- General: version, status (draft or final), history, author, reviewer, release date
- Participants (per Participant)
  - Name, department
  - Qualification
  - Experience
- Analysis of situations
  - Definition of possible function failures (with ids)
  - Driving scenarios
    - Description of the possible driving situation
      - E.g. from [7] and ISO 26262
    - Definition of the vehicle status
      - E.g. driving, parked, … (whatever is applicable in the driving situation)
    - Scenarios (with ids):
      - Cross product of diving situations and vehicle status (when it makes sense)
        - E.g. exclude high-speed driving on motorway (situation) and parked (vehicle status)
  - Analysis
    - Cross product of scenarios and possible function failures
    - For each element determine severity (S), exposure (E) and controllability (C)
      - E.g. with [7] for E
      - E.g. with [8] for S
        - S0: AIS 0 and <= 10% of AIS 1-6
        - S1: > 10% of AIS 1-6 (but not S2 or S3)

        - S2: > 10% of AIS 3-6 (but not S3)
        - S3: > 10% of AIS 5-6
    - E.g. with ISO 26262-3 Table B.6 for C
- Derive ASIL from parameters (see ISO 26262)
- For each failure determine worst case ASIL -> ASIL of this failure

	Scenario 1	Scenario 2	Scenario 3	Worst Case	ASIL
Failure 1	S E C	S E C	S E C		
Failure 2	S E C	S E C	S E C		
Failure 3	S E C	S E C	S E C		

- Result: Safety goals with ASIL
    - For each failure
        - Failure
        - Safety goal (i.e. avoid the failure)
        - ASIL level
        - Safe state
        - Fault tolerance time

6.5.1.4 Threat and risk analysis (TARA)

The TARA is the security equivalent to the HARA. It helps to derive the security goals in a defined and systematic way. Its results also greatly influence the following phases. Hence, it requires a confirmation review too and need to be done by a team of experts with sufficient education and experience. For a SEooC the TARA could be done by the customer/user or the security goal can be assumed, e.g. based on the generic item definition from 6.5.1.1.

In contrast to the HARA the SAE J3061 does not define one process of deriving the TARA. Also, the security level assigned to the security goals, are never used in the SAE J3061 document. It is up to the developers to make use of the security level information in the following phases, i.e. by tuning the degree to which security activities are done.

In this document we recommend to use the HEAVENS Security Model from [3] for the TARA (see 6.5.1.4.1).

- Input:
    - Item definition
- Activity:
    - Create TARA
- Output:
    - TARA
    - Confirmation report

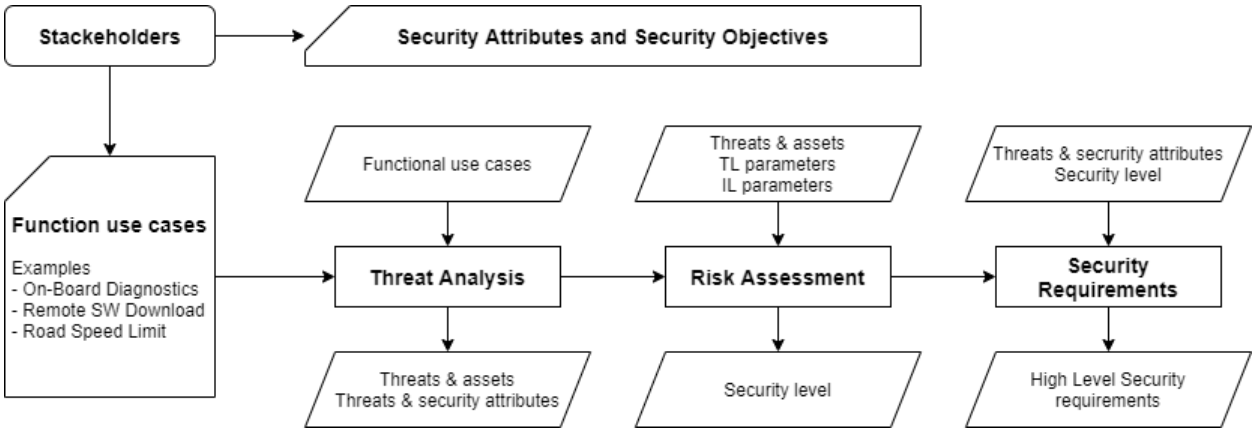
6.5.1.4.1 HEAVENS Security Model

The SAE J3061 standard [3] recommends several methods for deriving the security goals for the item to develop. The reason is there is no single TARA method currently well established in the automotive industry. Of the recommended methods of [3] the HEAVENS Security Model has three advantages over the other methods presented in [3]:

- It is described in most detail,
- it is the only methods that was already tested with an automotive project [17], and
- it is aligned with the Iso 26262 [2]

However – like the other proposed methods from [3] – the HEAVENS Security Model is not yet well established in the automotive industry.

In this document we only give a very high-level overview of the HEAVENS Security Model. See [3] Appendix A 1.5 for a detailed description of the method.



The figure above (based on Figure 23 of [3]) illustrates the HEAVENS workflow. The HEAVENS TARA starts with the Threat Analysis. The Threat Analysis derives from the Function uses cases (which come from the item definition) the assets and security attributes. The assets are the elements in the high-level architecture of the item definition. However, security experts might identify additional assets (e.g. cryptographic keys) which are not elements itself, but parts of an element. The security attributes are derived with the STRIDE method from Microsoft (see Table 9 of [3]).

The risk assessment derives security levels of for each threat & asset pair from the Threat analysis. Therefore, the Threat Level (TL) parameters and the Impact Level (IL) Parameters must be evaluated.

TL parameters:

- Expertise: level of generic knowledge required to carry out an attack
- Knowledge: availability of information and the community size from the perspective of an attacker
- Equipment: required equipment to identify or exploit a vulnerability and/or mount an attack
- Window of opportunity: combination of access type (e.g. physical, logical) and access duration (e.g. limited time or unlimited time) require to mount an attack

Each TL parameter is ranked between 0 and 3. Then the sum of all TL parameters is mapped to a single TL value between 0 and 4 (Table 11 of [3]).

IL parameters:

- Safety: severity level from the ISO 26262

- Financial: direct and indirect considers financial losses or damages
- Operational: operational damages caused by unwanted and unexpected changes of in a vehicle function
- Privacy and legislation: damages caused by privavy violation of stakeholders and/or violations of legislations/regulations

See A 1.5.1.2 of [3] for how to derive the IL parameters. The same chapter defines how to combine all IL parameters into a single IL value between 0 and 4 in Table 16. Then Table 17 defines the derivation of the Security Level: QM, Low, Medium, High, and Critical.

Like for safety a QM level means that no special security process needs to be followed. Because the SAE J3061 does nowhere map the security level to recommended methods (e.g. for design methods or testing methods) common sense and security experience should be applied by selecting the methods and the extend of how to apply them from SAE J3061 based on the derived security level.

Security levels are derived per threat. The security goals are then to avoid the given threat with the given security level.

The derivation of the Security Requirements (the last step of the HEAVENS workflow) is described below together with the derivation of the function safety concept.

6.5.1.5 Start Safety & Security Plan

If both HARA & TARA only lead to QM for **all** their safety/security goals, then the safety & security process can be stopped here and the QM process can be followed.

- Input:
  - Research project (publications, documentation, source code, …)
  - Safety&Security Plan from Impact analysis (if any)
  - This document
- Activity:
  - Checklist with all planned activities of the next phases
  - List work products to create
    - Documents, designs, analyses, source code, reviews, reports, …
      - Derived from this document
    - With authors, reviewers
      - Either specify by role or people directly
      - Ensure qualification and experience of people writing & reviewing the document
      - Author != reviewer
  - List Verifications, validations to perform
    - derived this document
    - Performed by whom
  - Do GAP-Analysis
    - Check each work product in safety plan
      - What can be taken from the research project?
      - Can it be reused without rework?
      - Adapt safety plan accordingly
  - Each activity should be mirrored by tickets that track the progress per document
    - One ticket per document version
  - Continuously updated
    - Check completed activities
    - Add new activities

  - Format: Jira Tickets
- Output:
  - Safety & Security Plan

The Safety and Security Plan is the is list of all activities required to fulfil the safety & security life cycle of the development of an item. Hence, it contains all activities defined in this chapter.

Best practice is to create for each activity one ticket in a ticket management system (see also 7.3 and 8.1.2.1). The safety & security plan is then just a snapshot of all the tickets that define the plan together with their state and current assignee. This snapshot can be automatically generated by the ticket management system if all tickets have a common label (e.g. “safetysecurityplan”). Best practice is to use ticket templates to ensure a common structure for all tickets of the safety & security plan.

The Safety & Security Plan must consider the defined safety and security levels, e.g. if the safety level is QM then no special safety activities are required and only security activities must be selected for the Safety & Security Plan. In this phase the plan might not yet contain all activities. It should at least contain all activities of the next phase and planning activities for all other phases.

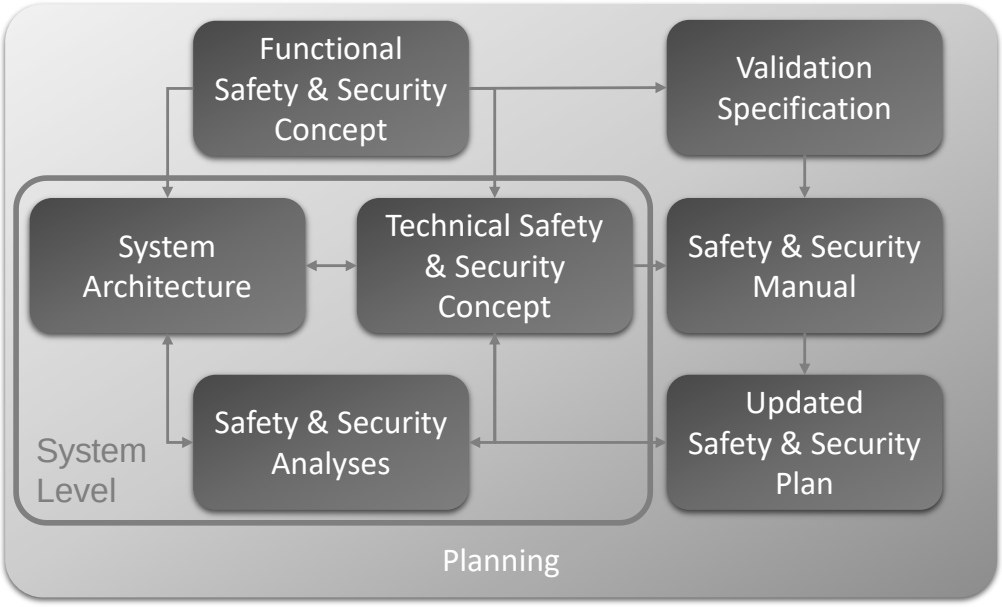
The plan will be updated in the following phases as some activities may trigger new activities not present in the plan yet.

A GAP-analysis helps to reuse artifacts of the research project, this project builds on. Hence, where possible the tickets refer to already existing work products that might be (partly) reused for the planned activity. However, the doing the activity must ensure, that the work product reaches the safety & security levels required by the HARA and TARA.

Whenever new activities are added to this plan in later phases, the GAP-analysis with the research project, helps to identify reusability for the development of the item.

If an impact analysis was done and because of the impact analysis HARA & TARA were skipped the safety & security plan of the impact analysis can be used in this phase without adaptation. However, if a HARA & TARA were done and yield different safety and security goals and/or safety and security levels for these goals. The safety & security plan from the impact analysis must be updated with the results from the HARA & TARA.

6.5.2 Extended Planning Phase

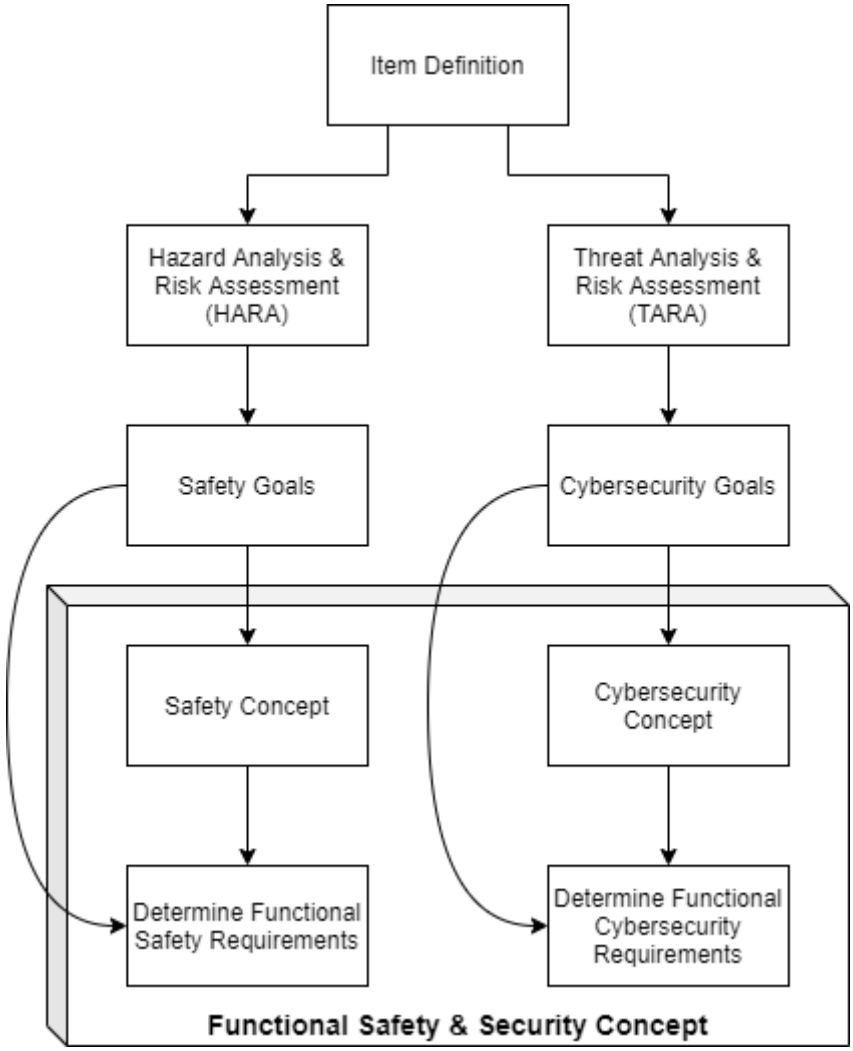


The project plan from 5.2 is replace by the safety and security plan (see above). However, it must be extended by all QM activities not already covered by the safety & security plan, i.e. if the HARA does not yield any safety goal. The test plan is replaced by the validation and verification specifications below (6.5.2.4 and the SW Integration Test Specification from 6.5.4). The release checklist is replaced by the continuously updated safety & security plan (e.g. the activity tickets in the ticket management system), the safety case and the confirmation measure report for them.

