

# Industrial Validator Whitepaper Magneti-Marelli

<b>Nature:</b>	Report		
<b>Dissemination Level:</b>	Public		
	<b>Distribution to Participants</b>	<b>Additional Distribution</b>	
	ALL INTERESTED companies represented	<a href="mailto:interested_all@interested-ip.eu">interested_all@interested-ip.eu</a> www.interested-ip.eu	
<b>DocID:</b>	INTERESTED_whitepaper_mag_final	<b>Creation Date:</b>	18/5/2011

<b>Project</b>	INTERESTED	<b>Contract Number</b>	214889
<b>Author</b>	Giacomo Gentile	<b>Organisation</b>	MAG

<b>Project</b>	INTERESTED	<b>Contract Number</b>	214889
<b>Internal Reviewers</b>	Marek Jersak Marco Di Natale Eric Bantegnie	<b>Organisation</b>	SYM EVI EST

# 1 Table of Contents

<b>1</b>	<b>Table of Contents .....</b>	<b>2</b>
<b>2</b>	<b>Motivation for the MAG Validator .....</b>	<b>3</b>
2.1	Overview.....	3
2.2	Market Requirements.....	3
2.3	Timing Analysis Advantages.....	4
2.4	Cost-Benefits Estimation.....	4
2.5	Process and Tools.....	4
<b>3</b>	<b>Technical implementation of the MAG Validator.....</b>	<b>6</b>
3.1	Current Approach .....	6
3.1.1	Weaknesses of the Current Approach .....	6
3.2	INTERESTED Approach.....	7
3.2.1	Evaluating the use of SysML.....	7
3.2.2	Applying Timing Analysis .....	8
3.3	Metrics.....	10
3.3.1	Effort.....	11
3.3.2	Quality .....	12
3.3.3	Maturity.....	12
<b>4</b>	<b>Appendix A - automotive SPICE requirements, where the time to achieve compliance can be reduced using the INTERESTED tool chain .....</b>	<b>13</b>
4.1	ENG.5 Software Design.....	13
4.2	ENG.7 Software integration test.....	14

## 2 Motivation for the MAG Validator

### 2.1 Overview

The aim of this document is to show how Magneti Marelli will use the INTERESTED workflow, what the expected benefits are, and how we intend to measure the improvement that the INTERESTED tool chain will provide to the entire software development process.

The performance metrics of interest are the development effort, in terms of men\*months, the quality of the software development process and of the software products, in terms of the number of iterations that are needed before signoff.

Finally, the adoption of the new set of methods and tools will be motivated in terms of the coverage of best practices, which in the case of Magneti-Marelli means better satisfaction of the SPICE standard. SPICE is the standard adopted in the automotive market to measure the maturity of the software factory.

The initial terms of comparison for measuring the benefits provided by the newly adopted INTERESTED tool chain are historical data from project management and change management, as described in this report.

### 2.2 Market Requirements

Today, there are difficulties in covering the automotive SPICE process requirements for engine control SW development with the support of an automated tool chain. The SPICE process requires traceability from Requirements, via Design to Implementation.

The traditional automotive process does not fully provide this traceability, with specific weaknesses in :

- System architecture
- SW architecture
- SW testing
- System testing

This is not just a MAG issue – the whole market needs to address these traceability requirements, in particular for safety-related functions.

In the next years, multi-core microcontrollers will be widely available in the automotive market. Forecasts clearly indicate that this new technology will be mandatory for computing-intensive automotive controls. Design methods and tools and basic software support is needed for multi-core devices, with an increased complexity compared to traditional flows for single-core platforms for the following reasons:

- The system-level design must plan for the allocation of functionality and tasks onto the two cores, in accordance with the application functional and time constraints and with the availability of the platform resources.
- The communication and synchronization interactions among runnables (and tasks) can now be of different types, according to the runnable-to-task and task-to-core allocations: intra-task and inter-task. The latter can be further classified into intra-core as opposed to inter-core. The implementation details of each communication type vary in a significant way, especially with respect to the time performance and the memory requirements. Also, the debug process must be aware of the implementation of communication mechanisms for a correct interpretation of the results.
- The compilation and build process, currently performed for the one and only core, needs to be extended to support an integrated build that allows for the generation of per-core images,
- The debugging process needs to support multi-core devices, meaning that the debugger must work in a parallel scenario where the user looks at two execution units at the same time. Moreover, the analysis of the debug results is complicated by the fact that the execution flow has an increased level of parallelism, not only logical (as in a single-core preemptive operating system), but also physical.
- During the testing phase at the system-level, developers need to be able to trace back functional and timing faults to the design decisions and the original specifications. Moreover, the design models must take into account the asynchronous nature of execution of each core.

