PRISSMA Project
Plateforme de Recherche et d'Investissement pour la Sûreté
et la Sécurité de la Mobilité Autonome
04/2021 - 04/2024

[L7.4] WP7.3: SECURE DEPLOYMENT PROCESS OF UPDATES

WP6.3 : Preuves à fournir concernant le processus de développement, avec identification de la couverture possible d'exigences d'homologation

Main authors: JF MARLIERE (UTAC), E ARBARETIER (AIRBUS PROTECT), C GAVA (SPHEREA).

Keywords: Update, Maintenance …

Abstract. In standard circulating vehicles, updates are made at the dealership or at the manufacturer. The purpose of this release is to project the solutions that will have to be implemented for the operational process of updating a vehicle equipped with artificial intelligence.

Résumé. Dans les véhicules en circulation, les mises à jour se font en concession. L'objet de ce livrable est de projeter les solutions qui devront être mises en œuvre pour le processus opérationnel de mise à jour d'un véhicule doté d'une intelligence artificielle

Table of contents

1 INTRODUCTION

Ai-based systems provide new properties leading to new opportunities as well as to new challenges, such as complexity, robustness, verifiability or predictability. Yet they have to provide at the same time a certain level of safety and security than to the possibilities of system updates. The subject of the update is already present in current vehicles in circulation. As part of PRISSMA, we need to add the presence of artificial intelligence. It is necessary to take into account the disparity of vehicles equipped with artificial intelligence, notably the fact that the technologies used will not all be at the same level of maturity. Several areas need to be considered, such as technological level or cybersecurity. The goal will be to verify if a known process can be applied with the presence of artificial intelligence, knowing that the updating of this process will have to be done at a finer granularity level than just the presence of AI.

AI-based systems can contribute to improve vehicle safety, with additional beneficial consequences on road safety, e.g. by allowing AD systems to predict currently unforeseeable behavior of other road users (e.g. detection of potential collision opponents). The use of AI and machine learning algorithms in type approved functions is limited for the time being. Even though, there are already well-established processes for how to test conventional software before and during deployment of an automotive product those processes might not be sufficient for AI based software. Software must be tested prior to deployment in order to comply to all related Laws, Regulations, Recommendations and Policies. This applies also to the update process. However, it needs to be evaluated in how far current regulatory provisions can sufficiently address the specific needs for testing and updating AI based software and guarantee its safe operation.

The objective is to perform updates at the dealership or at an authorized maintenance center. The objective is not to perform an OTA (Over-The-Air) update.

Remark: Cyber security issues about following processes are not covered by this document:

- Remote loading protocol: this is covered by state-of-the-art remote software updating processes whatever application domain may be
- Robustness of interaction / exchange protocol with infrastructures (5G or supervision ones…): this is covered by WP5 tasks and deliverables and relate to impact / compatibility / non regression analysis like all other performances and functional contributions

2 DEFINITIONS

All the definitions below are excerpt from document UNR 156:

"Vehicle type" means vehicles which do not differ in at least the following:

(a) The manufacturer’s designation of the vehicle type.

(b) Essential aspects of the design of the vehicle type with respect to software update processes.

"RX Software Identification Number (RXSWIN)" means a dedicated identifier, defined by the vehicle manufacturer, representing information about the type approval relevant software of the Electronic Control System contributing to the Regulation N° X type approval relevant characteristics of the vehicle.

"*Software update*" means a package used to upgrade software to a new version including a change of the configuration parameters.

"*Execution*" means the process of installing and activating an update that has been downloaded.

"*Software Update Management System (SUMS)*" means a systematic approach defining organizational processes and procedures to comply with the requirements for delivery of software updates according to this Regulation.

"*Vehicle user*" means a person operating or driving the vehicle, a vehicle owner, an authorised representative or employee of a fleet manager, an authorised representative or employee of the vehicle manufacturer, or an authorized technician.

"*Safe state*" means an operating mode in case of a failure of an item without an unreasonable level of risk.

"*Software*" means the part of an Electronic Control System that consists of digital data and instruction.

"*Over-the-Air (OTA) update*" means any method of making data transfers wirelessly instead of using a cable or other local connection.

"*System*" means a set of components and/or sub-systems that implement a function of functions.

"*Integrity validation data*" means a representation of digital data, against which comparisons can be made to detect errors or changes in the data. This may include checksums and hash values.

AI based system: AI based systems refers to connectionist AI systems such as neural networks which are trained using machine learning algorithms and data.

3 UPDATES PRINCIPLES

3.1 Industrial update process

Examples of processes and aspects that are recommended to be documented by the manufacturer:

(a) Risk Management:

(i) Risk identification (in line with ISO 3100 6.4.2 or equivalent standard)

(ii) Risk analysis (in line with ISO 3100 6.4.3 or equivalent standard)

(iii) Risk evaluation (in line with ISO 3100 6.4.4 or equivalent standard)

(iv) Risk treatment (in line with ISO 3100 6.4.5 or equivalent standard), including

(v) Processes used for keeping the risk assessments as current as possible

(vi) Safety performance of the organization and effectiveness of safety risk controls.