6.5.2.1 Functional Safety & Security Concept

The Functional Safety & Security Concept defines how to achieve the safety and security goals derived from the HARA & TARA. The result is a high-level concept with the safety and security requirements. This concept and the requirements should not define a technical solution, just the plain requirements that must be fulfilled by the technical solution derived below.

The following figure shows the high-level workflow from the item definition to the functional safety & security concept.



- Input:
  - Item definition
  - HARA
  - TARA
- Activities:
  - concept to achieve safety goals
    - set of requirements marked as function safety requirements
    - + additional documentation (e.g. publications if usable)
    - derivation of parameters
      - safe state
      - Fault Tolerance Time Interval
      - …
  - For Security Requirements follow the HEAVENS workflow (see A 1.5.1.3 of [3])
  - For a SEooC: concept can be an assumption
  - requires a confirmation review of an independent assessor
    - depending on the ASIL level
- Output:
  - Function Safety Concept
    - Incl. Function Safety Requirements

- Cyber Security Concept
  - Incl. Security Requirements
- Confirmation Report

6.5.2.1.1 Structure of the Functional Safety & Security Concept

The functional safety & security concept is derived from the Item definition with its preliminary item architecture (on element level), from the HARA with its safety goals and ASIL levels and from the TARA with its identified threats, assets & security levels per threat & asset.

Attributes of Safety Goals

For each safety goal write down the following attributes / parameters of the safety goal (e.g. as table):

- Integrity
  - Methods, technics, etc to ensure that the safety goal upholds
- Safe state
- Fault tolerance time
- Warning concept
- Degradation concept

Functional Safety requirements and allocation

For each safety goal write down all requirements of this safety goal. Allocate each requirement to an item (e.g. if they shall be implemented by another item), or element of this item. If possible, already specify components that shall implement this requirement.

Security Requirements

The derivation of the security requirements follows the HEAVENS workflow (see A 1.5.1.3 of [3]). For each asset each threat must be avoided with the derived security level.

6.5.2.2 For SEooC: Start Safety & Security Manual

- Input:
  - Either:
    - Item definition
    - HARA & TARA
    - Functional Safety & Security Concept
  - Or:
    - Assumed Safety & Security Requirements to be (partially) provided by the SEooC
- Activity:
  - Start with the Safety & Security Manual (see below)
- Output:
  - 1st draft of Safety & Security Manual
    - Will be refined in further phases
  - Update of Safety & Security Plan to refine the Safety & Security Manual

The Safety & Security Manual is for the customer or user of the SW-component. It helps the customer/user to put the SW-component into an item. Hence, the safety & security manual must contain all assumptions

that must be fulfilled to provide the safety and security goals for the item. The customer/user must check all assumptions of the Safety & Security Manual against the developed item. The following two questions must be answered:

- Does the SEooC implement all requirements expected by the customer/user?
- Does the item developed by the customer/user fulllfil all assumptions required by the developer of the SEooC?

Therefore, the Safety & Security Manual must contain the assumed system architecture (see below) with all assumptions on all components surrounding the SEooC. These assumptions can be formulated as requirements to these components (outside of the SEooC). Additionally, all requirements (functional, safety & security) implemented by the SEooC must be documented in the Safety & Security Manual.

Usually it is enough to list work products of the safety & security process, including the Item definition, HARA, TARA, function safety concept and so on with the applicable version numbers and append the relevant documents or give the customer/user access to them. When some of the documents are not available, because they were not created, the assumptions used in their place must be documented in the Safety & Security Manual.

Also, all the phases/activities that cannot be done by the creator of the SEooC, must be included as an TODO for the customer/user of the SEooC.

The manual must be extended throughout the safety & security process. In this first creation phase only the currently available documents need to be included. Further phases must be included into the Safety & Security Plan. When updating the Safety & Security Plan, consider documentation from the research project that might reusable for the Safety & Security Manual.

6.5.2.3 Planning of safety & security activities for system development (System Phase)

- Input:
  - Current Safety & Security Plan
  - This document
- Activity:
  - Update Safety&Security Plan
    - Plan all the activities to create the following documents in this section
    - Especially add validation plan to Safety & Security plan and Safety & Security Manual (see 6.5.2.4).
    - Checklist with responsibilities for validation & verification actions
- Output:
  - Updated Safety & Security Plan

As already discussed, the Safety & Security Plan requires constant updates. First of it must be refined to include the activities listed in this section. And second it also requires updates regarding the status of the already created documents. Ideally, these status updates happened in parallel to the document creation. At the beginning of a new phase it is a good time to review the Safety & Security Plan to ensure:

- All reusable artifacts from the research project have been considered
- All documents/activities required as input for the next phases are finished and final
- All follow-up activities are fully planned, i.e.
  - Well described
  - Assigned to specific people for doing and for review

6.5.2.4 Safety & Security Validation Specification

- Input:
  - Function Safety concept
  - Cybersecurity Concept
- Activity:
  - Creation of Safety & Security Validation Specification
    - Validation = Tests in vehicle
    - Plan for Phase "Verification of SW Requirements"
      - List of testing methods to apply with Focus on functional requirements
      - For safety: ASIL dependent (see ISO 26262 for recommended methods)
        - Restricted to validation measures on vehicle level
      - For security:
        - See SAE J3061 8.6.9 and 8.6.10 for recommended measures
      - For SEooC: only focus on SW-component -> must be part of safety manual
    - Note: full Validation for a SEooC-SW-component alone is not possible
      - Validation always requires in-vehicle testing
      - Must be done by customer/user in the context of the item using the SW-component
      - Must be left as a TODO in the Safety & Security Manual for the customer/user
  - requires a confirmation review of an independent assessor
    - depending on the ASIL level
- Output:
  - Either:
    - Safety & Security Validation Specification
  - Or (for SEooC only):
    - Updated Safety & Security Manual to include creation of Security & Safety Validation Specification as TODO for the customer/user

The Safety & Security Validation Specification is the test specification for the in-vehicle tests of the developed item. For the required test measures see ISO 26262 and SAE J3061. The selection of the test measures depends on the derived (or for a SEooC assumed) safety and security levels. All validation tests must be traceable to either a Functional Safety or Security Requirement. The other way around each functional safety and security requirement must be covered by at least one test of the Safety & Security Validation Specification.

Typically, these validation tests are manual tests.

The tests of the Safety & Security Validation Specification can only be specified on the vehicle level for the whole item. Hence, for an SEooC these tests cannot be specified, because a SEooC is only a (sub-) component of an item. Therefore, these tests must be done by the item creator (i.e. the customer/user of the SEooC). The Safety & Security Manual must be updated accordingly for a SEooC.

6.5.2.5 System architectural design specification

The system architectural design specification is composed of five documents:

- Overall item Architecture at-least down to the element level
- Technical safety concept
- Safety Analysis and the dependent failure analysis (DFA)
- Technical security concept
- System Level Vulnerability Analysis

- Hardware-Software-Interface Definition

Additionally, the following documents must be updated:

- Safety & Security Plan
- Safety & Security Manual (for a SEooC only)

- Input:
  - Item definition
  - Function Safety Concept
  - Cybersecurity Concept
- Activities:
  - System Architecture Design Specification
    - Defines item architecture on element level
      - E.g. sensor(s), ecu(s), actuators(s)
      - Document with appropriate UML diagrams
    - Defines HW & SW
    - For SEooC: SW-component must be either one of the elements of the architecture or a (sub-)component of an element of the architecture
      - SW-component implements just some (or one) of these element/component
      - Assumptions for other elements must be documented
        - Assumptions become part of Safety & Security Manual
    - Replaces HW definition from QM
    - Also include function parts of architecture
      - Focus not only on safety & security for this document
  - Safety analysis and the dependent failure analysis (DFA)
    - Inductive Analysis (usually FMEA [10] and see 11.1)
    - From ASIL-B: also deductive analysis (usually FTA [11])
    - For SEooC might be to TODO for the customer/user of the SW-component
      - TODO must be part of Safety & Security Manual
    - Requires a confirmation review of an independent assessor
      - Depending on the ASIL level
    - Most likely updated in “Verification of SW Requirements”
  - Technical safety concept
    - Derived from functional safety concept and safety goals (incl. ASIL)
      - Developed in combination with
        - System architectural design specification (see above)
        - Technical security concept
    - For SEooC: Becomes part of Safety & Security Manual
      - Usually just contain the assumptions about the surrounding components and the requirements to be implemented by the SEooC
    - Technical safety requirements
      - ASIL X related measures & safety mechanism to achieve safety goals
      - Specification of environmental requirements
      - System analysis
      - Specification of requirements for other systems and interfaces
      - For SeooC: For other elements define necessary assumptions
      - Derived from functional safety concept
    - Requires a confirmation review of an independent assessor
      - depending on the ASIL level
  - System Level Vulnerability Analysis
    - Identify vulnerabilities on the item level

        - See SAE J3061 8.4.2
        - Method: e.g. with Attack Trees (see A 1.6 of [3])
        - Requires a confirmation review of an independent assessor
    - Technical Security Concept
        - Derived from cybersecurity concept and security goals (incl. security levels)
            - Developed in combination with
                - System architectural design specification (see above)
                - Technical safety concept
        - Takes result of System Level Vulnerability Analysis into account
        - For SEooC: Becomes part of Safety & Security Manual
            - Usually just contain the assumptions about the surrounding components and the requirements to be implemented by the SEooC
        - Technical security requirements
            - Measures & security mechanism to achieve security goals
            - Specification of environmental requirements
            - System analysis
            - Specification of requirements for other systems and interfaces
            - For SEooC: For other elements define necessary assumptions
            - Derived from functional security concept
        - Requires a confirmation review of an independent assessor
    - Hardware-Software-Interface (HSI) specification
        - Contains the concrete requirements that the SW requires the HW to implement
            - E.g. writing 0 to register XYZ provides disables output port abs
        - Contains all the requirements of the HW to the SW
            - E.g. the SW has to check on start-up that the content of memory address abc is XYZ
            - Or HW must contain a tamper-proof key or an encryption component
        - specifies diagnostics that SW needs to run on HW
            - Cyclic Diagnostic tests
            - Diagnostics during power-up
            - See ISO 26262-5 Annex D
        - Needs to be updated in the SW development process
        - For SEooC: becomes part of the Safety & Security Manual
- Output:
    - System Architecture Design Sepcification
    - Technical safety concept
    - Safety analysis and the dependent failure analysis (DFA)
    - Technical Security Concept
    - System Level Vulnerability Analysis
    - Hardware-Software-Interface (HSI) specification
    - For SEooC- update Safety & Security Manual:
        - Documents are done partly and must be part of Safety & Security Manual
        - Analysis may be documented as TODO for customer/user

The five documents created during this phase must be created together. They must not contradict each other. Therefore, when doing the reviews of the security concept also a safety expert must be included and vice versa. The system architecture must include all elements and if possible, components of the item.

The safety analysis and the security analysis (vulnerability) must be done on item level based on the system architecture. The resulting issues must be addressed by the technical safety concept and the technical security concept, respectively. In practice that means the analysis and concept must be created side by side but in independent documents (for clarity).

The technical safety concept must specify how to implement the functional safety concept, i.e. each functional safety requirements must be traceable to at-least one technical safety requirement. The same applies for the function security concept and the technical security concept. Traceability includes the derivation of the safety and security properties of the functional requirements down to the technical requirements. Both technical concepts refer to the defined system architecture.

Because software and hardware are implemented mostly independent of each other, the HSI is the central document to communicate, document and check requirements of HW on SW and the other way around. These requirements might be the result from the two technical concepts for safety and security, but also from the functional (QM) requirements. The HSI must be updated in the following phases.

For SEooC: the system architecture can focus on the surroundings of the component to implement. Hence, the two technical concepts (for safety & security) will only contain the (assumed) requirements to be implemented by the SEooC and the (assumed) requirements the SEooC has on the surrounding components in order to work correctly (and provide the required safety and security requirements). All these assumed requirements must be part of the Safety & Security Manual. The analyses may make no sense for a SEooC. In this case they need to be done by the customer/user (TODO of analysis must be part of Safety & Security Manual).

6.5.3 Extended SW Requirements

This phase marks the beginning of the SW development phase. Hence, beginning with this sub-sub-section the focus is on software only. The hardware development process is out of scope of this document. Nevertheless, the Hardware-Software-Interface (HSI) specification must be considered in this phase, because all requirements from the hardware on the software need to be implemented by the software. The HSI can be updated in the SW development phase, either if new requirements on the HW emerge or existing requirements need to be refined. The same can happen the other way around. Any change of the HSI must be traced and communicated to SW and HW development teams. Both teams must then update their requirements and iterate over all the following phases to ensure, that the updated requirements are implemented and verified.