Timing requirements must be defined and analyzed for each component in the system in order to be able to trace timing faults to incorrect design decisions and faulty implementations.

In conclusion, we expect to experience problems in dealing with physical concurrency that are similar to those that happened in other industrial fields where parallel computing was introduced years ago. In addition, the time-critical nature of most automotive control applications further adds to the complexity of the design problem.

## 2.3 Timing Analysis Advantages

A timing analysis process and tools shall enable system-level Timing Design together with the system-level Functional Design. The expected advantages of using timing analysis are:

- System-level validation (no functional errors due to timing errors)
- Traceability and impact analysis (bridge between system design and product)
- Early product development support (during RFQ support the selection of the computing platform, choosing between single or multi core solutions)

It is expected, that the driver for system/software architecture modeling will be the *impact analysis* process, which is enabled through timing analysis. The SPICE standard requires evidence that the Tier 1 suppliers have the capability to perform a feasibility analysis when a car OEM asks for the addition of new functionality to an existing product. The goal is to have a good planning prediction and low rate of defect introduction. The performance predictions must be fast, in the order of a week, at least as long as only changes to the SW are required. Changes to the HW require a deeper impact analysis.

For multi-core environments, the timing analysis must take into account the different cost of inter-core communication as opposed to local communication. The optimal allocation of tasks in a multi-core platform must take into account communication overheads and worst-case blocking times in a proper model. Unfortunately, the possibility of estimating these costs based on measurements on prototype implementations is even less likely than it was before, given the combinatorial growth of the solution space with the increased number of the cores. In this scenario it is mandatory to apply analytical methods supported by appropriate tools to predict the time behaviour against the timing system constraint.

## 2.4 Cost-Benefits Estimation

The principal metrics used to verify the benefits obtained with the adoption of the INTERESTED tool chain is the number of days spent in design iterations, reworking the application. The benefit is estimated comparing the data collected during the project against the historical data collected by each Industrial validator. Other metrics are secondary benefits like quality improvement and better support for the development process.

Until now, system testing at MAG Powertrain is done by looking at raw Lauterbach traces only. This is very inefficient, and will be even less adequate in multi-core solutions, given the increased difficulty in verifying the correct (causality-driven) execution of the runnables, and their completion within the deadlines.

It is expected that a formalized software-architecture model, together with systematic timing analysis, will save **one person-year** effort per year for verifying functionality and responding to change requests. This estimate already takes into account the additional effort required for model-based design and timing analysis.

Additional benefits expected are in terms of improved software quality and maturity.

## 2.5 Process and Tools

An improved process and the corresponding tools must in particular effectively support the top-down design flow, while still providing a link to and enabling the transition from the legacy bottom-up design flow. The steps of this process steps are:

- 1) The design of the initial system-level model for the functional architecture (e.g. using SystemDesk and/or Papyrus). The authoring tools must support the refinement into a software architecture model.

- 2) The transformation of the functional architecture into a system timing model for scheduling analysis (e.g. with SymTA/S or RT-Druid). The scheduling analysis tools must support the synthesis of the software architecture configuration, such as runnable to task/core mapping, modeling inter core communication, including rule checking for execution rates, partial ordering etc.
- 3) The prediction of the schedulability of the HW/SW architecture, sensitivity analysis and optimization support. The augmentation of the scheduling analysis model with code execution times and inter core communication. These timing information can come from measurements (e.g. via Lauterbach, with further interpretation from the Syntavision TraceAnalyzer), from static code analysis (e.g. from aiT for existing and new processors), or from the AUTOSAR BSW supplier concerning the BSW execution times.
- 4) The return of the synthesized software architecture configuration to the authoring tool.
- 5) The synthesis of the software architecture model based on the function architecture and the software architecture configuration
- 6) As a condition for the above points, the central tool exchange format is AUTOSAR

Furthermore, prototype development cycles and product development cycles have to be distinguished. The full AUTOSAR methodology creates too much overhead for prototype development. Therefore, for prototype development, it should be sufficient to execute steps 1-3 rapidly, in order to estimate the feasibility of a specific software architecture configuration.