Examples of processes and aspects that are recommended to be documented by the manufacturer:

(a) Safety governance:

(i) Safety policies and principles (in line with the concept stated in ISO 21434, para. 5.4.1 and ISO 9001 Automotive 5.2, but from safety perspective)

(ii) Management commitment (in line with the concept stated in ISO 21434, para. 5.4.1 and ISO 9001 Automotive 5.1, but from safety perspective)

(iii) Roles and responsibilities (ISO 26262-2, para. 6.4.2, this relates to the organizational as well as to the project dependent activities)

(b) Safety culture (ISO 26262-2, para. 5.4.2)

(c) Effective communications within the organization (ISO 26262-2, para. 5.4.2.3)

(d) Information sharing outside of the organization (in line with the concept stated in ISO 21434, para. 5.4.5 and ISO 9001, but from safety perspective)

(e) Quality management system (e.g., as per IATF 16949 or ISO 9001 or equivalent) to support safety engineering, including change management, configuration management, requirement management, tool management etc.

Examples of processes and aspects that are recommended to be documented to ensure the robustness of the design and development phase:

(a) A general description of the way in which the organization performs all the design and development activities

(b) Vehicle\system development, integration, and implementation.

(i) Requirements management (e.g. Requirement capture and validation)

(ii) Validation strategies, including but not limited to

a. Assessment of the physical testing environment

b. Credibility assessment for virtual tool chain

c. System Integration level

d. Software level

e. Hardware level

(iii) Management of functional Safety and operational safety, including the continuing evaluation and update of risk assessments and relationship with In-Service Safety

(c) Management of design changes and changes to design and development processes

Examples of processes and aspects that should be considered to assure that responsibilities are properly discharged:

(a) Roles and responsibilities during the design and development

(b) Qualifications and experience of persons responsible for making decisions affecting safety

(c) Coordination between design and production

Examples of processes and aspects that are recommended to be documented to ensure the robustness of the production phase include:

(a) Quality Management System accreditation (e.g., as per IATF 16949 or ISO 9001 or equivalent)

(b) A general description of the way in which the organisation performs all the production functions including management of working conditions and the environment and equipment and tools.

Examples of processes and aspects to be documented to assure robustness of distributed production:

(a) Liaison between the vehicle and/or ADS manufacturer and all other organisations (partners or subcontractors) involved in the production of the system/vehicle

(b) Criteria for the acceptability of "subsystem/components" manufactured by other partners or subcontractors. (i.e., deployment of production assurance requirements to supply chain)

Examples of processes and aspects that are recommended to be documented:

(a) Organizational policy for supply chain

(b) Incorporation of risks originating from supply chain

(c) Evaluation of supplier SMS capability and corresponding audits

(d) Processes to establish contracts, agreements for ensuring safety across the phases of development, production and postproduction

(e) Processes for distributed safety activities.

It is recommended that the manufacturer should provide an explanation of the design provisions built into "The System" to ensure functional and operational safety. Possible design provisions in "The System" are, for example:

(a) Fall-back to operation using a partial system.

(b) Redundancy with a separate system.

(c) Removal of the automated driving function(s).

4 SOFTWARE UPDATE REGULATION

4.1 Approval

Approval is applied by document checks and by the test of a vehicle of the vehicle type or by sampling. It must be checked whether the manufacturer has well implemented the SW update they have documented.

4.2 Certificate of compliance for the SW management system (SMS)

The manufacturer must provide with the following documents:

- Document describing the SMS.

4.3 SW update management system specifications

- The manufacturer must focus on the processes in place to manage SW updates.
- The manufacturer must ensure to update and maintain the SW versions, including initial version by a process to keep a possibility to track the SW versions. The manufacturer must be able to prove that the SW updates were conducted.
- The manufacturer must validate SW compatibility and interdependencies. A process to assess the impact of a SW update (on the components or the functionalities) must be implemented. A process to determine the compatibility of a vehicle with the new SW update must be implemented.
- The manufacturer must be able to give a process to detail the SW update (purpose of the update, functions updated. How the function is affected. The manufacturer must provide with condition for update application. Confirmation that the update was safely conducted and verified.
- The manufacturer must conceive the SWUMS in order to ensure that safety and security are preserved before performing the update (process to ensure that the update can’t be intercepted and modified before or during the transfer, process to ensure that the delivery and installation of the OTA update will not affect the vehicle safety)

4.4 Vehicle type specification

- A prerequisite is a valid SW update management system. We must ensure the possibility of managing the SW versions (by checking that their authenticity and integrity are protected)
- The manufacturer must consider the versioning of the SW updates (to consider new hardware and/or component modifications). A multi criteria compliancy must be implemented.

