Consistent Cost Estimation for the Automotive Safety Model based Software Development Life cycle

Demetrio Cortese
FPT Embedded Software Development
CNH Industrial
Turin, Italy
Email: Demetrio.Cortese@cnhind.com

Abstract—The Safety Model based Software Development Life-cycle, focused on high-level executable models of the automotive systems to be fielded, has a high maturity level. It allows compression of the development cycles through a wide range of exploration and analysis including high fidelity of simulation, automatic test case generation and even test session at low cost early in the development phase. Many Software Development Teams in the automotive industry are already using model-based development for their safety critical software. The current software development effort estimation through sophisticated models and methods (COCOMO COnstructive COst MOdel, COCCOMOII, functional point, etc.), obtained at the early stages of development life cycle, is often inaccurate because of the long duration between the initialing phase of the project and delivery phase. Also, not many details of the functions are available at that time. All these models require as inputs accurate estimates of specific attributes, such as line of code (LOC), number of complex interfaces, etc. that are difficult to predict during the initial stage of software development. Effective software project estimation is one of the most challenging and important activities in software development. Proper project planning and control is not possible without a sound and reliable estimate. The basis of our approach for estimation of the development cost of a new model based development project is to describe it in terms of complexity and then to use this description to find other similar model–based functions that have already been completed.

Keywords-Model Based; Executable specification; Cost estimation; Embedded Software; Autocode generation; Software Engineering; Functional Safety.

I. INTRODUCTION

The assessment of the main risks in software development discloses that a major threat of delays is caused by poor effort/cost estimation of a project. As a consequence, more projects will face budget and/or schedule overruns. This risk can affect all phases of the software development life cycle, i.e., Analysis, Design, Coding and Testing. Hence, mitigating this risk may reduce the overall risk impact of the project in a consistent way.

Existing estimation techniques, such as function point and use case estimation, are done after the analyses phase and the cost/effort is measured in terms of lines of codes for each functionality to be incorporated into the software. Therefore, it is very clear that only a specific part of the total software development effort is estimated and this estimation is delayed until after all the analyses and designs are completed. Current software cost estimation methods first try to determine the size of the software to be built. Based upon this size, the expected effort is estimated and it is utilized to calculate the duration (i.e., time required) and cost (monetary/human resources) of the project.

We have adapted a different approach and suggested that effort estimation shall be carried out for each phase of the development process. Applying a phase-based approach offers a project manager the possibility to estimate the cost at different moments in the life cycle. A milestone offers the possibility to assess each phase and to measure and analyze possible differences between the actual and the estimated, step by step. Each milestone should, therefore, be considered the time for deciding whether the estimation can be adjusted. This mechanism leads to continuous and dynamic assessment of the relation between activities and relevant costs estimation. The philosophy, therefore, is to identify the project functionalities and define the software development process in all phases. It is clear that the end of a phase is characterized by a milestone.

Our approach is a quick and consistent method, based on:

1. Reference Model based Software Life cycle in all phases, as described in Section II,
2. Definition of an implementation scale of the existing model based functionalities (five levels), as described in Section III, subsection A;
3. Definition of a complexity scale of the existing model based functionalities (five levels), as described in Section III, subsection B;
4. Complexity Functionality by asking System Experts for estimate on each functionality of the new project, as described in Section III, subsection B;
5. Calculation of uncertainty, as reported in Section IV;
6. Affinity process through comparison and tuning for the new functionalities having as reference the historical data from point 1 and 2, as given in Section V;
7. Corrector factors (Team Skill, process customization) to be adjusted according to their risk, considered in Section V;

In this paper, the application of the above effort estimation model for a Software development project for a new Engine ECU (Electronic Control Unit) will be also presented.

II. MODEL BASED APPROACH APPLICATION

In the automotive domain, the model-based approach is a consistent way to master the management and complexity in the developing software systems. In last 10 years, the CNH industrial Embedded Software Development Team targets the Model-Based Application Software Development of the Functional Safety Systems for all CNH Industrial application by implementing a Methodology that ensures the safety critically relevant process satisfies important OEM (Original Equipment Manufacturer) requirements [8][9]:

- High quality
- Reduction in time to delivery
- Reduction in development cost

This strategy, through the CNH Industrial Infrastructure framework, allows innovation in existing processes and yield benefits in the medium term:

- A reduction of the Design Life-Cycle Process
- Anticipating issues at early design phases of development, leading to reduction in systems project risks
- Increasing effectiveness and timeliness of the system verification life-cycle, with reduction of systems time-to-delivery.

ISO (International Organization for Standardization standards) [4][5][6], such as ISO 26262 (Truck & Bus), ISO 13849 (Construction equipment) and ISO 25119 (Agricultural) do not specify formally any development process or validation tools but provide only recommendations. A clear description of the CNH Industrial process tailoring has been done in [1].