Best practice is to update the Safety & Security Plan always together with the HSI. So, any change to the HSI is reflected by adding activities to follow up these changes to ensure they are all accounted for in the implementation.

This phase is done separately for each SW component of an item. For an SEooC this phase is the same as for the development of a complete item.

- Input:
  - Item definition
  - System Architecture Design Specification
  - Technical Safety Concept
  - Technical Security Concept
  - Hardware-Software-Interface (HSI) specification
- Activities:
  - Software Requirement specification
    - Derive SW-only requirements from input
    - As always mark safety and security requirements
  - Update Safety & Security Plan
    - Plan the activities of the software development
  - Create SW Verification report
    - Document that contains (links to) all verification reports (reviews, test reports) of the SW development

- Links can be
  - Document with version (for review documents in Word)
  - But also ticket numbers, if code-reviews and builds are attached to tickets
- Is filled in the SW development process
- Can be part of Safety & Security Plan

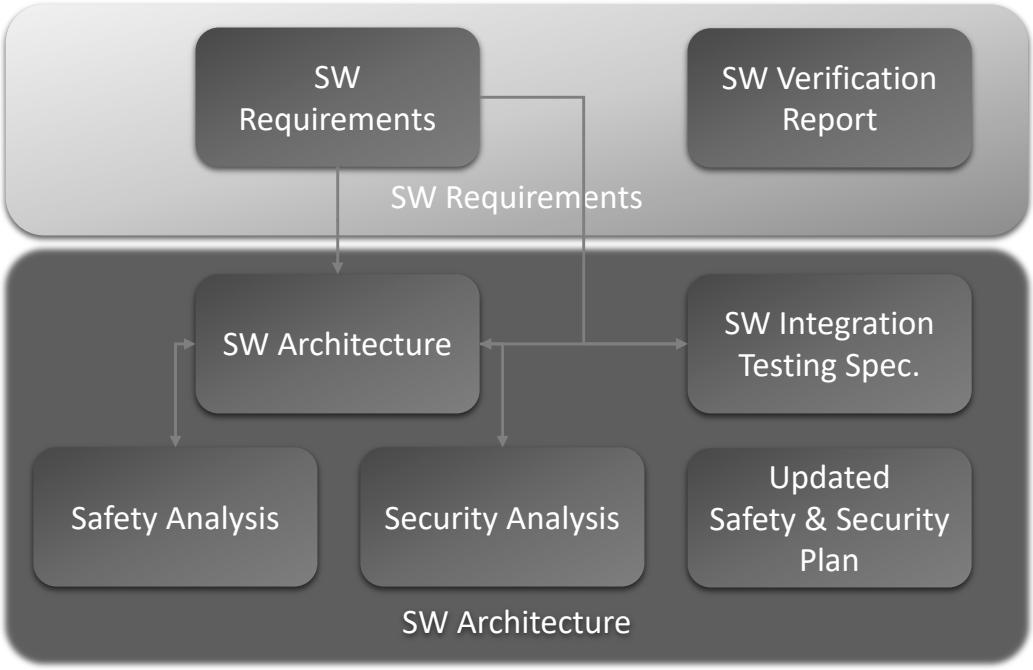
- Output:
  - SW requirements
    - Function requirements as before for the SW-component
      - Now derived from item definition and "System architectural design specification"
    - Safety requirements for SW-component
      - Especially marked with "safety"
      - Derived from technical safety concept
      - Link these requirement back to requirements of technical safety concept
    - Security requirements for SW-component
      - Especially marked with "security"
      - Derived from technical security concept
      - Link these requirement back to requirements of technical security concept
    - Requires review depending on the safety level & security level
      - Review methods depend on maximum ASIL (see ISO 26262-8, Table 2)
        - Best practice independent reviewer (inspection)
          - Also for security SW requirements
      - Review criteria:
        - Appropriately grouped together
        - Completeness
        - Consistent
        - No duplication
        - Maintainability
        - Testable/Measurable
        - Suitable for SW development
        - Compliance and consistency with technical safety concept
          - Incl. HW-SW-interface
        - Compliance with architecture on element level
  - Updated Safety & Security Plan for SW development
  - SW Verification report
    - Contains only the review of the SW requirements

In this phase all software requirements for all SW components to be developed must be specified. These requirements are derived from the inputs of this phase. Safety and security requirements must be marked and traced back to the technical safety & security concepts. Special care must be taken for the HSI, because the HSI is the communication interface between HW and SW development. As described above, changes to the HSI must be traced and lead to a (lightweight) iteration of the SW development phase. This should be triggered by updating the Safety & Security Plan.

Independent of the HSI, because this phase marks the start of the SW development, the Safety & Security Plan should be updated to include all activities specific for the SW development phase.

Throughout the SW development phase, the SW Verification report contains all the verification and test reports of the SW development phase. Because all these verification and testing activities need to be planned in the Safety & Security Plan, it is good practice to include or link these reports to the planned activities (tickets). Hence, the SW Verification Report is just a view on the Safety & Security Plan, that

6.5.4 Extended SW Architecture design



- Input:
  - Item definition
  - System Architecture Design Specification
  - Technical Safety Concept
  - Technical Security Concept
  - HIS
  - SW Requirements
- Activities:
  - Design SW architecture similar to 5.4
    - Refines system architecture design specification
      - Static design aspects: class or block diagrams
      - Dynamic design aspects: state charts, timing or scheduling diagrams, sequence diagrams
      - Design goals for safety:
        - Comprehensibility
        - Consistency
        - Simplicity
        - Verifiability
        - Modularity
        - Abstraction
        - Encapsulation
        - Maintainability
        - See ISO 26262-6 Table 3
      - Additional design goals for security:
        - Confidentiality
        - Integrity
        - Availability
    - Architecture must contain safety, security and function

- Every element in a diagram requires a unique id
- Required for FMEA (see below)

- Mark SW-(sub)-components and units with safety & security requirements for SW requirements
- For every SW-component/unit it must be clear for which ASIL level & security level it is developed
- Without any methods to avoid interference all SW-components that share a hardware also share their maximum ASIL level/security level
  - Interference can be avoided with a suitable real time OS with strong safety & security guarantees
    - Then: SW components that run in different VMs/processes can have different ASIL-Levels and be even QM
    - Note: OS requires a safety qualification/certification -> COTS Linux is not sufficient
    - Note: for security such an OS may be required to establish trust boundaries
- Requires review based on ASIL level & security level

- Safety Analysis of SW Architecture
  - Focus is on Freedom from Interference
    - Timing (deadlocks, synchronization, scheduling errors, …)
      - Best practice: safety real-time OS
    - Memory (corruption, stack overflow/underflow, corruption of other SW components)
      - Best practice: safety OS
      - All SW components of one ASIL can run in the same partition
    - Information exchange (repetition, loss of information, delay, insertion, corruption, …)
      - Best practice: black channel protocol with
        - CRC
        - Message counter
        - Addressing
        - Watchdog
  - Form: SW-FMEA in Excel Details in 11.1
  - Analysis usually leads to new requirements for error detection and handling
  - Requires review based on ASIL level
    - See ISO 26262-8
- Software Vulnerability Analysis
  - SW Architecture defines trust boundaries based to be analyzied
    - 1. Step: Analyse data that crosses these trust boundaries with attack trees to do threat modeling (see SAE J3061 A 1.7)
    - 2. Step: threat categorization based on methods like STRIDE, ASF, or DREAD
      - Note: STRIDE is also part of the HEAVENS workflow of the TARA
    - 3. Step: Identify areas with the highest risk and plan cybersecurity controls to
      - reduce or mitgate this risk,
      - accept the risk,
      - disclose the risk (i.e. to the end-user)
      - terminate the risk (i.e. change functional behavior)
    - Note: some security measures can replace safety measures, e.g. cypthographic authentification can replace black channel authentification and CRC
  - Requires review

    - Specification for "SW integration testing"
        - List of testing methods to apply
            - For safety: see ISO 26262-6 table 8
            - For security: see SAE J3061 8.6.6
            - Requirement based testing
            - Interface testing
            - Resource usage testing
            - From ASIL-C on: fault injection testing
        - Plan tests derived from SW architecture, requirements, safety analysis and security analysis
            - All test-specifications must be linked to the SW requirements they test
            - See ISO 26262-6 table 11 for methods for deriving integration test cases
            - See SAE J3061 A.2 for methods for deriving integration test cases
        - Test specification as tickets if possible
        - Specify required code coverage for tests
            - From ASIL-C on: function coverage & call coverage
                - E.g. all functions are called at least once and all function calls are executed at least once
                    - Can be measured with statement coverage
        - Note: static code analysis will be done on unit level
        - Test on embedded target HW
        - requires a confirmation review of an independent assessor
            - depending on the ASIL level
- Output:
    - SW Architecture
    - Safety Analysis
    - Vulnerability Analysis
    - Specification for "SW integration testing"
    - Updates to SW Verification Report
        - Contains review reports from this phase

In this phase the SW Architecture must be defined and analyzed. The design criteria for safety and security a listed above. The safety and security (vulnerability) analyses should be done by individual experts. Findings of these analyses might require updates of the SW Architecture. Based on the final SW Architecture the SW integration tests must be specified. Safety and security have different requirements on the SW integration tests. Hence, on integration level usually different tests must be done for safety and security.

All the documents created in this phase: SW Architecture, safety and security analyses and the integration test specification) require reviews. These reviews become parts of the SW Verification Report.

6.5.5 Extended SW Unit Design & Implementation

- Inputs:
    - SW Requirements
    - HSI
    - SW Architecture design
- Activities:
    - Develop SW Unit design
        - Notation for design in natural language (in ticket)
            - From ASIL-C on: + UML diagrams, Simulink or Stateflow models
        - Design principles:
            - For safety: see ISO 26262-6, table 8

        - For security: SAE J3061 8.6.5
    - Implementation must follow a some coding guidelines
        - E.g. MISRA C – see also ISO 26262-6, table 8
        - For Security: CERT C
- Outputs:
    - SW Unit Design
    - SW Unit Implementation

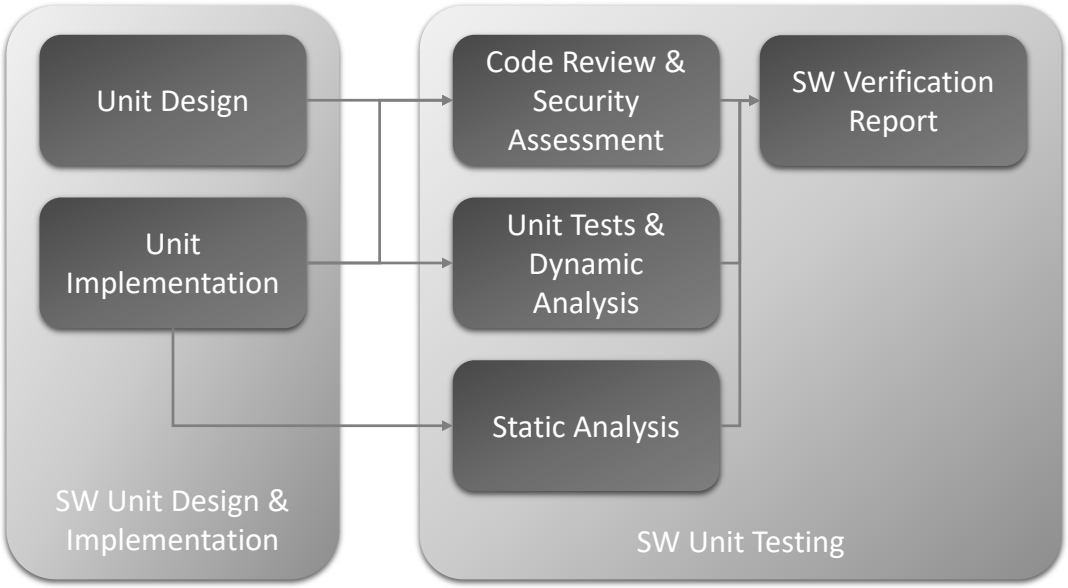
In this phase the SW Units must be designed (i.e. specified) and implemented. The design can be done in tickets and the implementation in source code must be linked to these tickets (i.e. by linking each commit to one ticket). Many of the design principles for safety and security overlap on the unit level (e.g. input validation, range checking, ban of unsecure string C API).

It is recommended to do the implementation on the overlapping subset of MISRA C (for safety) and CERT C (for security). Hence, the static code analysis should check for both standards in the next phase.

It is best practice to implement the unit in a separate (git) branch. This enables code reviews as pull-request (see next phase).

Note: The result of this phase is the complete source code, not necessarily any binary code. This will be generated in a later phase.