- The execution of the update must be done safely. We must ensure safety of the over the air updates by providing to the user information updates and restore the update in case of interruption.

5 SOFTWARE UPDATE MANAGEMENT SYSTEM REGULATION

5.1 Relevant information

Relevant information is documented and securely held and available to an approval Authority or its technical service upon request.

This information is provided by following contents:

- Design document:

Design specification of an entity must include:
- its designations
- its internal architecture
- a list of its processing units
- the function and role of each of these units

- Modelling requirements:

Design document of an entity must provide structure and connection graph of the entity including:
- use of procedures or specific tasks
- call for external services.
- access to variables in a modification or reading mode.

- Data description

Design document of a component must describe the interface of the component: functional definition of all its parameters transmitted by the component as well as those received from other components, with their format, tolerance interval and use domain. This description must be consistent with data dictionary.
Remark: in case of an AI based SW, Data Set used for Design, Validation and Evaluation have to be provided as well as their description and characterization.

- Exception

Internal and External interfaces of entities must not use exceptions even those being accepted by the language: errors (parameters getting out from their domain use) must not generate exceptions.

- CPU time and memory allocation

Design documentation must include an allocation process of CPU and memory resources dedicated to different tasks. This allocation must warranty correct operation of the software; design must schedule means to assess performances of the entity all over the project. These means can consist in benchmarking tasks, simulations sample cases, or RAMS analyses.

- Time dependency specification

Design document must describe for every entity possible interactions of different service uses: reentrance (or not), time exclusion between services (or in the contrary, possibility of concurrent processing). This description may improve architecture modularity and facilitate software changes.

- Synchronization mode specification

Design document must list synchronization devices and mechanisms used and justify them: semaphore, matching point, connection point…; they must be justified. System services used to support these synchronization processes must be described in an explicit way. Design document must emphasize choice logics and justification between synchronous and asynchronous mechanisms. Synchronization mechanisms based on execution time durations or priority levels is acceptable only for elementary components and must be strongly justified.

- Shared variables:

Direct or indirect use of common memory zones by different tasks (shared variables). If this use is associated by a protection mechanism (a semaphore for example), the aim of this protection must be detailed, the correction of the mechanism must be detailed, and its good operating must be checked.

- Modelling Language:

Design documentation must include a global model of the tasks as well as their synchronization, formalized in a rigorous modelling framework like UML, SART or SDL. This model is essentially descriptive, but it is highly recommended to add more detailed models which enable. Actual use of formal methods which can be enabled this way is complementary to classical development processes and can produce Front End verification tasks rather than Late ones.

5.2 Identification

Information regarding all initial and updated software versions, including integrity validation data, and relevant hardware components can be uniquely identified.

This information will have to comply with following rules:

- Designation rules

Every single entity must be named with a unique symbol which will be used in all phases of its life cycle. This symbol will be used also in all documents. It will be initialized at the beginning of the creation of the entity in one central repository.
The same naming process will be applied to all data exchanges by the entities.

- Name constitution

The symbol or character string will be as clear, relevant and expressing as possible. Underscore symbol will be used to separate different text strings.
Dictionaries will list for all types of entities the different names used and their definition: programs, modules, functions, procedures, tasks, data…

- Data Dictionary

In the data dictionary, for all data will be displayed:
- Its symbolic name,
- Its localization (global or specifically local for one entity)
- Its type (integer, real, structure, etc.),
- Its initialization value (or default value),
- Its definition, role, or usual meaning
- Its unit in case of physical asset
- Its use domain

5.3 Accessibility and update information

Accessibility and update capacity should be guaranteed before and after the update: it concerns what is called Software Baseline to warrantee steadiness and stability of the software.

Software Baseline is defined by IEEE (IEEE Std. No. 610.12-1990) as “an agreed definition and review of product qualities that serve as a foundation for further development and defining change, and this modifying can only be done through formal change control processes.”

A baseline is a milestone and a reference point in software development that is indicated by the completion or delivery of one or more software configuration items, as well as a formal technical assessment of a set of predetermined products. The baseline is a project database that is shared. It is the responsibility of Software Configuration Management (SCM) to ensure the integrity of a group of products.

The main goal of the baseline is to decrease and regulate vulnerability or project weaknesses that may easily damage the project and lead to uncontrollable changes. This may be accomplished by correcting and altering configuration items (a variety of important deliverables) at critical stages during the product development life cycle. Every element linked to the baseline has to be kept under formal change control.

Briefly summarized, a baseline is a snapshot of the source code tree at any point in time that is static (i.e. unchanging). The baseline’s goal is to provide a constant reference point for changes that occur after the baseline is established. And furthermore, the process of testing the baseline itself assures operational capability and performance assurance of the baseline.