Here, we repeat the basic concept stating that, while the requirements of the Functional Safety standard cannot be tailored, the activities performed to meet the Standard can and should be tailored. That is, while the requirements must be met, the implementation and approach to meeting these requirements may and should vary to reflect the system to which they are applied. It is each software development's responsibility to produce evidence that they follow development processes addressing the safety-relevant requirements, and traceability from requirements to implementation. Additionally, traceability from requirements to test cases that checks the correctness of requirements against the developed software, is required. Besides, it is always important to verify that their development tools do not introduce errors in the final software product. In general, there two kinds of safety requirements: process oriented and technical. Both need to be addressed and properly documented within a project of software development. In the following, we identify process oriented requirements (what needs to be done to ensure software safety). Technical requirements are those that specify what the SW function must include. In order to manage the safety requirements, the software development process should:

- Identify, manage and monitor the safety requirements of the software product life-cycle, including generation of requirements, design, coding, test and operation of the software.
- Ensure that software acquisitions, whether off the-shelf or outsourcing, have been evaluated and assessed.
- Ensure that software verification activities include software safety verifications
- After the Final Software delivery, ensure that all changes and reconfigurations of the software are analyzed for their impacts to system safety.

In the last years, a consistent Model-Based Application software development Life–cycle (as shown in Figure 1,), compliant with ISO 26262 "Functional Safety", has been identified [2] in CNH Industrial.

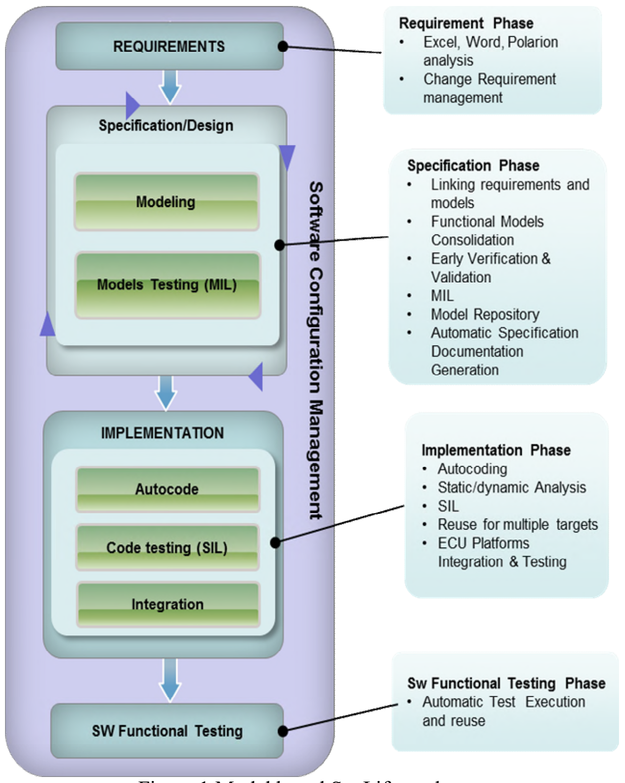
REQUIREMENTS
Specification/Design
Modeling
Models Testing (MIL)
IMPLEMENTATION
Autocode
Code testing (SIL)
Integration
SW Functional Testing
Software Configuration Management
Requirement Phase
Excel, Word, Polarion analysis
Change Requirement management
Specification Phase
Linking requirements and models
Functional Models Consolidation
Early Verification & Validation
MIL
Model Repository
Automatic Specification Documentation Generation
Implementation Phase
Autocoding
Static/dynamic Analysis
SIL
Reuse for multiple targets
ECU Platforms Integration & Testing
Sw Functional Testing Phase
Automatic Test Execution and reuse

Figure 1 Model based Sw Life-cycle

Our approach responds to the demand of a collaborative environment that increases productivity and drastically cuts the development time. It will be obtained by capturing and disseminating the expertise of different and distributed teams. This comprehensive environment helps the Engineers in all life cycle stages from high-level data and architecture models through to fully tested and running Software Modules, harmonizing life cycle phases for the OEM application. One of the most powerful aspects of our approach is that it establishes a common language designed to engage all stakeholders in a process that leads to optimal applications outcomes, rather than outcomes that are locally optimized to the needs of any particular area.

Figure 1 describes our Model based Software development life cycle, where phases and tasks have been identified.

III. Project Complexity definition

Before we begin a piece of software development estimation, there needs to be an understanding of the scope of the project in terms of process and functionalities of the project. In case of the Model based development with a high maturity level, it can be extremely challenging to estimate the project effort through two pillars: complexity of the development process and complexity of the functionality content.

A. Software development process complexity

Today, engine functionalities often are large and complex, and the usual approach in our department consists of building the large functionality from smaller Software components. A Software component is a unit of complexity that required a reasonable effort, in general, and far less difficult than the whole functionality. We identified five different SW components: very simply, simply, medium, complex and very complex. The above classification is based on the timing specified by the Model based Software development process, as consolidated in the Embedded Software development department in FPT (FIAT PowerTrain) Industrial.