In this document, we would like also to note an issue currently existing in multi core environments: AUTOSAR currently supports multi-core devices in terms of the platform model and the allocation model, **but** commercial RTE generators still do not support the generation of communication primitives for inter-core communication (the new features are under development). This means that, in order to fulfill the project goals, MAG had to develop a multi core OS and a build process that redirects a special COM channel to multi-core communication, allowing the use of AUTOSAR system model, while building execution time support (RTE generation and OS configuration) using current commercial tools.

## 3 Technical implementation of the MAG Validator

### 3.1 Current Approach

Starting from the software requirements, software architecture is currently described in terms of:

- 1) A list of SW components. The list is organized in a three-level hierarchy: functions, sub-functions and components.
- 2) For each component, its interface is defined by listing its data-flow and send/receive ports. Each interface port is further described in terms of the min/max admitted value, the measured (physical) unit and the associated basic type (integer, float or enumerative).
- 3) For each component, the list of the events to which it reacts
- 4) For each event, an admitted partial execution order defined between each sender-receiver component pair
- 5) The list of the external signals to which the component reacts

This information is written in a document named DBF (from Distinta Base Funzionale – loosely translated into Functional Bill of Material), developed in Microsoft Excel and stored and managed in a Configuration Manager. The engineer that develops and maintains this document is a *senior control engineer* (DBF manager) that has the responsibility to ensure the consistency of the system in terms of system decomposition (number of components that constitute the entire system) and composition of its components at the interfaces (internal and external). This document is constructed and later updated in either a top-down partitioning or a bottom-up composition approach.

Starting from the DBF document, the senior software architect starts to define or updates the software architecture. For each class of Software Architectures is available a general reference document that defines layering structure, admitted communication methods between layers and configuration manager Architectural constraints, templates and naming conventions. The static design is done according to these general prescriptions using the Configuration environment. The dynamic design is done directly at code level, using the C programming language for configuring the application and its relations with the operating system. In detail, the following steps are performed:

- 1) Tasks and interrupts definition for each core if required
- 2) Runnables calling sequence definition for each task
- 3) Priority assignment and task scheduling configuration
- 4) Communication among tasks and data protection analysis
- 5) Communication definition between core if required
- 6) Appropriate firmware platform selection and configuration

In a **top-down** approach, starting from the software requirements and the DBF document the control engineers develop the algorithms for each component. Each algorithm must comply with the interface specifications as described in the DBF. If, during development, the interface requirements are found to be not feasible or not adequate, then a request for **change** of the DBF must be issued and managed.

In a **bottom-up** approach, the DBF manager starts an analysis process to understand if there is an available legacy component from the library such that its behavior and its interface meet the functional specification and the architecture interface requirements of the neighbour components. In case any suitable library component is found, a dedicated custom development is started.

#### 3.1.1 Weaknesses of the Current Approach

The analysis of the architecture feasibility (in both cases, top-down and bottom-up) is done manually. In some cases, engineers develop their own scripts (without a standardized approach) to help perform the analysis on composability.

Architecture feasibility is not supported by a predicted timing analysis at concept level: timing requirements are not part of the architecture description, timing choices are done by the experts on functional

considerations and the overall effects of the sum of these considerations are evaluable only when a SW/HW prototype is available. This implies that if the proposed architecture is not schedulable on a specific HW architecture, the timing issues are detectable only after performing code development and subsystem integration phases.

For multi-core microcontrollers, the number of the possible configuration solutions is at least double compared with single core solution. This means that the effort in the selection of the optimal design configuration can be doubled, compared with a traditional single-core design. Moreover, (performance) measurements are much more complex in a multi-core environment. Commercial AUTOSAR version 4 tools do not implement yet the generation of an RTE layer for multi-core platforms, Furthermore, the current AUTOSAR version 3 description does not support a multi core environment at all, and an extension must be defined. Finally, OSEK/AUTOSAR operating systems also do not provide adequate support for multi core predictable scheduling and communication and yet another custom implementation was required.

Moreover, the software architecture is derived from the DBF using largely a manual process, based on the interpretation of the DBF by the software engineers and by coding in C language the algorithms

## 3.2 INTERESTED Approach

The INTERESTED process flow will differ from the existing one according to the following guidelines:

- The DBF shall be designed using a model-based design tool; specifically CEA's tool Papyrus shall be evaluated for encoding the Architecture system specifications in SysML / MARTE.
- The software architecture and the component description shall be derived using AUTOSAR tools; specifically dSPACE System Desk as well as the open-source tool ARTOP shall be used to construct an AUTOSAR description.
- Using the AUTOSAR description, timing analysis shall be performed on the SW Architecture conceptual definition.