Baseline testing is therefore the validation of the documents and specifications that test cases are based on. In general, a baseline is a standard that serves as the foundation for any new production. This is a term used in software testing to describe benchmarking an application’s performance. During baseline testing, several issues are detected and resolved.

In other words, the practice of analyzing a system or component to identify its initial performance and characteristics is known as baseline testing. It aids in the establishment of a "normal" working condition that may be compared to future test findings. This enables businesses to spot changes in performance or behavior that may signal a problem or issue.

Anything from software programs to network infrastructure may be tested as a baseline. Such tests are usually done before a system goes into production to make that it satisfies all of the criteria and works as intended. When problems arise, baseline testing can also be utilized to troubleshoot them.

Baseline testing is used to determine if the applications or software's performance has deteriorated over time as a result of modifications. If the findings demonstrate no change in performance, the data is then utilized to develop metrics that may be used to determine the application's baseline performance in the future.

If the client requirement is not correctly baselined, the information in subsequent papers developed based on the business requirement document will be erroneous. The development team will begin work with the wrong requirements, and the testing team will test the improperly built functionality as well. During user acceptance testing, a problem will arise, and the product generated will not meet the expectations of the client.

To put it another way, baseline testing ensures that the program or application under test provides a consistent user experience under all test conditions and after any modifications.

The baseline test assesses the product's performance and creates a benchmark against which future comparisons may be made.

Such testing verifies that the requirement document produced meets the client's needs and expectations. Any variation from the norm might lead to serious problems in the future. If done correctly, this testing can result in high-quality products that arrive on time or ahead.

Baseline testing is carried out to guarantee that the product's performance does not suffer as a result of the new alterations in the application. To ensure that the performance meets the baseline requirements, appropriate steps should be taken. These steps will result in the benefits listed below.

The following are some of the advantages of using the baseline testing method:

- It resolves the challenges that arise throughout the requirement collection process.
- It aids in the establishment of a performance baseline for the product.
- It finds and fixes any problems with performance measurements.
- It helps improve the software's quality.
- Configuration errors can be found easily.
- This testing type helps to find the blockages for different performance criteria.
- It helps to increase the tester's efficiency.
- Automation gets easier as, rather than writing multiple statements, only one needs to be checked.
- Helps to ensure that all the scenarios covered have no impact on the user experience.

5.4 Software versions

The vehicle manufacturer can verify that the software versions present on a component are consistent with those present on the RXSWIN.

Version Management Process must comply with referenced Software Configuration Management System.

In software engineering, software configuration management (SCM or S/W CM; also expanded as source configuration management process and software change and configuration management) is the task of tracking and controlling changes in the software, part of the larger cross-disciplinary field of configuration management. SCM practices include revision control and the establishment of baselines (see previous §). If something goes wrong, SCM can determine the "what, when, why and who" of the change. If a configuration is working well, SCM can determine how to replicate it across many hosts.

The goals of SCM are generally:

- Configuration identification - Identifying configurations, configuration items and baselines.
- Configuration control - Implementing a controlled change process. This is usually achieved by setting up a change control board whose primary function is to approve or reject all change requests that are sent against any baseline.
- Configuration status accounting - Recording and reporting all the necessary information on the status of the development process.
- Configuration auditing - Ensuring that configurations contain all their intended parts and are sound with respect to their specifying documents, including requirements, architectural specifications, and user manuals.
- Build management - Managing the process and tools used for builds.
- Process management - Ensuring adherence to the organization's development process.
- Environment management - Managing the software and hardware that host the system.
- Teamwork - Facilitate team interactions related to the process.
- Defect tracking - Making sure every defect has traceability back to the source.

Remark:

With the introduction of cloud computing and DevOps the purposes of SCM tools have become merged in some cases. The SCM tools themselves have become virtual appliances that can be instantiated as virtual machines and saved with state and version. The tools can model and manage cloud-based virtual resources, including virtual appliances, storage units, and software bundles. The roles and responsibilities of the actors have become merged as well with developers now being able to dynamically instantiate virtual servers and related resources.

5.5 Interdependencies of the updates

Any interdependencies of the updated system with other system can be identified.

These interdependencies will be issued from an impact analysis.

There are three primary methodologies for assessing the impact of changes on the functioning of a business, product, or process.

- Traceability impact analysis: Traceability analysis is a common component in software development used to discover the links between different product components. This type of analysis prioritizes consistency, ensuring that all the elements of a change align with each other, and meet all requirements.
- Dependency impact analysis: In contrast to traceability impact analysis, this method of analysis studies dependencies—the effects a change could have on an entire system. For example, a company considering changing its distribution could run a dependency analysis to study whether the change would interfere with supply chain operations.
- Experimental impact analysis: Experimental impact analysis looks at the effects of past disruptions to predict the impact of future changes. Following this method, a company analyzes historical data to see how certain decisions caused problems in the past, effectively planning for the potential problems of a future decision.