6.5.6 Extended SW Unit Testing



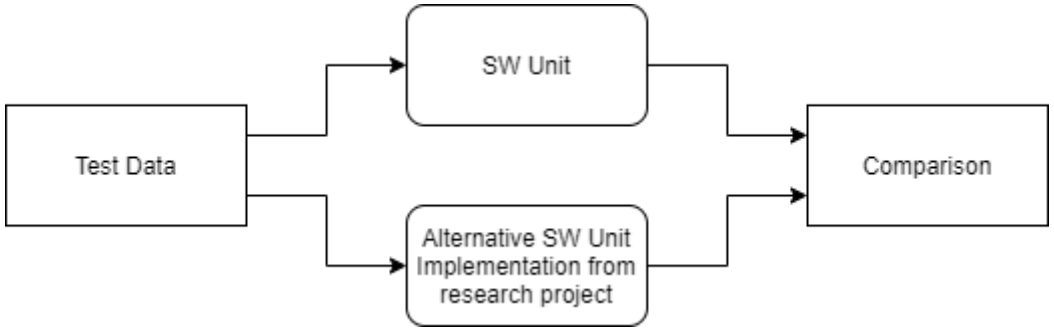
- Inputs:
    - SW Requirements
    - HSI
    - SW Architecture design
    - SW Unit Design
    - SW Unit Implementation
- Activities:
    - Static analysis:
        - Automatically check coding guidelines (e.g. a MISRA C checker)
        - Measure code complexity
            - Define useful upper bounds

        - Compile for the target platform
    - Code Reviews:
        - Manual code review
        - Required for both safety and security
        - Check SW Unit Implementation
            - for design guidelines
            - consistency with its design and SW requirements
    - SW Unit Testing:
        - Black-box-Testing
            - Tests are derived from SW requirements that are (partly) implemented by this unit
                - tests must be linked to SW requirements
                - For safety: methods see ISO 26262-6, Table 8
                - For security: methods see SEA J3061 8.6.7
        - White box-testing
            - Interface tests
            - For ASIL-D: fault injection tests & resource usage tests
        - Measure test coverage
            - required coverage metric: ISO 26262-6 Table 9
            - add justification for uncovered code paths
        - Combine with dynamic analysis
            - E.g. use Valgrind to scan for memory or concurrency errors
            - E.g. use clang/gcc sanitizer to scan for undefined behavior
- Outputs:
    - SW Unit Verification specification
        - Test case specifications
        - Code review objectives (with check lists)
        - Static Analysis configuration
    - SW Unit Test Implementation
    - Extended SW Verification Report
        - Include test reports
            - Incl. reports from static analysis
            - Incl. justification for failed static analysis checks and violations of complexity metrics
        - Review logs
    - SW Unit Binaries for the target platform
    - Cybersecurity Assessment

The SW Unit Testing includes three integral parts: static analysis, code reviews and unit testing. The SW Unit Verification specification must specify all test cases (functional, safety and security) including the required code coverage and dynamic analysis configuration, the code review objectives (i.e. a check list to apply for each review) and the configuration for the static analysis tools. Note: it is best practice to centrally define the generic requirements on the SW Unit Tests (i.e. required code review check lists, configuration of static analysis tools and dynamic analysis tools). The unit test code review checklists must also contain checking the test reports of the CI. The individual unit test specifications should be done in a sub-ticket of the unit ticket. The test implementation should also be done on the same ticket and be code reviewed to ensure, that the test follows its specification.

The test implementation should be based on a unit test framework (e.g. GoogleTest [18]) that produces a test report in the JUnit format. Each unit test must be linked to the SW requirement tested by that test. GoogleTest can be extended to include these links into the test report.

The test implementation can happen in parallel to the SW Unit implementation. However, the test implementation on this level requires a SW Unit Design before it can be started. Another approach is to develop the SW Unit together with the SW Unit test in test driven development [6].



If the architecture between the research project and the final SW is similar and the interfaces of all or at-least some SW Units match between research project and software development project, then it might make sense to do back-to-back tests. For back-to-back tests two SW Units (i.e. the SW Unit to test and an alternative implementation from the research project) are given the same input data and the outputs are compared. Of course, that can only be done, if the SW Unit is not derived from the SW Unit of the research project, but independently developed (ideally in a different programming language and be a different development team).

A CI system should run the static analysis tool(s) and the unit tests. The static analysis tool should check the coding guidelines (MISRA C and/or CERT C) and measure the code complexity. The unit tests should also be run by the CI. Optionally, dynamic analysis can be enabled for running the unit tests, e.g. with Valgrind or clang's sanitizers. Additionally, the code coverage must be measured. The resulting reports should be linked from the tickets, where the unit is designed (e.g. via the CI).

Related code reviews should also be linked to the tickets. Best practice is to use pull-requests for manual code reviews [19]. These pull-requests should be linked to the tickets. It is also the task of the (code) reviewer, to check whether the reports (from the CI) for violations (must be part of the check lists for code review). Each violation must be individually argued, why it is safe and secure or be removed. Static analysis tools usually provide support for defining individual exceptions. Only, if all violations found in code review are either removed or successfully argued, the code review may approve the pull-request for merging.

In summary the unit design tickets link to:

- The implementation (e.g. the pull-request and the individual tickets)
- Test specification ticket; which links to
  - Test implementation
  - Code review of test implementation
- CI reports; which include
  - Static analysis reports
  - Unit Test Reports (optionally including dynamic analysis reports)
- Code review

In practice one should plan for at least 2-3 iterations between Unit Implementation and Unit Testing until the pull-request can be approved. The implementer can always update the pull-request with additional commits to remove violations from the code. At the end, the code reviewer approves the final state of the pull-request with all changes done in these iterations.

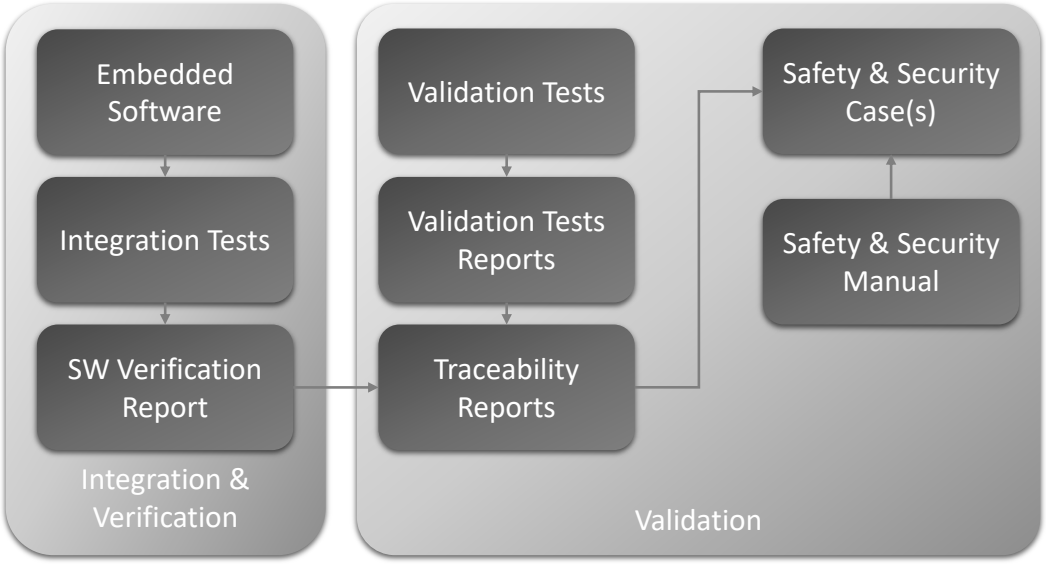
CI reports (all test reports plus reports from dynamic and static analysis tools) and Code review (reports) must be part of the SW Verification Report. When all reports are appropriately linked to the implementation tickets and the implementation tickets are appropriately labeled, this should happen automatically.

In practice the dynamic analysis can be done either on the target platform or on a development platform. Some tools (e.g. Valgrind and gcc/clang sanitizer) are not supported on all embedded target platforms. In this case, it makes sense to compile the SW Unit twice:

- Once for running the unit tests (incl. dynamic analysis and code coverage measurements) on the development platform.
  - Even when testing on the target platform, it makes sense to have a second compile software unit for testing, i.e. including dynamic analysis and debug information.
- Once for the target platform with the defined settings for the target hardware.
  - Note: The compiler settings can be influence by the HW via the HSI.

Only the binaries for the target platform are used in the follow up phases.

6.5.7 Extended SW Integration Testing



In this phase the Embedded SW is produced and tested for the target platform. Therefore, the target platform, i.e. the target hardware, must be available.

- Input:
  - Specification for “SW integration testing”
  - SW Units Implementations
  - Target HW platform
- Activities:
  - Implement tests from “SW integration testing”
  - Link units from SW Unit Implementations for target platform -> Embedded SW
  - Run tests on target platform
- Output:
  - Embedded SW
  - Update SW Verification Report
    - Add test reports

To create the Embedded SW, all the binaries of the SW Units for the target platform are linked together. Depending on the target platform, additional activities might be necessary to produce the Embedded SW, for it to be up-loaded or installed on the target platform.

Then the Embedded SW is tested with the SW Integration tests, typically in a HIL environment. Therefore, these SW Integration tests must have been implemented based on the "SW integration testing" specification. These test implementations must be reviewed. Please note, that there are many different tools to support HIL integration tests. In practice no single tool provides all the necessary functionality. Tools that might be necessary for HIL integration testing are:

- Environment simulation (e.g. custom Simulink models, Car Maker, VTD)
- Rest bus simulation (e.g. CANoe, dSpace tools for rest bus simulation, NovaCarts Rest Bus Simulation)
- Test development & test automation (e.g. ECU-TEST, TRACE-CHECK, TPT)
- Debugging interface to the HW (e.g. CANape, Lauterbachs TRACE32)
- Test management & test execution (e.g. TEST-GUIDE)

The tool selection depends on

- the functionality of the item
- the safety and security requirements to test for
- the expirence of the test engineers, developers, and test experts,
- the target HW platform

Test development can happen parallel to SW Unit implementation. That is, because the test development only depends on the specification for "SW integration testing".

The test reports should also be attached to the tickets of the SW integration testing. Similar to the SW Unit tests, the integration tests must be linked back to the technical (safety & security) requirements and the SW requirements the tests cover.

6.5.8 Extended Verification of SW Requirements or Validation of the item

With this phase the SW development is finished as soon as the Embedded SW reaches the goals for SW integration testing (test coverage, requirement coverage (see traceability below)).

However, if a complete item is developed, the item must be validated, i.e. the item must be tested in real vehicles according to the Safety & Security Validation Specification. Usually, these tests are manual tests, e.g. in test cars or on specific test benches. These tests must be executed and the results collected in test reports. For safety reasons it makes sense to only start these tests after the SW integration testing was completed without any safety related test failures. Like in the previous phases the validation tests must link back to the function safety & security requirements they test.

- Input:
  - SW Verification Report (incl. all test reports)
  - HW Verification Report (if it is not an SEooC)
  - Safety & Security Validation Specification
  - SW Requirements
  - Technical Safety & Security Requirements
  - Functional Safety & Security Requirements
  - Safety & Security Plan
  - Safety & Security Manual
  - Embedded SW
- Activities:
  - For Validation

        - Create & perform validation tests on vehicle level based on the Safety & Security Validation Specification
    - Calc Traceability
        - Coverage report (= which SW Requirements are directly or indirectly tested)
            - Problem: Are there any SW (safety) requirements without tests?
        - Test status report (= which SW Requirements are successfully tested)
            - Problem: Are there any SW (safety) requirements for which at least one test failed?
    - Justify untraced requirements
        - Goal: all safety requirements must be covered and successfully tested
    - Finalize Safety & Security Manual
        - Ensure that all information for the user is accounted for and up to date
    - Create Safety Case
        - Argument based on updated Safety Plan, SW Verification Report and Traceability analysis, why SW product can be released
            - All open activities in safety plan must have been checked
            - SW Verification report and traceability show no unjustified problems
        - For of a formal justification and not so much technical
        - = Release recommandation
    - Create Cybersecurity Case
        - Assessment of all open cybersecurity issues
            - Based on the calculated traceability and the Safety & Security Plan
        - All security related issues in the Safety & Security Plan must have been closed
- Output:
    - Safety Case (incl. traceability analysis and justification)
    - Cybersecurity Case (incl. traceability analysis and justification)
    - Updated Safety & Security Manual
    - Updated Safety & Security Plan
    - Embedded SW/item

In this phase the Safety Case and the Cybersecurity Case are build. For both the traceability of the safety & security requirements must be calculated (e.g. with a tool like Reqtify [37]):

- Coverages: How many of the requirements are covered by tests?
    - Calculated with the help of the links from tests to requirements.
- Test status report: How many of the requirements are successfully tested?
    - Calculated from the test reports of the SW Verification Report

If a complete item is tested, the test and test results of the HW Verification report must be included in the traceability calculation. Any not tested safety or security requirement must be justified (if possible), e.g. with reference to other activities that could replace tests from the Safety & Security Plan.

Safety Case and Cybersecurity Case can be argued together. The SAE J3061 names the creation of the Cybersecurity Case the Cybersecurity Assessment. For both cases all safety & security related activities of the Safety & Security Plan must be successfully completed. Hence, first the Safety & Security Plan must be reviewed for open activities and then updated.

For an SEooC: The Safety & Security Manual must be finalized, by reviewing that all required information in the manual is complete and up-to-date. The completeness and “up-to-date-ness” can be based on the Safety Case and Cybersecurity Case.

6.5.9 Extended SW Release

- Similar to 5.9
- Archive and document
  - Functional Safety Assessment & Function Safety Audit
    - Depends on ASIL
  - Final Cybersecurity Review
  - Every document from this section, i.e.
    - Item definition
    - HARA (incl. confirmation review)
    - TARA (incl. confirmation review)
    - Function Safety Concept (incl. confirmation review)
    - Function Security Concept (incl. confirmation review)
    - Safety & Security Plan (incl. review)
    - Technical Safety Concept (incl. confirmation review)
    - Technical Security Concept (incl. confirmation review)
    - HSI
    - SW Requirements
    - SW Architectural Design
    - SW Unit Design & Implementation
    - SW Unit Test Specification
    - Specification for “SW integration testing”
    - SW Verification Report (incl. all linked reports and reviews)
    - HW Verification Report (incl. all linked reports and reviews)
    - Safety Case
    - Cybersecurity Case
    - Safety & Security Manual
    - Embedded SW/item
- Note:
  - Archiving must be done in a way that ensures the integrity of the archived files, e.g. by storing the SHA-256 checksums of the archived files.