### Benefits:

- 1) Capability to check the consistency of the interfaces using automatic tools in a standard way
- 2) Capability to check if the AUTOSAR software architecture is compatible with a system-level (SysML) description
- 3) Capability to add timing information at the DBF level and to predict feasibility at design time using the integration of the architectural design tools with timing analysis tools
- 4) Capability to extend this approach to timing analysis for a dual core (multi core) solution
- 5) Capability to use AUTOSAR 3 (extended) to support dual core solution and design a configurator for a multi core operating system able to derive the OS configuration from ARXML version 3. All this extensions are aligned with AUTOSAR version 4 concept of multi core solution.
- 6) Capability to export the component interface in an AUTOSAR format and to leverage the capabilities (analysis/export/code generation) of AUTOSAR tools, including the open-source tool ARTOP.

### Penalties:

- 1) extra effort in terms of the formal development of the DBF design
- 2) understanding and leveraging the complexity of the SysML language compared to the currently available table format representation in Excel.
- 3) need to train the DBF manager to a new tool and language

### 3.2.1 Evaluating the use of SysML

One goal is to verify **IF** the SysML / MARTE language can be specialized and constrained by a profile and then used in a **simplified** way to make the architecture description more formal and suitable to automatic processing and analysis without adding an excessive effort. At the same time, we expect that the automatic verification and consistency checks that are enabled by the adoption of a SysML approach will provide a big improvement in terms of quality and fidelity of representation as compared to the current excel-based documentation.

The introduction of a more formal architectural description model also means that maintaining the integration between a SYSML/XML document and analysis environments (AUTOSAR, timing analysis, ...) is much easier than using a Microsoft Office description.

When the tool chain including Papyrus and timing analysis tools will achieve import/export integration with AUTOSAR tools by importing/exporting component descriptions in AUTOSAR format we expect to leverage integration with AUTOSAR tools in a flow that goes from system design to code generation. Of course, there must always be the possibility to directly add information in an AUTOSAR tool.

We expect the following when describing the Software architecture using the AUTOSAR standard:

#### **Benefits:**

- 1) The Software architecture is described in a standard way, a software architecture description document is derived directly from the AUTOSAR model
- 2) The Software architecture glue code is derived directly from the AUTOSAR model using an RTE generator. This process is without manual interference (no extra glue code without software architect control is admitted, there is a clear separation between the component description functional view and the software architecture view)
- 3) The timing analysis tool receives its input (the software architecture model) directly from the design tool
- 4) The consistency verification between the DBF specification and the related SW architecture (from either Papyrus or Excel) is performed with a standard tool
- 5) The comparison between two versions of the software architecture can be performed with limited effort and producing an exact and quantitative output.

#### **Penalties:**

- 1) a new artefact should be delivered and managed, extra effort must be planned
- 2) additional skills and competencies should be required from the software architect engineer
- 3) AUTOSAR 3 does not support some advanced features of MAG Powertrain SW architecture

## **3.2.2 Applying Timing Analysis**

The possibility of performing analysis in the time domain entails the following:

- 1) the capability of predicting the timing performances of a designed hw solution
- 2) the capability of verifying, given an hypothesis of a HW platform, the satisfaction of the timing constraints at design time;
- 3) the capability of performing schedulability analysis at the task level also in multi core solution scenario.

### **3.2.2.1 Code execution time analysis**

#### ***BEFORE***

Today, component profiling needs measurement of the execution times and is performed using a concrete platform implementation on an MAG target ECU or an Evaluation Board (EVB), depending on the stage of development. Typically, at early stages or during HW evaluation, the use of an EVB is necessary.

In detail, when the software is available, the code is cross-compiled for the desired HW target and the measurements are performed directly on the target. Measurements are of course affected by the possible limited coverage of the stimuli. The exploration of different settings of the HW platform is done directly on the HW.

For model-based code generation (MAG uses Target Link tools from dSPACE) TL supports different EVB and an integration of the software design tool with the EVB is available. After integration, it is possible to profile time performances and stack consumption for each specific input set.

For manual code generation, the flow is different. The software engineer integrates the component in a prototype. Availability of a complete prototype is necessary because the measurements are done on the MAG ECU and the simplest way to have something that works on the ECU is to perform the measurements on an integrated SW that contain the SW-component to be measured. At the end, the Lauterbach tools are used to profile the time performances and the stack consumption in the ECU by the component. The input stimuli are realized through a static simulator or other instruments connected to the ECU. Again, the confidence of the measure is directly related to the input coverage.