For example, to carry on a relevant impact analysis process, one will have to follow these steps:

- Gather a team. Prepare for the process by gathering key stakeholders related to all the information regarding the proposed change, including project managers, product managers, software engineers, developers, designers, or senior management. The exact makeup of the team will depend mainly on the proposed change and the type of company but consider the different types of impact a change or business disruption could have and involve relevant team members.
- Brainstorm: Consider the proposed change, then think about which areas of business operations it could affect. For example, consider how the change may impact the company's business strategy, sales and marketing goals, and internal business processes. List all possible workstreams that the change may affect. Gather information by getting input from department heads or having project team members answer questionnaires.
- Evaluate:

Once you've gathered all the relevant information, identify the benefits and consequences of the proposed changes. If you're conducting an impact analysis of potential unplanned disruptions such as a power outage or software malfunction, collect information on how this would affect output and possible downtimes and evaluate mitigation and recovery strategies. Unlike a risk assessment plan, an impact analysis report of this kind should focus on specific breakdowns and not general unplanned disruptions like natural disasters and regulatory changes.

- Report findings:

Report the findings of the potential impact of changes in an impact analysis report. Present the findings using an analysis model, such as a decision tree or influence diagram. Presenting a visual representation of the findings helps team members visualize the total impact of the proposed change.

- Make adjustments:

Based on the report's findings, adjust the proposed change. Adjustments can include contingency plans in case any of the estimated problems occur.

- Implement the change:

Finally, your team is ready to make the proposed change. Monitor the performance of the change and prepare to intervene if problems arise.

Remark:

There are several notable reasons why teams weighing a significant decision should perform an impact analysis.

- Minimizes loss: Impact analysis is a form of risk management—it helps teams mitigate potential losses due to significant changes within an organization, process, or product.
- Prepares for problems: Knowing that problems could arise is not always a reason to refrain from making a change. However, this analysis does help businesses prepare for tackling potential problems should they occur.
- Encourages team buy-in: Significant changes in a business environment can disrupt team harmony. Impact analysis allows team members to come together to assess the impact of a change and plan for its implementation.

5.6 Targets vehicle

The vehicle manufacturer has to identify target vehicle for a software update.

Criteria to decide and validate software update are provided by System Configuration Management and more precisely Work Breakdown Structure Management of all vehicles potentially concerned by a software update.

5.7 Compatibility of a software update version

One must confirm the compatibility of a software update with the target vehicle configuration before it is issued through an assessment of the last known software/hardware configuration of the target vehicle.

Several types of software compatibility testing can be performed, including for example:

- Operating system compatibility testing checks whether the software works correctly on different versions of the operating system, such as Windows, Linux, or macOS.
- Browser compatibility testing ensures the software works correctly on different web browsers, such as Chrome, Safari, Firefox, or Edge.
- Mobile device compatibility testing ensures the software works correctly on mobile devices like smartphones and tablets with various operating systems, screen sizes, and resolutions.
- Hardware compatibility testing checks the software's performance with different hardware configurations, such as CPU instructions or hard RAM requirements.
- Network compatibility testing ensures that the software works correctly with different network configurations, such as different types of routers, firewalls, and proxies.
- Database compatibility testing checks whether the software works well with databases like MySQL, Oracle, or Microsoft SQL Server.
- Third-party application compatibility testing ensures the software works correctly with different third-party applications like antivirus software or other security tools.

Apart from these, there are 2 types of compatibility testing in QA: forward compatibility testing checks whether the software is compatible with future versions, while backward compatibility testing checks whether it is compatible with older versions of hardware, software, or network configurations.

When performing compatibility testing, following stages have to be validated:

- Identify the target environment for the software application by analysing the various hardware, software, and network configurations on which the software will be used.
- Create a test plan that outlines the scope and objectives of the compatibility testing, identifying the specific tests like functional testing, performance testing, and user acceptance testing.
- Implement automation by selecting appropriate tools and frameworks to automate the execution of compatibility tests. This effort includes automated configuration setup, test script creation, and result analysis.
- Execute the tests by installing and configuring the software application in the target environment and running compatibility tests to identify any issues or conflicts.
- Troubleshoot and resolve issues by making changes to the software application or the target environment to ensure the software is fully compatible.
- Document and report the results to the relevant stakeholders by creating a detailed report of the testing process that includes issues or conflicts and the steps taken to resolve them.