In this last phase the release of the Embedded SW or the whole item must be prepared. For a whole item a Function Safety Assessment and Functional Safety Audit are necessary depending on the ASIL level. For the audit a reviewer reviews the development process independent of actual developed item. For the assessment the Safety Case and all work products it refers to (Safety & Security Plan, and down to SW Units) are reviewed. In practice both reviews usually just sample the work products to review because of the volume. For security, a Final Cybersecurity Review must be done. SAE J3061 8.4.9 lists all the work products to be included in the review.

After the successful Functional Safety Assessment & Audit and the Final Cybersecurity Review, the Embedded SW (if it is an SEooC) or the item can be released for production. All work products listed above

must be archived and available for production and/or later use (e.g. to show that the Embedded SW was developed according to ISO 26262).

7 Supporting Processes

Supporting Processes are additional templates, steps, and documentation, that are shared by several projects and support project management. Some supporting processes have additional steps for safety & security. These steps will be especially highlighted below. For other supporting processes it makes only sense that QM is done on the same quality level as it is done for safety & security. Hence, these supporting processes do not distinguish between QM and safety & security.

7.1 Document management

Document management defines how to create documents that are either part of a project or define how to do a specific process or process step. Especially all documents that are subject to the Functional Safety Assessment (which are part of the project), Function Safety Review (define processes) and the Final Cybersecurity Review (both project and process) must be under document management. However, it is best practice to also put all documents for QM projects (both documents part of the project and documents defining the project's process) under document management.

Documents are any work product of a project that is not code itself, e.g.:

- Item definition
- Safety case
- Generated documents

Documents that define the development process, e.g.

- Definition of V-model process (most likely split into several documents for each phase)
- Definition of Scrum
- Definition of Document Management itself (and other supporting processes)

A document requires besides its content the following entries:

- Title
  - E.g. not just "HARA", but "HARA for item …"
- Autor
  - For generated documents who generated the document
    - Can be either a person or a system (e.g. Jenkins)
- Reviewer
  - Only for documents that require reviews
    - i.e. review reports themselves do not require reviews; they will be reviewed by assessors and/or auditors
- Status:
  - Draft
  - Final
  - Revoked
- Version
  - Version of the document and not the item
    - E.g. design documents, test reports can exist in multiple versions of for on item
  - For manually written documents either use simple increments
    - E.g. version 1, 2, 3, 4, …
  - Or semantic versioning [9]
- Version history for reviewed and manually created documents (in form of a table) - For any past version:
  - Version
  - Autor

    - Reviewer
    - Changes
- Optional (but useful) sections of a document
    - Purpose – should answer:
        - What is the content of this document?
        - Who is the reader of this document?
    - References
        - Sources of this document with their version number!
        - E.g. a specific version of the document "HARA" is derived from a specific version of the document

It is good practice to work with organization wide usable templates for each document. This ensures that important/required content is part of the document (e.g. version history, status – see above). Depending on the document the template can contain check lists to fill out, i.e. for review reports.

Best practice for documents is to use Confluence. Together with a Confluence template the required documents parts can be (partly) automated, especially the history. If documents must be given to external users, they can be exported with the PDF export.

However, the PDF export of Confluence can only be customized to a certain degree. Hence, one might like to use MS Office (Word, PowerPoint, Excel). Also, for MS Office templates help to ensure the usage of the required document parts. Some tools (e.g. FMEA tools) can also export documents in this format.

The formats of generated documents depend on the tool that generates the documents. Such documents may be online documents (e.g. some browsable static analysis report). However, safe document storage must be ensured (see 7.2).

The document creation process for manually created documents must be documented, e.g. with a Jira ticket. It is especially important to also archive the review comments to later prove in an assessment or audit, that the documents were really reviewed. Auditors expect for non-trivial documents intermediate states with remarks from reviewers. For example, comments in MS Office, then the documents with these comments must be archived (e.g. in the Jira ticket). Or for Confluence one can use inline comments of Confluence, which are archived by Confluence itself.

The document status must be tracked in the document and the Jira ticket. While in writing and in review a document is in Draft. Only after a document was successfully reviewed, e.g. all review comments are addressed and the reviewer has not furthered remarks and contains all required parts, it can be released. With the release the document is Final. The release must be documented in the Jira ticket together with the name of the releasee (who should not be the document author). If a document is fully replaced, e.g. by an updated version, it can enter the state Revoked.

To document project specific tailored process changes, it is useful to create project specific documents, which refer to the generic documents and only document the differences to the generic documents.

Documents must be stored in a way, that all developers, reviewers, auditors, etc. can access them. The document storage must be documented and people that must regularly work with these documents, must know where they are stored and how to access them.

Also, documentation can be done in other tools like BitBucket for code reviews or in Jira tickets, to document reviewing. Here is the overlap between document management and configuration and change management. This kind of documents should also once be checked, whether it provides all the necessary document parts. Usually, this is already done be the tools or could be easily added. Another good possibility is to just user an existing link or add a new link to the information (e.g. a from the pull request to the Jira ticket of the SW Unit).

7.2 Configuration management

Configuration management defines how projects artifacts are stored. The goal is reproducibility. For example, the attaching of intermediate document versions with comments (mentioned in document management) is part of configuration management.

Usually, for source code the central part of configuration management is to use a source code management system like git or svn. Best practice is currently git. The other central part for source is a branching model like gitflow [34]. This workflow must be tailored to the company and the tools used in the company. For Configuration management release and development branches are more important. Feature- and bug branches are more relevant to change management (see below).

It is best practice to put **all** manual work products and created work products under version control. However, great care must be taken, whether the work products are easily mergeable by the version control tool. Work products could be difficult to merge because of their format:

- Either binary (e.g. images) or
- A complex text-based format, which is changed by the tool on many places at once even for trivial changes to the work product (e.g. IBM Rhapsody is known to produce such files).

The branching model and configuration management should define separate measures to deal with these files, for instance with a kind of organizational lock or by allowing just one editor.

Generated work products should either:

- be manually created and put under version control (e.g. exports of an FMEA tool) or
- automatically created by a build server.

The automatically generated work products binaries, test, and traceability reports, must also be archived, at least for any builds that are given to the outside world, e.g. releases, beta-releases. Solutions range from ZIP-files (packed by Jenkins) to packaging tools like conan [35] and package repositories like Artifactory [36]. This tooling requires a defined process in line with change management and most likely also the branching model.

How to assign version numbers for releases is also part of configuration management and interacts with the packaging of automatically generated work products. Best practice is semantic versioning [9]. If marketing requires different version schemes it is best to use two version numbers: a marketing version number and a developer version number. The mapping which must be one-to-one must be documented.

7.3 Change management

The topic change management covers the documentation of changes (new features, bugs, …) to the project.

Change management works tightly together with configuration management to document and track changes in development. The sections about planning and unit design, implementation and testing already contain information about change management (5.2, 6.5.1.5, 5.5, 5.6, 6.5.5, and 6.5.6). Scrum also defines parts of the change management (see 8.1).

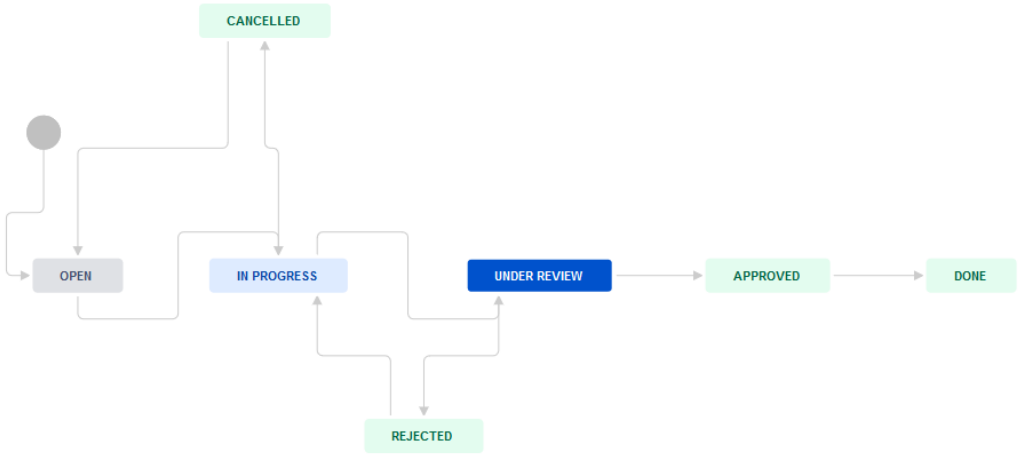
Change management usually provides a ticket system like Jira, with defined ticket types and workflows and links between tickets and tools and processes of the configuration management. For instance, the branching model is also part of change management for feature- and bugfix-branches. In Jira links between tickets can be typed, e.g. blocking links, and parent-child relations. This helps to structure tickets of different types. These link types can also be queried.

The release process should be defined as part of the change management. It relies on the release-branches and tags defined in the configuration management and defines release checklists (usually as ticket templates).

Required ticket types mentioned in this document are:

- Planning tickets: Sole purpose of these tickets is to plan and link to other tickets. For instance for each phase of the V-model a planning ticket should exist and all tickets for individual work products of this phase should be linked as child to the planning ticket.
- Document creation tickets: This ticket type is part of the document management. See 7.1 for more details.
- SW Unit tickets: Two tickets per SW unit (for the unit specific & implementation and for the unit test specification & implementation). For more details see 5.5, 5.6, 6.5.5, and 6.5.6.
- Test specification & implementation: Similar to the SW unit test tickets, but for implementing and reviewing the integration and validation tests (see 5.7, 5.8, 6.5.7, and 6.5.8).
- Manual test tickets: Tracks the progress of a executed manual test (either in integration or in validation phase). Note: automation tests are not tracked with separate tickets, but in the phase planning and releasing tickets, because automated test execution is done by the CI-system.
- Releasing tickets: Documents the release of product. Links to the appropriate work product tickets and serves as guard to ensure, that all work products are finished.
- Support tickets: Support requests are documented and tracked with support tickets. Some support requests can lead to bugs and new features.
- Bug tickets: Due to a support request or a bug found by developer a bug ticket is created. This helps also to document known bugs and work-arounds. When a bug is fixed, it most likely lead to changes to SW Units, tests, document etc, which should be documented by the appropriate tickets. Bug tickets of known bugs most be listed as known bugs (if they are not yet fixed in a certain product version) or as fixed bugs (if they are fixed in the current product version) in the product changelog.
- Feature tickets: Similar to bug tickets, feature tickets document possible further changes to the product. However, instead of defects for bugs, feature tickets track possible new features. Feature tickets do appear in the changelog for the version in which they are included.
- Analysis tickets: Tickets for small research tasks that should be done as part of the project. This tickets help to analyse some uncertainty within a project to derive decisions and follow-up actions. For instance, the first step to fix a bug is to analyze it and to derive the concrete bug fix as follow up action. In general, most of the analysis work should be done in context of the research project (see 9.1).

All these tickets may have slightly different workflows. The standard workflows of Jira or other ticket management systems might be reusable, but should be reviewed before just using them. For instance, this is the standard workflow of “task” ticket in Jira 8.7.1:



Important states of the workflow are:

- Open: a ticket is still in specification or implementation has not yet started.
- In progress: the work specified by the ticket is currently being done.
- In review: the work product, SW Unit, test, .. is being reviewed.
- Done: the tickets work was accepted as being completed and the ticket itself is documented.

The names may differ. Depending on the company and process. For instance, for Scrum it makes sense to add another state between Open and In progress for indicating, that a ticket is selected for implementation in the current sprint.

Ticket templates, that define the format and structure of the description of the ticket help to:

- Unify tickets of the same type,
- Ensure that important information is documented within the ticket, and
- Simplifies creation and handling of tickets with large contents like manual test tickets and releasing tickets.

7.4 Tool selection

Tool selection is about choosing the right tool for the right job. This section focuses on the software development aspects of this process. Other aspects like purchasing, and security of the tool installation are out of scope here.

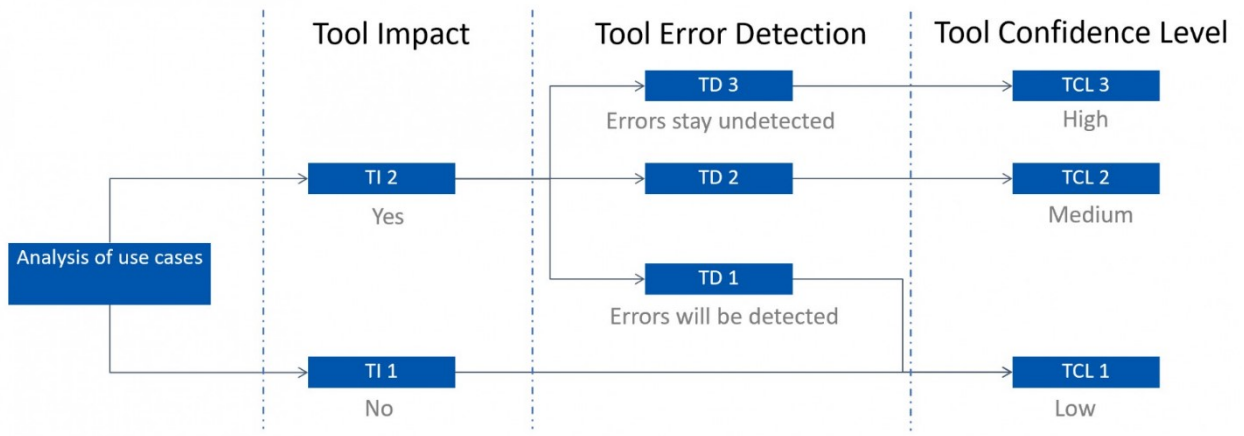
For each tool to be used in SW development, it should be documented where the tool comes from, whether the tool license is complaint with the requirements of the SW product development and the company. Also, how, and where defects of the tool can be tracked and a list of known bugs with workarounds. If one or more specific specialists for this tool exist in the company should also be documented.