### **AFTER**

The software project leader needs to evaluate the timing performances of the SW components in these cases:

- 1) At early stages to evaluate a new HW platform.
- 2) At advanced stages to define an optimization strategy (discover the components from which the system would benefit more if optimized).
- 3) During software development to help the software designer to keep the software optimized as much as possible to the timing target. In these cases, when the goal is not met, a change of the control algorithm is started or a redesign of the software implementation is issued.

Adopting the INTERESTED tool chain we expect the following benefits:

- 1) AbsInt's tool aiT should be used by the software engineer at each compilation cycle without the need of an actual target (this is true for both manual- and model-based code generation). In brief, the main advantage in using aiT by itself for MAG is the ability to explore HW configurations quickly without the need to purchase several HW platforms and learn to use their corresponding development environments. In addition, the time/stack estimates are not affected by availability of input vectors but the tool performs analysis for each possible path exhaustively. The analysis can be made more precise by identifying redundant paths or focusing on the paths that should be optimized in the software or algorithm.
- 2) The integration between AUTOSAR tools and AbsInt's aiT will reduce the time necessary to setup the configuration of aiT providing a quicker way to produce estimations.
- 3) The integration between AbsInt's aiT and the scheduling analysis tools RT-DRUID / SymTA/S will allow better support and more accuracy for scheduling analysis by avoiding the back annotation of the time/stack consumption into the models fed to the scheduling analyzer

The only concern in using aiT is related to the accuracy of the measurements and the current limitation of some HW configurations that can limit the SW exploration.

### **3.2.2.2 Scheduling Analysis**

Scheduling analysis can be performed in different situations and with different purposes:

- during software freezing before the product is delivered to the customer
- during development when an overrun error occurs
- before starting a new development to estimate the amount of available time for execution before the task misses its deadline and/or a timing fault occurs.

### **BEFORE**

Today, the analysis is performed starting from a measurement of each task. The operational conditions of the ECU during task execution measures are in a car used for test trips or at HIL (Hardware in the Loop under predefined checklist). A special hardware counter is defined and associated to each task and min/max and average execution times are recorded during the trip. The procedure may be automated by using an ORTI description in conjunction with the Lauterbach tools. After the collection of this data an estimation of the worst-case execution time of each task is provided and information is forwarded to the RT-DRUID tool for schedulability and sensitivity analysis. The report describing the time attributes and the results of the analysis

is used as a documentation of the system integration. In addition, the timing results are used by the software project leader to plan the next actions before the introduction of new functionality in the system (to understand the amount of time available for execution to each task).

For multi core environments the analysis of the basic timing performance figures is performed using a measurement from a prototype using the Lauterbach debugger and the system-level scheduling analysis is performed using an analytical formula without any tool support. RTD at the beginning of the project was not able to model multi core and intra-core communications.

## **AFTER**

### **Top down improved flow:**

Using the integration between AUTOSAR and UML/SysML/MARTE as supported by Papyrus, we can explore different options for mapping runnables into tasks and cores. The analysis results will guide the mapping iterations and will be provided by RT-Druid and/or SymTA/S. RT-Druid and/or SymTA/S will receive the input data from aiT or (in case aiT is not available) from the results of measurement done on the target at task/ component level, and will estimate load and worst-case response times (in case of SymTA/S, also distribution of typical-case response times). At this stage, the aim is to explore different mappings of the runnables and use the analysis results to drive the optimization process.

Given an optimal mapping, RT-Druid and/or SymTA/S generates a configuration of the operating system. The configuration starts from an AUTOSAR description extended to support a multi-core platform. MAG chose to focus on AUTOSAR 3, because the commercial RTE generators that start from a version 4 description do not provide (up to now) support for the generation of inter-core communication in multi core platforms. The scheduling analysis performed also takes into account the intra core communication methods and the corresponding blocking times (evaluated with a formula completely different from the single-core case) and overheads.

Moreover, the scheduling analysis tool must be able to validate that the scheduling order is compatible with the partial order constraints, and must be able to back annotate the schedule configuration to the AUTOSAR specification.

### **Bottom up improved flow**

In this case the improvements are mostly related to better support the software integration testing stage.