There are several challenges that software development teams may encounter during the process of compatibility testing:

- Diverse configurations: With the increasing number of configurations in the market, it can be challenging to perform compatibility testing with all possible hardware and software configurations.
- Limited resources: Testing a software application for compatibility requires significant resources, including hardware, software, and testing expertise, which may not be available to all teams.
- Time constraints: QA compatibility testing can be time-consuming, especially when dealing with large and complex software applications, making it harder to perform in the intended timeframe.
- Dynamic environments: Software environments are constantly changing with new updates and upgrades affecting the compatibility that can be hard to keep up with. Minimizing the impact requires tools and techniques for automatic environment provisioning, configuration management, and version control.
- Technical expertise: Compatibility testing requires vast technical expertise in hardware, software, and networking that not all software development teams may have.

Remark:

Performing compatibility testing during the software development lifecycle, particularly during the testing phase, is crucial for identifying and resolving compatibility issues before the software release. This way, one can initiate compatibility testing as early as possible in the development process to proactively address any potential problems with different hardware, software, and network configurations. By conducting compatibility testing early, one can minimize the risk of last-minute surprises and ensure a smoother and more successful software launch. One should also perform compatibility testing whenever one release a new software version or change the system environment.

Here are some tips for conducting successful software compatibility testing:

- Develop a comprehensive compatibility testing plan that covers all possible hardware and software configurations to ensure accurate results.
- Use automation-minded bug-tracking tools that can help you save time and resources, reduce errors, and increase test coverage.
- Test on real-world configurations that users may encounter to ensure the software application performs as expected.
- Prioritize critical configurations that users will most likely use to ensure the software application is fully compatible with the most important configurations.
- Test early and often to identify compatibility issues early on, reducing the risk of expensive and time-consuming fixes later.
- Involve a diverse testing team with varying levels of technical expertise to identify compatibility issues from different perspectives and increase the chances of finding critical issues.
- Keep up with software updates and changes to ensure the software application remains compatible with the latest technologies.

5.8 Affect approved systems.

One has to assess, identify and record whether a software update will affect any type approved system. This shall consider whether the update will impact or alter any of the parameters. Such assessment may be performed through FMECA (Failure Mode Effect and Criticality Analysis) or its adapted format for software SEEA (Software Error Effect Analysis).

Slight differences are found between different existing FMECA standards: FMECA analysis process typically consists of the following logical steps:

- Define the system.
- Define ground rules and assumptions to help drive the design.
- Construct system block diagrams.
- Identify failure modes (piece-part level or functional)
- Analyze failure effects/causes.
- Feed results back into design process.
- Classify the failure effects by severity.

- Perform criticality calculations.
- Rank failure mode criticality.
- Determine critical items.
- Feed results back into design process.
- Identify the means of failure detection, isolation and compensation.

Perform maintainability analysis.

- Document the analysis, summarize uncorrectable design areas, identify special controls necessary to reduce failure risk.
- Make recommendations.
- Follow up on corrective action implementation/effectiveness.

FMECA may be performed at the functional or piece-part level. Functional FMECA considers the effects of failure at the functional block level, such as a power supply or a software module. Piece-part FMECA considers the effects of individual component failure or errors. A piece-part FMECA requires far more effort but provides the benefit of better estimates of probabilities of occurrence. However, Functional FMEAs can be performed much earlier, may help to better structure the complete risk assessment, and provide other type of insight in mitigation options. The analyses are complementary.

The criticality analysis may be quantitative or qualitative, depending on the availability of supporting part failure data.

System definition

In this step, the major system to be analyzed is defined and partitioned into an indented hierarchy such as systems, subsystems or equipment, units or subassemblies, software modules and piece-parts. Functional descriptions are created for the systems and allocated to the subsystems, covering all operational modes and mission phases.

Ground rules and assumptions.

Before detailed analysis takes place, ground rules and assumptions are usually defined and agreed to. This might include, for example:

- Standardized mission profile with specific fixed duration mission phases
- Sources for failure rate and failure mode data
- Fault detection coverage that system built-in test will realize.
- Whether the analysis will be functional or piece-part
- Criteria to be considered (mission abort, safety, maintenance, etc.
- System for uniquely identifying parts or functions.
- Severity category definitions
- Assumptions for error type definition for software modules

Block diagrams.

Next, the systems and subsystems are depicted in functional block diagrams. Reliability block diagrams or fault trees are usually constructed at the same time. These diagrams are used to trace information flow at different levels of system hierarchy, identify critical paths and interfaces, and identify the higher-level effects of lower level failures.

Failure mode identification

For each piece-part, each software module or each function covered by the analysis, a complete list of failure modes is developed.

For functional FMECA, typical failure modes include:

- Untimely operation
- Failure to operate when required.
- Loss of output
- Intermittent output
- Erroneous output (given the current condition)
- Invalid output (for any condition)

5.9 Affected functions.

One will assess, identify, and record whether a software update will add, alter or enable any functions that were not present or enabled.

For this purpose, an SEEA (Software Error and Effect Analysis) will be performed, as a transposition of FMECA to softwares.