Usually tools need to be adapted to the company and their processes, e.g. with templates, custom plugins, custom patches, and custom processes incl. documentation. This should be also documented for each tool.

Hence, for each tool a Confluence page should be created to document the above-mentioned topics. Also, the approved versions and platforms for each tool must be documented. Each tool user should watch the Confluence pages of all tools in use, to be notified about updates of the tool.

For safety & security, more work is necessary. Each tool must be analyzed, whether it could potentially introduce safety or security violations into the final product. This step is called tool impact analysis. For instance, a compiled can mis-compile a source code or a testing tool could report test success for failed

tests. When a tool can have any impact on safety and security, the tool error detection must be analyzed in the context of the overall usage of the tool in the development process. For example, for compiler bugs it could be argued, that they will be found by unit tests (if the unit testing is done on the target platform and done to a high degree of confidence). Hence, it may make sense to change the development process slightly to ensure that a tool has a good error detection. However, it is difficult to argue how at least some of the possible failures in a test tool can have no safety & security impact on the final product. The following picture from [38] illustrates the process. At the end a tool confidence level is derived.



For any tool confidence level above TCL 1, the tool needs to be qualified for the required use cases. The qualification could either be done by the tool vendor (sometimes sold separately as qualification kit) or in-house, by validating all use cases (e.g. with doing testing). In any case the result of the qualification must be documented. And only tools that are either successfully qualified or a tool confidence level of TCL 1 may be used in safety & security projects. Note, the qualification must be redone, if use cases change or the software is updated to a new version. More details about tool qualification can be found for example at [38].

The whole tool section and tool qualification process also applies to tools that are build in-house or are open source, especially for safety & security projects.

7.5 Qualification of Software Components

Qualification of software components must be done for 3rd-party software (open source and commercial) that is used as part of the final software either in source code or binary format. The process is like the tool selection process (see 7.4).

For each 3rd-party software documentation must be maintained about:

- its versions,
- known bugs and workarounds,
- Support links and defect tracking,
- In-house experts,
- license and any required follow-up action for this license
  - Like delivering the license together with the final product, which is required for some open source 3rd-party software components.
- If the software is qualified for safety and/or security its safety and/or security manual must also be part of the documentation about the software.

For safety & security there are two cases:

1. If it is already qualified up to the required safety and/or security levels, the software can be used (according to its safety and/or security manual).
2. Otherwise, the software must be analyzed for its use cases. For example with a software FMEA. This software FMEA might lead to additional requiremens that in turn lead to work around possible safety and/or security issues found in the software FMEA. These requirements must be generically documented in an in-house safety & security manual of the software and be applied to any SW project including this software on a use-case basis.

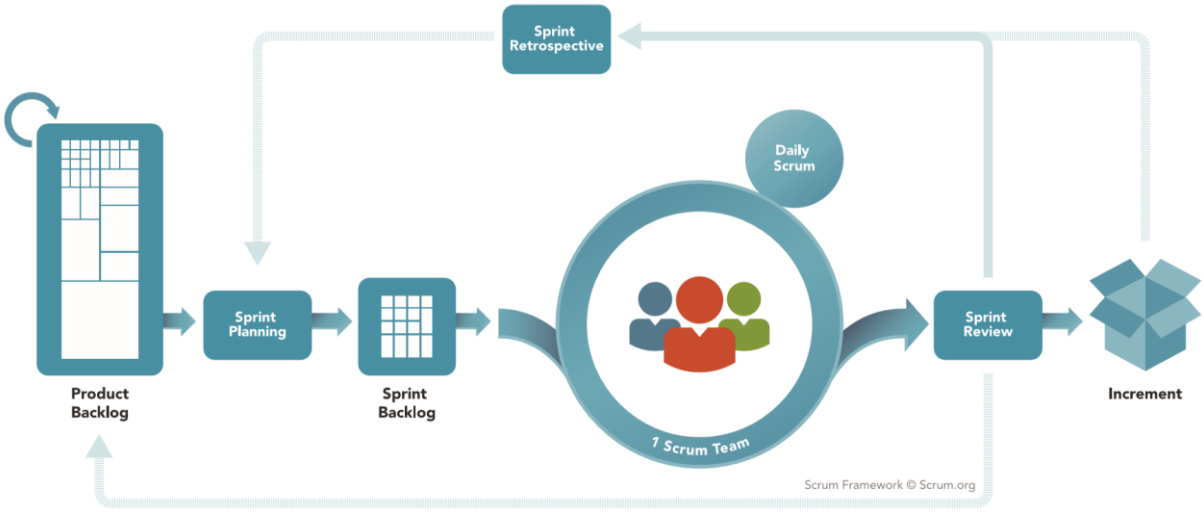
8 Agile development process based on scrum

The previous chapter described the essential work products of a SW development project for different quality levels (QM, safety, and security) with dependencies. Although these dependencies between the work products and the phases of the V-model define order, it is uncommon the for a single software project to go through the V-model from left to right without iteration. Every time a verification measure on the right side detects a "quality violation" (e.g. a failing test) one must go back to the left side and start partly over (e.g. fix a bug, or rework a requirement).

This iterative approach is much more common in SW development today. This chapter introduces the iterative SW development process Scrum [15]. First it defines what Scrum is and how it works on a generic level. Then it discusses on how to apply the iterative Scrum to the on the V-model of the previous chapter.

8.1 Generic Scrum

Scrum is an iterative SW development process. The scrum process is suited to long running projects including projects without any defined end. The iteration is shown in the figure from [16] below:



On the left Scrum has a product (or project) backlog. This backlog contains any "task" that must be done for the project in the future (bugs, features, requirements). The term task will be refined later in this chapter. The backlog will be continuously updated, e.g. by planning new features or changing existing features. Each iteration is a sprint. A sprint starts with a Spring Planning meeting. In this meeting the Scrum team selects all tasks from the backlog to implement or address in the next sprint. All selected tasks go into the Sprint Backlog. Now the team works continuously for a defined and fixed period on this Sprint Backlog. Each working day the team does one "stand-up" meeting called Daily Scrum to discuss urgent topics. At the end of the sprint duration the team reviews the tasks in the Sprint Backlog in the Sprint Review meeting. The goal is to release a new increment of the product. This might be a beta version or even a real release. But especially for safety and security the product increment after the first iterations might be still incomplete. After the Sprint Review the team does another meeting the Sprint Retrospective. Here the team discusses the lessons learned from the last sprint and with that in mind goes on into the next iteration with the Sprint Planning of the next sprint.

8.1.1 Roles in Scrum

Scrum defines 3 key roles in a Scrum Team:

- Product Owner/Project manager
- Scrum Master
- Scrum Developer

Each Scrum Team has one Product Owner and one Scrum Master. The whole team together decides on the length of a sprint. Best practice is to have Sprints between 2 – 4 weeks. The longer the sprint the more complex and larger the tasks of the product backlog may be. However, longer tasks reduce agility, i.e. the ability to react to unforeseen problems. In safety and security projects sprints are typically longer than in QM projects. Changing sprint duration can be done in the Sprint Retrospective, but usually not within a sprint. Also, to the Scrum Team's ability to judge the size of the Sprint Backlog from experience it is best to change the Sprint duration not at all.

8.1.1.1 Product Owner/Project Manager

Product Owner is the "Scrum" name of the project manager from the previous chapter. In this chapter we mostly stick to project manager, but might sometimes also use the term product owner to better reflect the responsibility of this role.

The product owner manages the Product Backlog. In some settings the product owner is the customer of the product. But it may be also the product manager and, thus, the product owner is also responsible for safety and security. The product owner's main tasks are [15]:

- Defining backlog items,
  - Any work product of the previous chapter is part of the backlog. Its creation might be a single task or split over several tasks for larger work products.
- Ordering and prioritizing the backlog to best achieve the overall goals and missions
  - The product owner must also take into account the value any in-between increment with this ordering
  - Of cause the ordering is strongly influenced by the dependencies between work products
- Ensuring transparence for the team by ensuring the backlog is completely visible by the whole team
  - The goal is that the team knows the overall goals and what to do next
- Ensuring that the team understands each task in the backlog
  - If necessary, the tasks must be discussed in the Sprint Planning

The Product Owner may also take on the role of a Scrum Developer, but must not share the role of Scrum Master. The Product Owner may be part of several different Scrum Teams (even in different roles).

8.1.1.2 Scrum Master

The Scrum Master supports the team by implementing Scrum. It is a kind of servant leadership. The Scrum master is independent of the product owner/project manager. Both roles must not be shared by one person. The Scrum Master supports the team by ensuring the interactions from outside of the team happen in a way the maximizes the values the Scrum Teams produces. That does not mean that the Scrum Master blocks or intercepts all communication between the Scrum Team and the outside world, but the channel and organize it in ways that the Scrum Team can maximize "output".

The main tool of the Scrum Master is communication. The Scrum Master knows the process by heart and helps anyone in the Scrum Team to understand and live Scrum. Besides project management experience, knowledge of Scrum and very good communication skills are a requirement of a Scrum Master. A Scrum

Master does not need to be a SW developer. However, experience with SW Development are an advantage. The Scrum Master might be also a Scrum Developer. A Scrum Master can be part of multiple Scrum Teams.

The Scrum Master helps the project manager (product owner) to organize the backlog in a way to achieve the tasks of the project manager (see above). For the Scrum Developers the Scrum Master coaches the team in self-organizing and cross-functionality (see below). The Scrum Master removes impediments for the Scrum Team. The Scrum Master couches the Team and if necessary, supports the introduction of Scrum in Team which not full live Scrum yet.

8.1.1.3 Scrum Developer

The Scrum Developers from a self-organizing team. Neither project manager (product owner) nor the Scrum Master tell the team how to implement backlog tasks. Both only define the requirements and the priorities.

Scrum Developers must be cross-functional. That means if necessary, Scrum Developers must have safety and security experience. They must do architecture, implementation, code reviews, write tests, do manual testing, asses HW metrics and so on. Of cause, some Scrum Developers may have more expertise in certain fields than others, but in general any Scrum Developer must be able to do implement any task from the backlog. Regarding safety and security, it is best practice to let the more experience Scrum Developers support creation and do reviews, but let not as much experienced Scrum Developers do the implementation. The Team may also decide to practice pair programming in order to improve the experience level of Scrum Developers.

Also, the Scrum Developers take accountability for the implemented tasks. Sub-optimal results or failures must be addressed by the whole Scrum Team in the Scrum Retrospective meeting and hence forward in the next iterations.

The size of the Scrum Developer team must be at least 3 in order to achieve useful progress. It should be less than 10 in order to avoid sub-teams or to destroy the accountability. If Scrum Master or Product Owner are also Scrum Developers, they also count into the Scrum Developers team size, otherwise they do not count.

Independent reviewers for safety and security should not be part of the Scrum Development Team (e.g. for Safety I2 and I3 and in general for security). The project manager and the Scrum Master must organize the reviews with these external reviewers. The goal is to sync these reviews with scrum sprints. The Scrum Developers must decide (based on the Sprint Planning) when the work products to review are reviewable.

8.1.2 Scrum Elements

8.1.2.1 Product Backlog

Best practice is to organize the product backlog with tickets of a ticket management system. All tasks of the backlog are tickets. Usually the creation of a work product (e.g. function safety concept including function safety requirements, SW architecture, unit design, unit test, …) is one task. It is best practice to specify tasks as user stories and use cases. That would mean, to implement a user story, requirements, design, implementation, and tests must be extended. Because a use case touches the whole (SW) V of the V-model. However, in safety and security context use cases are often to coarse grained. Especially, updating requirements, design and integration tests will be inefficient.

Hence, it is best practice to prioritize the (preliminary) creation of work products on the top of the V-model by the product manager, e.g. concepts and requirements and validation and integration test specifications. However, these preliminary work products may be refined by the development team in later iterations.

Besides the work products of the previous chapter, bugs resulting form support may also be user stories. Again, the priority of the bugs is defined by the project manager.

The team may also request the project manager to put tasks into the Product Backlog. Mostly these are tasks to reduce technical depth, e.g. code or documentation refactoring. These tasks could be the result of the Sprint Retrospective. While the project manager defines the priorities also for these tasks, it is good practice the allow a certain percentage of such tasks in each sprint.

8.1.2.2 Sprint Backlog

The sprint backlog is defined by the Scrum Team (Product Owner, Scrum Master and Developer) at the beginning of a sprint in the Sprint Planning. Within the sprint the sprint backlog documents the progress. The goal for each sprint is to complete any task of the sprint backlog. However, uncompleted tasks are discussed in the sprint review.

8.1.2.3 Increment

An increment is a release of the build product. However, that only works in QM projects. In safety and security projects a release requires full traceability and a final assessment. Hence, in safety and security projects only pre-releases may be done. Especially, in the first iterations, pre-release might not be functional at all (this is also true in QM projects).

8.1.3 Meetings in Scrum

All scrum meetings must be organized and if applicable documented by the scrum master. All Scrum Team member must attend each Scrum Meeting (if not ill or on vacation). Preferable Scrum Meetings are attended in person. However, video conferencing is a good option, if:

- All attendees are visible in the meeting by video
- All attendees see a shared screen with the Sprint Backlog, Product Backlog, Meeting protocol in the wiki, etc.

8.1.3.1 Sprint Planning

The Sprint Planning happens at the beginning of the Sprint. The project manager selects tasks from the backlog to implement by the development team in the next sprint. The development team defines how many of the selected tasks (based on the priority of the project manager) will be implemented in one sprint.

Also, the Development Team can discuss the order of tasks to consider for the Sprint Backlog with the project manager and Scrum master in the light of impediments (e.g. missing HW or other resources) and dependencies between tasks.