When the software integration is completed and signed off, we will improve the integration testing procedures by adding time-related verification actions:

- Generate an aiT configuration file using RT-Druid/SymTA/S. RT-Druid/SymTA/S will start its analysis from the AUTOSAR description received as input.
- Calculate the WCET of each task/component by aiT
- Take the calculated WCET as an input to the scheduling analysis tools (RT-Druid/SymTA/S). The results of the analysis (load, WCRT, distribution) are immediately available without any HW support.

## **3.3 Metrics**

MAG, in order to estimate cost/benefit in adopting the INTERESTED tool chain has used as the **main metrics** the number of days spent in design iterations / reworking the application, and as a **secondary metrics** the improvement of the quality in project management (reduction of iterations) and how the INTERESTED tool chain helps in performing best practice required by AUTOMOTIVE SPICE in a more effective way.

In the final phase of INTERESTED, the work was extended to multi-core processors. For multi-core solution it is not possible to provide current efforts, as there is no existing data. We can however forecast that for dual core WITHOUT INTERESTED the effort will double versus the current single core developments. Therefore, we multiply our existing single-core reference numbers by the factor 2, to arrive at dual-core reference numbers. Here, the main advantage of the INTERESTED tool chain is that it is independent from the number of the cores in terms of design. The impact is only a small extra effort spent in analysing the results of the scheduling analysis due to the larger number of solutions that the tools have considered. As such comparisons can be automated (e.g. using the scripting capabilities of SymTA/S) we expect, that the benefit of INTERESTED will essentially double for dual-core CPUs.

### 3.3.1 Effort

The evaluation of the possible savings and improvements on the software development process requires the extraction of measured data on development costs/efforts. These costs have been registered in the change management system in use at MAG. For the analysis, we considered registrations from all projects on diesel engines developed by the diesel engine groups.

Three classes of changes are recorded in the system:

- *Enhancements* or functional changes due to new requests, related to the addition of new functionalities, or improvements of existing functionalities
- *Software defect fixes*, due to errors found in the software as consequence of a wrong implementation of a requirement.
- *Specification defect fixes*, related to the situation of a correct software implementation, but a specification that is not able to satisfy completely the input requirement

The following table lists the number of each of these changes classes for the last three years.

	Enhancement changes	SW defect fixes	Specification defect fixes
2007	113	268	29
2008	136	99	45
2009	219	111	116

Our analysis is focused on 2009. A more accurate analysis of the enhancement changes reveals that in 70% of the cases these changes were related to component integration issues. The motivation of this kind of changes could be a not complete understanding of the requirement or an incorrect architecture-level design/decision, but in any case its implementation introduced a modification of the previous architecture design.

Matching these data with the working effort days available for each diesel project from the project Management effort monitoring environment, we have found the following: the total number of work days (including rework) in 2009 for the diesel engine team was 2750 days. Out of these 2750 days, 800 have been recorded as days spent because of changes (of any one of the three previously listed classes). These days are labelled as spent on rework.

#### ESTIMATES

From the previous data, we estimate that 70% of rework has a relationship with changes in the architecture design or issues with architecture design, for a total effort spent in fixing architecture-related issues of  $800 \cdot 0.7 = 560$  days.

We estimate that 50% (and possibly more) of these days could be saved if we apply the INTERESTED tool chain, because of better design, analysis and automatic verification of consistency. This amounts to a total saving of  $560 \cdot 0.5 = 280$  days a year.

On the cost side, the application of INTERESTED will probably result in additional work for the build manager. This additional effort can be estimated by considering the number of software releases (builds) issued in one year.

We have one build per week every year, which amounts to roughly 50 software builds each year. We estimate that 50% of these releases are because of rework. One new instance of a DBF is needed for each release. Although most of these developments are incremental, an estimated effort of 2 days of work is required for creating the DBF in its current form for each build. In the future, we expect the DBF to be replaced by a set of SysML/AUTOSAR diagrams. These diagrams will be possible more complex to define. Even if we expect the greatest part of the additional work to be limited to the creation of the initial design (not to incremental changes), an average increase of 1 day of work for each build is expected (to be charged to the INTERESTED tool chain).

Considering 50 build every year, this amounts to roughly a total of 50 days. The forecasted improvement is therefore  $280 - 50$  days = 230 days every year. In percentage, the savings refer to 2750 total days of work per year. Therefore, the expected saved effort is  $230/2750 = 8.3\%$ . If the additional work on the SysML/AUTOSAR diagrams were 2 days/build instead of 1, then the savings would be  $180/2750 = 6.5\%$ .