Correct and consistent application of previous rules should be proofed in a SEEA analysis where:

- for specific performance criteria considered as critical for the application (safety, security, availability, time response…),
- for entities sensitive as to or especially involved in the production of this performance,
- for all types of errors concerning this entity (algorithms, data formatting or transmission, synchronization, …)
- for every propagation levels of the errors,

… all effects will be precisely described in terms of Functional Loss or Degradation, Physical Damage, or Data Alteration or Suppression.

In all cases Compensating Provisions, Corrective Action concerning Specification or Coding process of the software, or Detailed Recommendations and Actions targeting Test and Validation tasks will be produced.

All these actions will be assessed in terms of Costs and Workload in a Reference List, which will be updated all over the project; real application of these actions will be traced and followed up in the framework of a Quality Board, so that they may not affect delivery milestones of the project.

SEEA report will be produced and will integrate information listed above: SEEA tables will be displayed in Worksheet Formats and a specific part will be dedicated to the action management process.

SEAA tables will include functions impacted by contributions of software modules, in case of their update.

5.10 Safety

One will assess, identify and record whether a software update will affect:
- Any other system required for the safe and continued operations.
- Or add functionality of the vehicle compared to when it was registered.

Attention and effort dedicated to software update will be apportioned to severity of risk able to be generated due to this update: therefore this will be related to a risk analysis applied to the system integrating this software update; every time an update is applied to a software module being part of a system presenting safety critical functions (like a transportation system), every time safety assessment report of this system has to be consequently updated.

For this purpose, Safety Analysis may include Preliminary Hazard List (PHL), Preliminary Hazard Analysis (PHA), Failure Mode and Effect Analysis and its variants, Functional Failure Analysis (FFA), Hazard and Operability Analysis (HAZOP), Fault Tree Analysis (FTA), Event Tree Analysis (ETA).

Typical Safety Analysis process may consist of:
- System model analysis
- Hazard and Operability Analysis (HAZOP) and Functional Failure Analysis (FFA)
- Hazard analysis
- Failure Mode and Effect Analysis (FMEA): see previous §
- Fault Tree Analysis (FTA)

Hazards and risk

A hazard is defined as a "Condition, event, or circumstance that could lead to or contribute to an unplanned or undesirable event." Very rarely does a single hazard cause an accident or a functional failure. More often an accident or operational failure occurs as the result of a sequence of causes. A hazard analysis will consider system state, for example operating environment, as well as failures or malfunctions.

While in some cases, safety or reliability risk can be eliminated, in most cases a certain degree of risk must be accepted. To quantify expected costs before the fact, the potential consequences and the probability of occurrence must be considered. Assessment of risk is made by combining the severity of consequence with the likelihood of occurrence in a matrix. Risks that fall into the "unacceptable" category (e.g., high severity and high probability) must be mitigated by some means to reduce the level of safety risk.

IEEE STD-1228-1994 Software Safety Plans prescribes industry best practices for conducting software safety hazard analyses to help ensure safety requirements and attributes are defined and specified for inclusion in software that commands, controls or monitors critical functions. When software is involved in a system, the development and design assurance of that software is often governed by DO-178C. The severity of consequence identified by the

hazard analysis establishes the criticality level of the software. Software criticality levels range from A to E, corresponding to the severity of Catastrophic to No Safety Effect. Higher levels of rigor are required for level A and B software and corresponding functional tasks and work products is the system safety domain are used as objective evidence of meeting safety criteria and requirements.

Fault tree analysis (FTA) is a type of failure analysis in which an undesired state of a system is examined. This analysis method is mainly used in safety engineering and reliability engineering to understand how systems can fail, to identify the best ways to reduce risk and to determine (or get a feeling for) event rates of a safety accident or a particular system level (functional) failure. FTA is used in the aerospace, nuclear power, chemical and process, pharmaceutical, petrochemical and other high-hazard industries; but is also used in fields as diverse as risk factor identification relating to social service system failure.9 FTA is also used in software engineering for debugging purposes and is closely related to cause-elimination technique used to detect bugs.

In aerospace, the more general term "system failure condition" is used for the "undesired state" / top event of the fault tree. These conditions are classified by the severity of their effects. The most severe conditions require the most extensive fault tree analysis. These system failure conditions, and their classification are often previously determined in the functional hazard analysis.

5.11 Updates information

A process whereby the vehicle user can be informed about updates, may include following steps able to assure a successful software update:

There are, however, a set of steps and considerations an organization would typically undertake to perform a successful software update:

1. Identify and commit Stakeholders

Remember to include both internal and external stakeholders. End users, trainers, content developers, administrators, support and hosting teams, QA and testing teams, system owners, external suppliers, and service partners. Work out how each group interacts with the system, how an upgrade will affect them, and what level of ongoing engagement they might require.