Each task requires a definition of done. That could be either a reviewed document or a document without review (and the review will be done in the next Sprint). The same goes for implementation with or without tests. This definition of done must be documented within the ticket of the task. See below on how to deal with too large tasks.

A best practice to decide on the completion of the Sprint Backlog is to use story points. For each sprint the team decides on several story points to implement within the sprint. All the tasks of the backlog are assigned story points by the Scrum Developers by a transparent open vote (e.g. hand signs). Usually only a small fixed set of story points are available for a task. For example:

- 1 for tasks judged as simple/trivial

- 3 for tasks judged as normal
- 5 for tasks judged as maximum
- 8 for tasks judged as too risky or too big for one sprint

The story points per sprint are highly individual per Scrum Team and sprint duration. It is the task of the Scrum Team to learn over time the right number of story points per Sprint.

Tasks judged too risky should be split into an analysis task and one or more implementation tasks. The analysis task is done in the sprint at hand. The result of the analysis tasks might be beside a proof of concept or a concept, the implementation tasks, to be done in follow up sprints. The same goes for too big tasks. However, the case the too big task can be easily split into several tasks, the analysis task can be skipped. The product manager must learn over time to create tasks to a size of 3 or 5.

Documentation of the Sprint Planning is the assignment of tasks to a sprint backlog in a ticket management system. Usually a sprint backlog is just a specific version of a product/project managed by a ticket management system. Additional, documentation can be done in a wiki as a meeting page.

8.1.3.2 Daily Scrum

The daily scrum is a brief daily meeting at a fixed time. Best practice is to attend the meeting standing to keep the meeting duration around 15 minutes. Every attendee tells what she/he has done since the last Daily Scum and if she/he encountered any impediments. Impediments are not solved ad hoc, but by either taking responsibility (usually be the Scrum Master for missing resources) or be scheduling follow-up meetings (e.g. between developers for discussion, pair programming, with the project manager for clarification on tasks, etc.).

Usually, Daily Scrums are not documented at all. However, follow-up meetings should be. Either in the ticket of the task involved or in separate meeting protocols in the wiki.

8.1.3.3 Sprint Review

In the Sprint Review the team reviews together each task (in terms of tickets) of the Sprint Backlog. The team decides together, if the task has reached its definition of done. The project manager, may use the Sprint Review to try out features or look at documents. However, testing should be done beforehand and documented in the task's tickets. Developers can also demonstrate the result of important tasks (e.g. with demos or by explaining analysis or documents like architectures).

Tickets of tasks that are done (by their definition of done) can be closed. All other – incomplete – tasks are moved back to the Product Backlog. Usually, these tasks are considered first in the next Sprint Planning, but might be reevaluated (see above for dealing with too large and too risky tasks).

No blame or judgement is done on incomplete tasks. This discussion is addressed in the retrospective.

8.1.3.4 Sprint Retrospective

The goal of the retrospective is to improve the Scrum process for the team. Therefore, each team member presents first what was "good" and "bad" in the last sprint. However, it is not the goal to blame individual persons. First of the whole developer team is accountable. That also means developers must help each other. The Daily Scrum is the place to request and offer help to each other (see above). However, personal conflicts may occur. It is the job of the Scrum Master (with the help of the project manager) to solve these conflicts.

In general, the team should do more of the good points and change doing the bad points in the next sprints. How this can be done is discussed openly after each team member give her/his statement.

The retrospective must be documented with a meeting protocol in the wiki. The derived tasks, actions and lessons learn must be addressed where appropriate (e.g. the sprint planning).

8.1.4 Documentation/Implementation

8.1.4.1 Scrum Board

It is best practice to visualize the Sprint Backlog as a Scrum Board or Kanban board. This can be done the ticket management system and, additionally, on a wall/whiteboard with A6 or A5 large post-its. The Scrum Board is split into three columns: TODO, In Progress, Finished. Depending on the project and team other columns are possible, e.g. In Review and Testing. Each (post-it) task is moved the column to column (TODO being usually left and Finished right). These task changes are mirrored in the ticket management system by a ticket workflow with the same states as the columns on the board. Beside TODO and Finished each task/ticket has an assignee. The assignee may change in agreement within the team (e.g. by going from In Progress to In Review from implementer to reviewer).

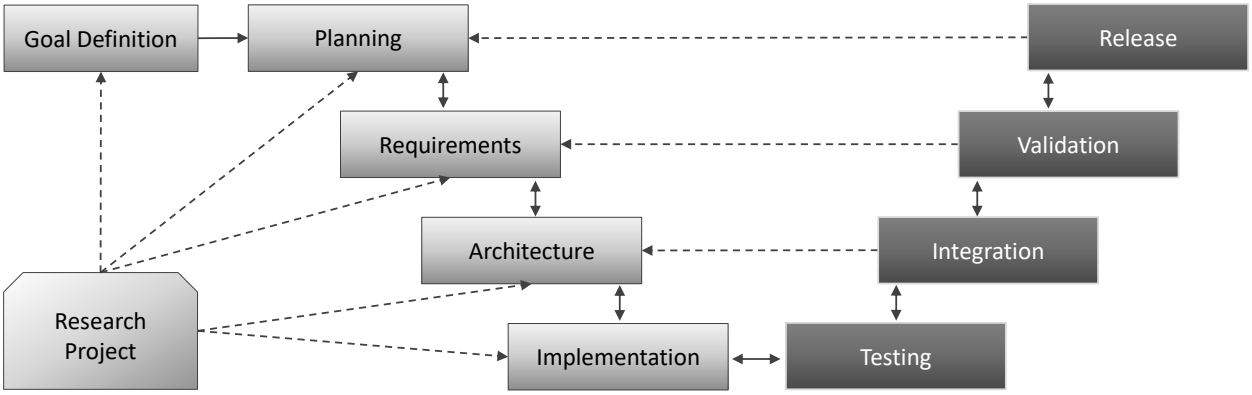
8.1.4.2 Atomicity of tasks

Ideally each task is implemented in the increment/product at the end of a sprint. However, in practice this will not always happen. To keep the increment clean of incomplete tasks, the tasks result must be kept away from the final product if the task is not finished. For documents in a wiki or shared folder this can be done by storing several versions of a document, with each version marked as draft if the task for this version is not finished. For code (implementation, tests, LaTeX documents, …) feature branches of the version control system (e.g. git) must be used. For each task one feature branch is created. The task is implemented and reviewed in this feature branch. And only if the underlying task is complete the feature branch is merged into the master branch. The feature branch may be deleted after this merge.

8.2 Iteration of the V-Model

The V-model provides more of a what should be done and in what order. However, it does not define (except of the order) any priorities between the work products. The regular planning, review and retrospective meetings in Scrum help to adapt the process of creating the work products of the V-model to the project being done.

Also, the V-model already contains iterations:



Any phase on the right arm points back to the phase on the same level of the left arm. This describes for possible loops of iterating the V-model:

- Implementation loop: This is the most tight loop. Any SW unit is iterated until its testing is successfully completed. Usually that is done with one sprint (at least that is the goal).
- Integration loop: Any defects found in integration testing have to be analyzed in the context of the architecture. It might be that only one SW unit needs to be changed, e.g. because in protocol was not specified or implemented correctly. But it could also turn out, that several SW Units are affected. If possible this loop could also be done in one sprint. But it might turn out, that found defects are a result from SW Units finished in previous sprints. Then updates to these SW units have to be planned for one of the next sprints.
- Validation loop: This is similar to the integration loop, but just for validation. Here it is much less likely, that the found issues can addressed in the same sprint. Also, the requirements are usually derived at the start of the project and, hence, as long as the phases below are unfinished the validation cannot be fully performed, i.e. traceability will be incomplete.
- Release loop: This loop will only be iterated if management does not release the software (for QM) or the project fails either the function safety audit & assessment or the final cybersecurity review. In any case the planning will most likely have to be updated and depending of the reasons some parts of the Goal Definition might need to be repeated, e.g. if the HARA was found to be unsufficient. The release loop is only iterated on demand, i.e. the release step does not pass audit or assessment or the management does not approve the release. The text below explains how the iteration is done in practice without the release loop.

These loops are iterated in different kinds. The Implementation and Integration loops can be done as Scrum sprints. This is also true for the validation loop, but this requires a high degree of automation of the traceability analysis. And usually the validation testing will be skipped in most sprints. A validation test should only be done, when the development of all the safety goals is completed and all security relevant integration tests pass.

In practice the Goal Definition and the Planning phases are usually finished in Scrum before starting with the following phases. Then starting from that point in time sprints could be done on the Validation loop. Each sprint adding new SW requirements until all technical requirements are covered and fulfilled.

Depending on the project it might be better instead of starting right away with the Validation loop, to first specify SW requirements and architecture within one sprint and then start with the Validation loop. In that way, the architecture does not need to be re-engineering in each sprint, because new SW requirements require completely different architectures.

Also, for safety & security it is impossible to provide an increment after each sprint, because function safety audit & assessment as well as the final cybersecurity review should be only done once for each release. In this context it is common to anticipate follow-up work, i.e. at least a second iteration of the release loop. It is the task of the auditors and reviewers to find issues, that need to be addressed in this second iteration.

As already mentioned, the product backlog will serve as safety & security plan.

8.3 Requirements for Scrum

Like the V-model Scrum can only be applied to projects and software products of a minimum size. If a project has not enough people to ensure independence between the different roles, Scrum and the V-model cannot be applied.

Also, the supporting processes are a requirement to setup Scrum (see 0). Short running project that would only result in one sprint also make not much sense for Scrum.

On the other hand, the Scrum team must not be too large (see 8.1.1.3).

8.4 Roles

Section 8.1.1 already defines the central scrum roles:

- Product Owner/Project Manager
- Scrum Master
- Scrum Developer

Section 6.3 defines the roles for developing safety critical and cybersecurity critical products:

- Product manager/owner
- Safety Manager
- Security Manager
- Safety Engineer
- Security Engineer
- Independent Safety assessor
- Independent Security assessor

The safety manager and the security manager may be filled in by the product manager. However, this might require a very high specialization to this person, that requires at least a formal education in a relevant safety standard and a relevant security standard and experience in both (e.g. as safety or security engineer). Whereas safety engineer and security engineer can map to Scrum developer. However, again this requires for the scrum developer at least some in-house education about safety & security and the relevant processes and standards.

The independent assessors cannot be mapped into the scrum team, because they would not be independent any more. Experts from external company (consultants) or from sufficiently independent department of the same company must fill these roles. Of course, assessors require the same education and experience as safety and security managers.

8.5 Alternative Models

Section 8.1.4.1 already mentions the Kanban board. It might be possible to use the whole Kanban process [39] instead of Scrum. However, Scrum is much more common in practice for software projects. Kanban might be better suited to processes like support (see 9.3).

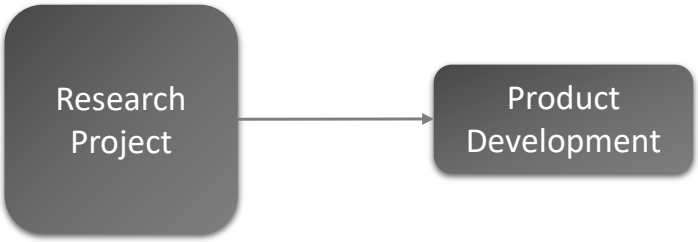
Scrum itself is not that common in safety and security projects. The standard method is just to focus on the V-model alone, walk along its arms, and iterate where necessary. The advantages of Scrum are to break the whole development timeline in smaller chunks with defined points form retrospective and possible changes. Hence, it defines a framework to deal with unexpected changes. Such a framework is missing in the “classic V-model”.

9 Project Life Cycle

This chapter takes the bigger picture into account and looks beyond software development.

9.1 The Research Project

This document assumes that every software development project is based on a research project. In practice this is common for most complex software development projects. Often a kind of preliminary project, which goal it is to derive concepts and prototypes is done before starting to develop a safety critical or cybersecurity item.



It is job of the project manager of the software development project to assess the research project while doing to planning to reuse as much as possible from the research project.

Furthermore the research project can already prepare for the follow-up software development project:

- Derive a good system architecture & software architecture
  - SW Units should be designed with testability in mind, even if unit testing is not done
  - Freedom from interference should already be considered
- Consider safety & security
  - Are there interesting question in novel safety & security concepts?
  - Pre-liminary safety & security analysis help to enable reuse of the architecture and with that possibly also SW Units
  - Expected safety & security levels should be anticipated and be taken into account when deriving the architecture.

The project manager has two basic options for going from the research project to the product development (software project):

- Evolution: Identify and fill in gaps in research project (tests, documentation, safety & security concepts, analysis, …). This option is only possible for a research project that already is very near to the software project. It is less likely, that evolution works for a safety and security critical project, because of the many requirements imposed by the safety and security standards, that are usually not fulfilled by a research project.
- Revolution: The research project helps to define use case and requirements as well as basic algorithms, but implementation and architecture are not reusable.

Experience helps the project manager to make the decision between these two approaches. The question is always what and how many parts of the research project look reusable:

- Architecture
- Source-Code
- Documentation

- Tests
- External components

9.2 Versioning

Versioning was already talked about in the context of configuration management (see 7.2). The best practice is to use semantic version [9]. Some projects give each increment produced by a sprint a unique version number. This helps to do rapid releases. However, especially in safety and security contexts it is unrealistic the finish every sprint with a finished release.

Usually bugfix releases are usually possible without a complete re-audit and re-assessment. But therefore, an impact analysis must be done to ensure, that the bugfix has no unexpected influences on other parts of the system (see 6.5.1.2). Even then it depends highly on the required changes if the bugfix can be done within one sprint.