However, if we expect the quality of the process and the accuracy of the estimates to increase, then we should also expect a reduction in the total number of release/builds, at least for the share (70%) that is caused by architecture defects (35 out of 50). If the number of these builds for rework drops by 50% (from 35 to 18), then the additional effort for the build manager to define the SysML/AUTOSAR diagrams is only 32 days. Therefore, the savings could be  $248/2750 = 9\%$ .

As initially stated, these numbers apply to single-core processors. If we move to dual-core we expect a significant increase in effort due to a larger number of changes and fixes. As the effort to use the INTERESTED tool chain will increase only slightly, the overall savings will be higher, better than 10%.

### 3.3.2 Quality

In qualitative terms the halving of the work days because of rework implies a better quality of the SW releases, which means an increased customer satisfaction.

We also expect to reduce the release of new builds for rework for an estimated 50% from 50 to 33 releases per year; this is because of the decrease in the number of defects related to the architecture.

This implies a general increase of the global SW development cycle efficiency that up to now is not evaluable in a quantitative form: we expect the possibility of better evaluating this item after 2 years of application of the INTERESTED tool chain.

### 3.3.3 Maturity

Today, there are difficulties in covering the automotive SPICE process requirements for engine control SW development with the support of an automated tool chain. The process today is largely manual and thus very time-consuming. Among SPICE requirements, mainly ENG5 and ENG7 could benefit from the INTERESTED tool chain. It is expected, that the time required to achieve compliance with these requirements can be reduced significantly with the INTERESTED tool-chain. In Appendix A, we provide the original definition of each best practice from version 2.2 of the automotive SPICE standard. The BP, where the time to achieve compliance will be reduced by INTERESTED are marked in blue, the BP out of scope in grey.

## 4 Appendix A - automotive SPICE requirements, where the time to achieve compliance can be reduced using the INTERESTED tool chain

### 4.1 ENG.5 Software Design

The purpose of the Software design process is to provide a software design that implements and can be verified against the software requirements.

**ENG.5.BP1: Develop software architectural design.** Use the functional and non-functional software requirements to develop a software architecture that describes the top-level structure and all the software components including software components available for reuse.

*NOTE 1: See also REU.2 – Reuse Program Management.*

The adoption of Papyrus will help better fulfil this BP: a formal language is adopted to describe the top-level structure of the software architecture that is today described in Excel inside our information repository. Papyrus is a design framework. Using the export functionality versus AUTOSAR, we can provide an interface to an AUTOSAR flow. Using the import functionality, we can enrich the top level architecture with timing constraints and time attributes coming from a software architecture already implemented. Of course, it is also possible add constraint directly in Papyrus

**ENG.5.BP2: Allocate software requirements.** Allocate all software requirements to the components of the software architectural design.

**ENG.5.BP3: Define interfaces.** Identify, develop and document the internal interfaces between the software components and external interfaces of the software components.

Both UML/SysML and AUTOSAR software descriptions fully describe the internal and external interface of the system and its components.

**ENG.5.BP4: Describe dynamic behaviour.** Evaluate and document the dynamic behaviour of and interaction between software components.

The adoption of timing analysis tools like RT-DRUID and SymTA/S is possible to define the timing behaviour and verify if this behaviour is achievable.

**ENG.5.BP5: Define resource consumption objectives.** Determine and document the resource consumption objectives for all software components.

*NOTE 3: Resource consumption is typically determined for resources like Memory (ROM, RAM, external / internal EEPROM), CPU load, etc.*

Today this BP is not completely fulfilled due to difficulty in collecting the data for each component. Historical data refer, when available, to tasks and not components. Using the integration between AUTOSAR and AiT we should be able to systematically collect data for each component / task in an automatic way. When the system is built and integrated we will perform measurements and evaluate for each task its schedulability and the time available for additional execution of functions before it misses a deadline.

**ENG.5.BP6: Develop detailed design.** Decompose the software architectural design into a detailed design for each software component describing all software units and their interfaces.

*NOTE 4: Task execution time is highly depended on target and loads on the target which should be considered and documented*

Today, consistency among component interfaces and the interface defined at architectural level is evaluated by comparing the Simulink interface with the DBF interface as defined in the Excel file. When AUTOSAR is adopted to describe the SW architecture, the coherence between Simulink and AUTOSAR will be evaluated in a standard way and using a manageable standard format.

The use of Timing Analysis Tools should contribute to a correct evaluation of the task execution time on the specific target adopted.