2. Undertake Change Control Planning

Besides the technical work involved in upgrading your system, effective change control will probably be the biggest challenge surrounding a software upgrade. Keeping the balance between adequate communication and consultation, and overwhelming stakeholders with detail can be difficult. Consider assigning a dedicated Change Control Manager to the process and include developing a communication plan and (at a minimum) a high-level project plan as deliverables for that role.

3. Perform A Site Review

Performing a site review involves taking a stockage of any and all customizations within the site. Identify all bespoke developments and dig out the specifications and other relevant documentation relating to this work. Assess each development for upgrade or

reimplementation. For this purpose, following questions may be asked: Is this customization in the core system or has it been developed as a plugin? Is this feature (or something very similar) now available in the new software version? Is the current development or the business need for this feature still applicable? Is your software integrated with any other systems or applications?

4. Establish Upgrade Requirements

Based on our site review, do we need to re-develop customizations and integrations or upgrade bespoke plugins? Do we need to make allowance for historical data from those customizations? Can we upgrade to the latest version of the software from your current version or do you need to upgrade to a more recent version first? Check the technical specifications of the new software—do we need to upgrade, install, or purchase anything as part of the upgrade? Do you we need (or want) to upgrade your theme/design or will it work on the upgraded version?

5. Make An Upgrade Plan

Who will perform the upgrade? What needs to be upgraded? What needs to be redeveloped? Does anything need to be newly developed? How will the upgrade be performed? Is there any guidance on the upgrade from the software supplier? What time, people, resources, and budget do you need to apply to the upgrade process?

6. Make A Roll-Out Plan

When will the upgrade be performed? Will there be an outage? How long will it be? What sort of message needs to go out to stakeholders? What will be the implementation process? What is scheduled rollback plan?

7. Backup Everything

Store that backup somewhere very safe and secure. Backup again, and store that file somewhere else safe and secure.

8. Run A Trial Upgrade

The best way to know if any issues will crop up during the upgrade is to try upgrading a copy of the live site. It will identify any code and/or database conflicts and where further work will be required. It’s important to allow plenty of time for this step and to engage with relevant stakeholders.

9. Upgrade A Staging Site

Once the technical aspect of the upgrade has been completed and any issues resolved, it’s time to upgrade a copy of the live site to a staging or testing environment. Remember to switch off any email processes so users aren’t messaged as part of any testing and make sure the staging site is only available to permitted users.

10. Perform Testing

This includes functional, non-functional, and technical testing, quality assurance, data integrity, security, performance, and theme/design testing including browser, and device testing. Vital to the testing process is a shared mechanism for capturing and tracking issues. Issues should be described in detail, have a unique identifier for tracking purposes, have a current status, and be allocated to someone until fully resolved. It is also important to allow time for, and manage expectations around, user acceptance testing. While an upgraded system might be functional, it might not fulfill all of the business requirements it was implemented to address.

11. Undertake Configuration

With freshly upgraded staging site now deployed, tested and issues resolved, site administrators will need some time to ensure any new features and functionality introduced by the updates are configured correctly. This might involve turning on or off features or updating settings to best accommodate your users.

12. Prepare Documentation

Preparing help files or user manuals, as well as technical documentation detailing any custom developments or deployment requirements is vital to the success of your current upgrade as well as any future upgrades. Leverage any documentation provided by the software supplier or your technology vendor and highlight the differences between versions to emphasize important changes in the new release.

13. Provide Training

Training the right people at the right time not only requires adequate training resources but adequate planning. Trainers and the trainees need to be available at the same time, in a suitable space. The amount of training, the type of training, and the length of training will depend on the end-user facing functionality and feature changes within your upgraded software. For some helpful hints on selecting a training provider, check out this article.

14. Prepare Infrastructure

If upgrade introduces a number of significant changes or one is timing the software upgrade with a re-release of the system, one will need to ensure hosting setup will be able to cope with an increase in access and usage—if only for a short period of time.

15. Go Live / Fall Back

One will have a few options around go-live including a complete switchover or running the old system and updated system in parallel for a period of time. Refer back to rollout plan to make sure everyone knows their role and responsibilities for the release.

16. Offer Support

There will probably be an initial demand on support resources after an upgrade has been released as users become acquainted with the new system. Changes to ongoing support might be required as a result of new system features and functionality or changes to processes and policies. Ensure the support team has enough documentation, training, and coverage to meet demand going into the upgrade release. For some helpful hints on selecting a supporting vendor, check out this article.

17. Perform An Upgrade Review

It is important to review the upgrade project in retrospect to identify what worked well and where improvements can be made for the next time an upgrade is required.

5.12 Documentation

One has finally to make the information of the software versions and their integrity validation data for all relevant software and document listing target vehicles for the update and confirmation of the compatibility available to responsible authority or the technical services.

All relevant contents have been listed in previous § 5.1, 5.3, and 5.4.

REFERENCES

UN Regulation No. 156 - Software update and software update