9.3 Support

The support process is necessary for the project life cycle after the release for production. It depends on the transfer agreement of the software. This agreement could include support with SLAs. A two or three level support process is often used in practice:

- Level 1: The support request is accepted by a member of a dedicated support team. The support team member documents the support request with a support ticket (see 7.3). If the support team member can answer the request right away (less than 30 minutes), he does so, documents and closes the ticket. This is often the case for support requests that can be answered by public available documentation (user manual, FAQ, tutorial, examples). If answering is not possible the support team member passes the ticket on to the second level support.
- Level 2: In this level the product manager of the software project concerned by the support request or a designated team member takes over the the support ticket. If the support request can be answered right away or with just a small analysis (1-2 hours), it is done and the support ticket can be closed. This is typically the case for obivious bugs and feature requests. In this case follow-up tickets must be created and linked with the support ticket. The person initiating the support request must be informed and the support ticket can be closed. Otherwise the support ticket goes into the third level support.
- Level 3: Now a developer takes over and debugs the support issues. The result could be a bug ticket, a feature request ticket or an update to the FAQ. In some cases several of these could happen.

For safety and security related support requests the safety and security impact of the support issues must be judged and documented. Here workarounds are much more important and need to be documented if available in the support ticket and, when applicable the FAQ.

10 Tool support for agile software development

Tools for supporting the software development process were already mentioned throughout the whole document. This chapter lists these tools again in one place.

Depending on the intended use cases the tools listed here might require further plugins.

This list of tools is of course not complete.

10.1 Tools for Scrum & Supporting processes

- Document management
  - Documents in Confluence [41]
  - Document creating documentation in Jira tickets [40]
- Configuration management
  - BitBucket with git and gitflow for version control [42]
  - Conan [35] and Artifactory [36] for archiving generated work products
- Change management
  - Jira [40]
- Tool selection
  - Documentation in Confluence [41]
- 3rd party software component selection
  - Documentation in Confluence [41]
  - FMEA with Excel

10.2 Requirement Engineering & Traceability

- Requirement management with tickets of type requirement in Jira [44]
- Test management with Jira tickets as described in this document
- Traceability
  - Reqtify [37]
  - Self-developed script

10.3 CI

- Jenkins [43] is the defacto industry standard for CI

10.4 Language Dependent tools

- Static analysis
  - Klocwork [33]
  - Cppcheck [30]
  - SonarQube [45]
  - Clang-tidy [31]
  - Clang-format [29]
- Dynamic analysis
  - Valgrind [25]
  - Clang/gcc Sanitizer [24]
- Code coverage

    - Bullseye [26] and CTC++ [27] or C/C++
    - Coverage.py [28] for Python
- Compiler
    - Gcc
    - Clang
    - Visual Studio 2017, 2019
- Build tools:
    - Conan [35] on top of Cmake [46] for C/C++ projects
- Testing
    - Unit-Testing:
        - GoogleTest, GoogleMock [18] for C/C++
        - Unitest [22] and unittest.mock [23] for Python
    - Integration-Testing of Command Line Tools with
        - Lit [47] [53]
        - OutputCheck [48]
        - FileCheck [49]
    - Integration and validation testing of embedded SW: ECU-Test [50]

11 Appendix

11.1 Structure: FMEA

A (SW) FMEA can be an Excel document. The first sheet "Version" contains the required document attributes (see 7.1). The remaining sheets focus each on one FMEA for one topic (e.g. one diagram of 6.5.4). The FMEA in such a sheet is a table with the following columns:

- No – unique id of the error
- Ref – a reference into the diagram been analyzed
  - Could be more than one diagram element
- Function – (safety) function the analysis focuses on
  - This function must be (partly) implemented by Ref
- Error the function: description of the error/failure
- Fault: cause of the error of failure
  - Could be SW bug of a component/unit
  - Some incorrect input
- Failure: result of the error
  - Note a failure of component A might propagate into a fault of component B
- Detection: How is the error detected
  - Might be online: e.g. check inputs
  - Could also be specific tests, that "proof" that the fault cannot happen
    - These tests must be in Integration test specification
- Handling: How is function/component but into a safe state
  - For Tests: requirement, that these tests pass
  - Online: mechanism to go into a safe state
    - i.e.: returning an error in function B (called by function A) if B's input values are invalid is not enough -> function B must be required to handle the error

11.2 Major points in contracts with external companies

Contracts with external companies about SW development projects must adapted to the specific requirements of the concreate SW project and the planned tasks of the company. Contracts should be drafted together with an attorney at law. Usually contracts for concrete and finalizable projects with a runtime within weeks or months look much different from some long going collaboration agreements, that lasts over several years. The following major points should be clarified in such a contract:

- Subject matter of the contract
  - Either
    - Concrete SW to build ideally with specifications (might a reference to a specification document)
      - For small self-standing projects or extensions of existing SW products
    - Development Service
      - For long running projects
      - Development is done in specific work packages (negotiated separately)
      - Specific tasks to be done in development, e.g.
        - Concept creation, review and evaluation
        - Development of POCs
        - Creation and maintainance of SW architecture and unit designs
        - Workpackage evaluation
        - Programming

        - Testconcepts and test specifications
        - Creation., review and maintaince of developer documentation
        - Creation., review and maintaince of user documentation
        - Programming of automated tests
        - Creation of checklists for manual tests
        - Running tests (automated and manual)
        - Maintaince of test results
        - Debugging
        - Maintaince of project plan
        - Reporting of project progress
        - Regular communication
        - Demonstrations
        - Observe of HRI quality guidelines
        - For Safety/Security
            - Analyzes of criticality
            - FMEA (for safety)
            - Simulation and modelling
            - Calculation and documentation of safety or security related metrics
            - Code-Reviews
            - Documentation-Reviews
            - Extended testing (e.g. fault insertion tests or pen tests)
    - Define HW and all non-functional requirements that are known
        - Especially safety and security requirements
- Concrete deliveries
    - With Definition of Done
    - If possible with deadlines
    - Approvement process or checklist
- Project plan
    - For long running projects
    - Resource plan
    - Generic approvement process or checklist for work packages
- For Safety / Security: Certification/Qualification
    - Who pays for the certification?
    - Require experience and formal eduction for safety or security from certain project members
- Maintaince
    - Support, Maintaince agreement for developed SW
- Rules for Subcontracting
    - Is subcontracting allowed?
    - If yes, define ground rules responsibilty and communication
- Required quality levels of contractor
- Communication / contact persons
    - Define primary contacts
    - Be as concrete as possible
- Define where the contractor should work
    - Either at HRI or in the company building of the contractor
    - Define how often meetings in person should be scheduled
- Infrastructure
    - Who provides which parts of the infrastructure?
    - Jira, Confluence, git-Repositories, CI, share for documentation, …
- Property rights
    - Who owns the programmed code?

    - What can be done with the programmed code?
        - Can it be open-sourced? What license?
        - Can it be used in production?
        - Can it be used for safety? What are the limits?
            - safety levels
            - assumptions about surrounding system
        - Can it be used for security? (same points as for safety)
- Cancelation terms
    - Who may cancel the contract?
    - Under which conditions and terms?
    - What must be paid?`
    - What are the property rights in case of cancelation?
- What happens in case of a change of control in one company?
- Modus of payment
    - Pay per delivery?
    - Pay per work package?
    - How are business trips paid?
    - Payment for support and maintaince
- NDA
- Liability and insurance policies
    - Definition of limits for liability
    - Requirements on insurance of contractor
- No-competition clause
- Export restrictions

This list must be reviewed with concrete project in mind. The list might be incomplete and some points might not be to necessary.

12 Sources

[1] V-Model XT 2.3; Release: 04.02.2019; http://ftp.tu-clausthal.de/pub/institute/informatik/v-modell-xt/Releases/2.3/; (German; accessed March 2020)
[2] ISO 26262 („Road vehicles – Functional safety"); edition 2 from Dec 2018
[3] SAE J3061 "Cybersecurity Guidebook for Cyber-Physical Vehicle Systems"; 14.02.2016
[4] Christof Ebert "Systematisches Requirements Engineering" 3. Edition, dpunkt.verlag 2010; ISBN 978-3-89864-709-0
[5] S.Bradner "RFC 2119: Key words for use in RFCs to Indicate Requirement Levels" March 1997; https://tools.ietf.org/html/rfc2119 (accessed March 2020)
[6] Jeff Langr "Modern C++ Programming with Test-Driven Development"; The pragmatic programmers; 2013; ISBN-13: 978-1-1937785-48-2
[7] VDA 702 Situationskatalog E-Parameter nach ISO 26262-3; VDA-Empfehlungen, 25. Juni 2015 (https://www.vda.de/de/services/Publikationen/vda-702-situationskatalog-e-parameter-nach-iso-26262-3.html) (accessed March 2020)
[8] Abbreviated Injury Scale: http://unfallanalyse.hamburg/index.php/ifu-lexikon/medizin/ais/ (accessed March 2020)
[9] Semantic Versioning 2.0.0: https://semver.org/spec/v2.0.0.html (accessed March 2020)
[10] Failure mode and effects analysis https://en.wikipedia.org/wiki/Failure_mode_and_effects_analysis (accessed March 2020)
[11] Fault tree analysis https://en.wikipedia.org/wiki/Fault_tree_analysis (accessed March 2020)
[12] What is Safety Element Out of Context (SEooC) in Automotive Functional Safety (ISO 26262) https://www.embitel.com/blog/embedded-blog/what-is-safety-element-out-of-context-seooc-in-automotive-functional-safety (accessed March 2020)
[13] Dave Hughes: Apply the ISO 26262 SEooC Model to Automotive Software (Aug 15, 2019): https://www.electronicdesign.com/markets/automotive/article/21808428/apply-the-iso-26262-seooc-model-to-automotive-software (accessed March 2020)
[14] ISO/SAE DIS 21434 "Road vehicles – Cybersecurity engineering" https://www.iso.org/standard/70918.html (accessed March 2020)
[15] Scrum-Guide https://www.scrumguides.org/scrum-guide.html (accessed March 2020)
[16] The Scrum Framework Poster https://www.scrum.org/resources/scrum-framework-poster (accessed March 2020)
[17] M. Islam et al., Deliverable D2 Security models. HEAVENS Project, Deliverable D2, Release 1. Dec. 2014
[18] Googletest - Google Testing and Mocking Framework https://github.com/google/googletest (accessed March 2020)
[19] Making a Pull Request https://www.atlassian.com/git/tutorials/making-a-pull-request (accessed March 2020)
[20] Unified Modeling Language https://www.uml.org/ (accessed April 2020)
[21] Catch2 https://github.com/catchorg/Catch2 (accessed April 2020)
[22] unittest – Unit testing framework https://docs.python.org/3/library/unittest.html (accessed April 2020)
[23] unitest.mock – getting startet https://docs.python.org/3/library/unittest.mock-examples.html (accessed April 2020)
[24] Sanitizer https://github.com/google/sanitizers (accessed April 2020)
[25] Valgrind http://valgrind.org/ (accessed April 2020)
[26] Bullseye https://www.bullseye.com/productInfo.html (accessed April 2020)
[27] CTC++ http://www.testwell.fi/ctcdesc.html (accessed April 2020)
[28] coverage.py https://coverage.readthedocs.io/en/latest/ (accessed April 2020)
[29] Clang Format https://clang.llvm.org/docs/ClangFormat.html (accessed April 2020)
[30] cppcheck http://cppcheck.sourceforge.net/ (accessed April 2020)
[31] Clang tidy http://clang.llvm.org/extra/clang-tidy/ (accessed April 2020)
[32] Pylint https://www.pylint.org/ (accessed 2020)

[33]Klocwork https://www.perforce.com/products/klocwork (accessed April 2020)
[34]Gitflow https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow (accessed May 2020)
[35]Conan, the C/C++ Package Manager for Developers https://conan.io/ (accessed May 2020)
[36]JFrog Artifactory – Enterprise Universal Repository Manager https://jfrog.com/artifactory/ (accessed May 2020)
[37]Reqtify https://www.3ds.com/de/produkte-und-services/catia/produkte/reqtify/ (accessed May 2020)
[38]When and how to qualify tools according to ISO 26262 https://www.btc-es.de/en/blog/when-and-how-to-qualify-tools-according-to-iso-26262.html (accessed May 2020)
[39]Kanban https://en.wikipedia.org/wiki/Kanban (accessed May 2020)
[40]Jira https://www.atlassian.com/software/jira (accessed May 2020)
[41]Confluence https://www.atlassian.com/software/confluence (accessed May 2020)
[42]BitBucket https://www.atlassian.com/software/bitbucket (accessed May 2020)
[43]Jenkins https://www.jenkins.io/ (accessed May 2020)
[44]Using JIRA for Requirements Management https://confluence.atlassian.com/jirakb/using-jira-for-requirements-management-193300521.html (accessed May 2020)
[45]SonarQube https://www.sonarqube.org/ (accessed May 2020)
[46]CMake https://cmake.org/ (accessed May 2020)
[47]Lit https://pypi.org/project/lit/ (accessed May 2020)
[48]OutputCheck https://pypi.org/project/OutputCheck/ (accessed May 2020)
[49]FileCheck https://llvm.org/docs/CommandGuide/FileCheck.html (accessed May 2020)
[50]ECU-TEST https://www.tracetronic.com/products/ecu-test/ (accessed May 2020)
[51]Financial Cost of Software Bugs https://medium.com/@ryancohane/financial-cost-of-software-bugs-51b4d193f107 (accessed May 2020)
[52]Error Cost EscalationThrough the Project Life Cycle https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/20100036670.pdf (accessed May 2020)
[53]Using LLVM LIT Out-Of-Tree https://medium.com/@mshockwave/using-llvm-lit-out-of-tree-5cddada85a78 (accessed May 2020)