**ENG.5.BP7: Develop Test Criteria.** Define the test criteria for each component concerning their dynamic behaviour, interfaces and resource consumption based on the software architectural design.

The test criteria are specified in the Papyrus framework and in the AUTOSAR description. The runnable execution order should be compliant with the Papyrus order and the SW integrated should be coherent with the AUTOSAR description.

**ENG.5.BP8: Verify Software Design.** Ensure that the software design meets all software requirements.

**ENG.5.BP9: Ensure consistency and bilateral traceability to software requirements.**

Ensure consistency of software requirements including verification criteria to software architectural design including verification criteria. Consistency is supported by establishing and maintaining bilateral traceability between the software requirements including verification criteria and software architectural design including verification criteria.

**ENG.5.BP10: Ensure consistency and bilateral traceability to software architectural design.**

Ensure consistency of software architectural design including verification criteria to detailed design including verification criteria. Consistency is supported by establishing and maintaining bilateral traceability between the software architectural design including verification criteria and detailed including verification criteria.

## 4.2 ENG.7 Software integration test

The purpose of the Software integration test process is to integrate the software units into larger assemblies, producing integrated software consistent with the software design and to test the interaction between the software items.

**ENG.7.BP1: Develop software integration strategy.** Develop the strategy for integrating software items consistent with the release strategy and an order for integrating them.

The INTERESTED tool chain integration suggest an integration strategy based on the capability to derive the software architecture from a top level Papyrus description and through an integration scheme based on timing analysis results.

**ENG.7.BP2: Develop software integration test strategy.** Develop the strategy for testing the integrated software items. Identify test steps according to the order of integration defined in the integration strategy.

*NOTE 1: The integration test will focus mainly on interfaces, data flow, functionality of the items etc.*

*NOTE 2: The Software integration test process should start with the beginning of the software development process There is a close link from Software Requirements Analysis ENG.4, Software Design ENG.5 or Requirements Elicitation ENG.1 in developing test cases and testable requirements.*

*NOTE 3: The software integration strategy contains different approaches of integrating software items depending of the changes (e.g. new units, changed units). The integration strategy also includes the most suitable test methods to be used for each integration approach. NOTE 4: The identified items and the order of integration will have an influence on integration test strategy.*

The integration test strategy suggested by INTERESTED is:

- 1) verify the consistency between AUTOSAR total order and papyrus partial order
- 2) verify, running SW on the target, the compatibility with the total order and priority defined in AUTOSAR
- 3) verify the stack and timing constraint using WCET (a trip measure is suggested periodically to verify WCET prediction)

**ENG.7.BP3: Develop test specification for software integration test.**

Develop the test specification for software integration test including test cases, to be executed on each integrated software item. The test cases should demonstrate compliance to the software architectural design and software detailed design allocated to each software item.

**ENG.7.BP4: Integrate software units and software items.**

Integrate the software units to software items and software items to integrated software according to the software integration strategy.

*NOTE 5: Software units are integrated to software components and software components to integrated software.*

*NOTE 6: The integration of the software units and software components also integrates their data. Data can be calibration data and variant coding data.*

Using the RTE AUTOSAR generation these steps are performed using a CAE tool like System Desk from dSPACE. Also, the operating system is configured through the AUTOSAR description

**ENG.7.BP5: Verify the integrated software.**

Verify each integrated software item against the test cases for software integration test according to the software integration test strategy.

*NOTE 7: Verification of the integrated software produces the test logs.*

The timing verification is performed on virtual platform like ABSINT or using Lauterbach integrated with RT-Druid/SymTA/s in order to ensure that the total order retrieved from the trip is compatible with the specification even if a overrun occurs or some faulty condition appears.

**ENG.7.BP6: Document the software integration test results.**

*Document the results of the software integration test. The results should be delivered and communicated to all relevant parties in an appropriate way.*

*NOTE 8: The test incident reports and the test summary report are based on the test logs.*

Using the tool chain, an integrated documentation of the timing verifications should be produced automatically after each step.

*ENG.7.BP7: Ensure consistency and bilateral traceability. Ensure consistency of software architectural design and software detailed design to test specification software integration test, including test cases. Consistency is supported by establishing and maintaining bilateral traceability between software architectural design and detailed design including and test specification software integration test including test cases.*

**ENG.7.BP8: Develop regression testing strategy and perform regression testing.**

*Develop the strategy for re-testing the software items if changed software items are integrated. Perform regression testing as defined in the regression test strategy and document the results.*