REPLY CONCEPT

EMBEDDED SOFTWARE DEVELOPMENT

LESSONS LEARNED FROM FUNCTIONAL SAFETY STANDARDS

Developing a robust embedded system when minimizing time to market is not an easy task. If issues arise within an embedded solution after the rollout, they can be catastrophic and might not be fixable without direct access to the devices. To avoid that, various practices need to be established for the development process.

When developing safety critical systems functional safety standards are defined. There are several standards to choose from – for example ISO26262 for road vehicles, IEC61508 for different industries and DO-178B/DO-178C in avionics. However, for non-safety critical product development, there is no universal approach in order to ensure product quality.

Using the full functional safety standards for non-safety critical embedded products is not nescessary. Nevertheless, it makes sense to look into them when setting up design guidelines to implement general approaches. In the following the important parts of the ISO26262 standard for road vehicles will be examined.

THE STRUCTURE OF ISO26262

ISO26262 outlines the whole lifecycle of the product development. It starts with the management of functional safety and goes from concept phase over development and supporting processes into production, operation and decommissioning.

Part 2 of the standard is the management part. It touches upon organizing the team to handle additional safety topics properly and enhance development procedures. This part suggests that the project should have a person responsible for the implementation of the relevant methodologies. This is usually the team lead, project manager or the scrum master.

Part 3 refers to the concept phase and hazard and risk analysis, which is basically looking for possible bottlenecks and troublemakers in the system. This is a part which also normally has to be done during initial requirements analysis and development planning.

Parts 4, 5 and 6 detail the product development phases - system, hardware and software development, respectively. Each system is assigned the safety grade, which will affect how many safety mechanisms have to be implemented, while in normal development this is more of a common sense approach depending on what robustness goals are set and how many resources have been allocated to achieve that.

Part 8 describes supporting processes, meaning selecting the right tools for the development tasks. This would be picking toolchains, additional tools and frameworks for development as well as for verification and other means of providing the software quality, deciding on the needed infrastructure.

The next paragraphs are focusing on interesting parts of the ISO standard and switch to commonly implemented practices and recommendations.

DEVELOPMENT ORGANIZATION AND SUPPORTING PROCESSES

At this point ISO26262 specifies the safety culture and organization, which is implemented through common development practices.

1. DOCUMENTATION. Usually from the start of the project a set of initial documents, for example lining out common and specific requirements, is present in some form. These and all other documents should be stored away reliably and easily accessible for all the developers. This is a highly important issue: Throughout the whole development process as much documentation as possible should take place. Meeting notes, feature and design descriptions, flowcharts and dataflows – any script created for the project should be entered into a documenting software (confluence, doxygen or any other suitable program) and kept for reference. An example for easy handling is to include the generation of doxygen docs as one of the build targets and not allow a release if the description is insufficient.

2. GENERAL PRACTICES. All of the processes should ideally be standardized and introduced from the very beginning of the project, since it may require significant effort to change and adapt those during development. Implemented practices should be aimed at catching a maximum amount of bugs at the compilation stage through all kinds of tests, leaving less and less bugs for the next stages. The reason is that the price of testing activities increases, compile time being the cheapest.

Choosing a development organization the popular approach nowadays is the agile way. It makes sense to discuss and introduce additional conventions such as code reviews, establishing coding rules and guidelines, merging rules and similar. Code review is usually mandatory, as it is very hard for one person to spot all potential issues. The sources should be looked into by as many eyes as possible.

3. SYSTEM DESIGN. The system design takes place from the start of the project and then continues well into development. It starts from the system architecture and proceeds for each feature to come. Features as well as designs should be properly described and agreed between team members and the development plan should be the resulting artifact. While making development plans it is necessary also to plan for testing, verification and validation activities. Here come into play also such concepts as security by design, because the flaws on this stage might result in great issues later.

4. BUILD SYSTEM, CI/CD. The key takeaway from this point is to not rely on manual builds. The integrated development environment (IDE) provided by the chip manufacturer often is not the best choice, as it gives little flexibility in automating the tasks described above. If possible, introducing a customized build system is recommended, especially in larger projects. Nowadays, in the era of containers and quality management with easy to set up backends, processes for Continuous Integration or Continous Deployment (CI/CD) should be included. Even for small projects, the whole development cycle should be kept uniform and, based on some preconditions, automated as far as possible. Preferably the environment is set up on a dedicated server.

Additional analysis tools are of great help – static and dynamic code analysis tools or code style checkers are available in many variations, from commercial to open-source and free implementations. As these tools are easy to include, they should be used as much as possible.

The means to provide delivery packages should be as automated as possible. As there will be a lot of software testing, sometimes also with external companies, all participants should have access to the same tools and information. Error detection as well as error resolution need to be fast: Tracking of issues, implementation of fixes and similar tasks should have high priority and be easy to follow up on via continued documentation such as needed logs or debugging information.

All in all, the methods that can be afforded should be implemented – many tools are available depending on the language and project type but mostly as automated and effortless as possible. The easier it is to use the safety ensuring mechanisms, the more effective they will be.

THE SOFTWARE DEVELOPMENT PROCESS

In this part ISO26262 covers types of failures which are not relevant for normal development processes. But also code coverage, detecting issues in the code, diagnostic capabilities, verification methods and testing software is lined out.

For preventing issues in the code, common practices come into play. Some of them are also outlined in ISO26262:

- A function should perform only one conceptual task.
- Functions should be logically complete. If there is a need to put the conjunction 'and' into the name

it is time to split it in two.
- There should be as little duplicate code as possible. If a part of the code is repeated two or more times, a separate function should be created.
- At the header of the source no magical constants should be declared as values.
- From the system as much output as possible should be available. Even if it is not feasible to store the logs, transparency of the system behavior during testing and debugging is a must.
- All functions with outputs should be checked for errors. Even if execution flow cannot recover from the errors – any issue should always be visible in logs or debug output.

In general close attention should be paid to dynamic allocation areas. The common recommendation is to limit the usage but if it is required, there are many tools available to help monitor or at least check the heap usage.

Besides these common practices, ISO26262 additionally outlines highly recommended principles, that must be complied with. Basically they are the same as the ones provided by MISRA-C:

- One entry and one exit point in functions
- No dynamic object or variables
- Initialization of all the variables
- No multiple use of variable names
- Avoid global variables or justify their usage
- Limited usage of pointers
- No implicit type conversions
- No hidden data flow or control flow
- No unconditional jumps
- No recursions

Further pieces of advice: Optimization should be the last stage of development and only take place where it has a significant effect.
Best practices should be easily addable – libraries and standard approaches should be preferred over customized ones.
A lot of code often can be reused from project to project with little changes. The disadvantage of this is that bugs migrate as well.
Debugging tools as well as hardware should be available at an as early stage as possible, even if it will not be the final hardware.

Lastly and most importantly: Security should always be taken into account. In the GDPR era internal data must be protected as widely as possible. This, among other measures, means usage of security areas in the chip, locking down the firmware, signing the firmware packages used for updates. For the production build normally as much output as possible is disabled. However, if secure mechanisms for system status and error reporting can be included, they should be.

TESTING

Testing is usually one of the largest parts of the development process. It is separated into the general types Acceptance Testing, System Testing, Integration Testing and Unit testing.

For catching bugs as many cheap ways as possible should be used. To note here is the fact that the quality of software increases with each level of testing.

CHALLENGES. Testing embedded software brings a number of challenges:

- Connectivity/no connectivity
- Observing internal states
- Developing special software applications for embedded systems - to provide stimulus and capture response
- Uniform solutions are usually not available and thus developed per case
- High level of hardware dependency; As software is often developed simultaneously with the hardware the hardware is not immediately available
- Defects are harder to reproduce in embedded systems because there can be high levels of interconnections, for example firmware dependencies on other parts of the systems. (Recommended solution: a part of the system can

be simulated, and a part can be postponed to later implementation)

ORGANIZATIONAL COUNTER ACTIONS. Test racks and test devices might be quite big or limited in quantity and thus not always available for all team members. In that case, parallel work can be organized by providing remote access to devices and coordinating time slots. It is also beneficial to have automated test setups.

In non safety critical systems focus on coverage does not need to be intense, but rather on identifying and covering critical parts. Efforts should also go into making testing and debugging cycles shorter. This will allow catching more errors and introducing more ways of testing. If possible, self-tests should be implemented. Various simulation-based testing approaches – model-in-the-loop, hardware-in-the-loop, software-in-the-loop, processor-in-the-loop, and so on – should be regarded. When making a release, generally smoke tests should be performed. Last but not least testing can cover non functional testing aspects like security and load testing.

FURTHER READING

1. ISO26262 standard for functional safety

2. https://embeddedartistry.com/blog/2018/04/26/embedded-rules-of-thumb/

CONCEPT REPLY

Concept Reply is the software development partner of the Reply Group specialised in IoT innovation and offers solutions for its customers in the areas of Smart Infrastructure, Industrial IoT and Connected Car from the idea through the concept phase to implementation, operation and support. With around 50 IoT specialists, the range extends from implementation in the embedded environment to gateway software or cloud applications.