Successfully integrating components result in the whole SW functionality. Integrating the components into a larger software system consists of putting them together in a sufficiently careful manner, such that the components fit together. The use of models consisting of different submodels within software development has numerous advantages. Different persons may work on different submodels simultaneously driving forward the development of the system. Different types of submodels allow the separation of different aspects of the system, such as structural aspects or dynamic behavior of system components. On this basis, we are viewing each component in terms of complexity of software development life cycle phases. Each phase ends with a milestone and defined outcomes. Using a model based approach, as defined in previous section, we can estimate the effort needed to perform each phase. Our approach offers the SW manager the possibility to estimate different component/models at different moments in the life cycle for different complexities of the models (very simple, simple, medium, complex and very complex). Next, the milestones and the outcomes for each phase offer the possibility to measure and analyze possible differences between the actual and estimate step by step, such that the objectives, the estimate or the planning can be adjusted. By defining its own SW lifecycles, an organization can collect and use its own local historical data obtained from completed MBD (Model Based Development) projects. For example, we are using the Model based SW development process since 2005 and, therefore, we have consistent data to consolidate the estimation of effort, as shown in TABLE I.

This table represents the effort for each defined phase of the MBD Software development process for different complexity of the models. It uses 5 functional complexity indicators to show the development effort in 6 consolidated phases (with relevant tasks) of the CNH Industrial Model based Software development life-cycle. For Intellectual Property Rights (IPR) reason, the table is empty.

This mechanism leads to continuous and dynamic assessment of the relationship between process phase and costs. In case of automatism, we will be able to introduce it during the SW development process, as for example, more automatic HIL test procedures.

TABLE I. SOFTWARE DEVELOPMENT COMPLEXITY

Embedded Software Development (ESD)								
Work Estimation Details			***Very Simple***	***Simple***	***Medium***	***Complex***	***Very Complex***	***Remarks***
SW Development Phases	***Activity/Task***	***Owner***	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	
Requirements								
	Functionality Requirements Management	SW Project Leader						through Polarion customization
	Functionality Interfaces Requirement Specification	SW Project Leader						Interfaces definition
	Functionality Requirements Specification	System Engineer						traditional paper based specification
	Functionality Requirements Traceability	MBD Engineer						Requirements, Model, Code, Documentation, testing)
Specification								
	Requirement Specification Analysis	MBD Engineer						through traditional review of documents
	Functionality Modelling	MBD Engineer						through Simulink customization
	Modelling checking	MBD Engineer						through Model Advisor customization

Embedded Software Development (ESD)								
Work Estimation Details			***Very Simple***	***Simple***	***Medium***	***Complex***	***Very Complex***	***Remarks***
SW Development Phases	***Activity/Task***	***Owner***	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	
	MIL Testing (including the report generation)	MBD Engineer						including the MIL (Model In the loop) Test report; through Internal and automized tools
	Sw Functionality Documentation	MBD Engineer						through internal and automized tools
Implementation								
	AutoCoding	MBD Engineer						through Embedder coder Customization
	Code Review	MBD Engineer						through polyspace customization
	Code Integration in the ECU platform (with successful Compilation)	MBD Engineer						including configuration files, compilation
	Unit Testing	MBD Engineer						including the unit test report
Functional Testing								
	Test Environment Setup	System Engineer						HIL (Hardware In the Loop) Automatic setup including automation in test preparation, execution and reporting
	Integration Testing	System Engineer						
	Performance Testing	System Engineer						
Delivery								
	Software Configuration Management	SW Project Leader and MBD Engineer						Through configuration management tool
	Release/Build Updates	SW Project Leader						based on the Basic Software platform provided by the supplier
Support								
	Post delivery Support (eventual incremental version)	MBD Engineer						we plan a sw bug to consolidate the functionality
Work Estimate Totals			**y**	**1,8y**	**3.6y**	**4,8y**	**6,4y**	

B. *Functionality Complexity*

An engine ECU Software is a collection of software functionalities, describing features that can then be broken down into smaller and smaller components. Our idea is to assign a complexity rating to all functional components. The estimates, provided by an expert who has a background in the requirements definition, can be modified to suit the experience/expertise and performance of the team or people who might actually perform the work. This technique captures the experience and the knowledge of the experts. During the lifecycle, re-estimates should be done at major milestones of the project, or at specific time intervals. This decision will depend on the situation. SW Changes may be made during the project and therefore the cost estimates either increase or decrease. At the end of the project, a final assessment of the results of the entire cost estimation process should be done. This allows a company to refine the estimation process in the future because of the data results that were obtained, and also allows the developers to review the development process. It is also true that there are only very few cases where the software requirements stay fixed. Hence, how do we deal with software requirement changes, ambiguities or inconsistencies? During the estimation process, an experienced expert will detect the ambiguities and inconsistency in the requirements. As part of the estimation process, the expert will try to solve all these ambiguities by modifying the requirements. If the ambiguities or inconsistent requirements stay unsolved, then it will correspondingly affect the estimation accuracy.

The approach is flexible and allows us to account for the effort for all components. Once we have defined the functionality breakdown and set complexity estimates, we will be able to have the relevant effort estimation based on the concept introduced in the previous paragraph (Software development process complexity). As described in TABLE II, this is obviously a very simple spread sheet, and the calculations made are not in any way close to being hyper-accurate. It provides a handy mechanism to document and trace effort against functionalities, and a framework for distributing effort to project tasks (like requirement, implementation, testing etc.) across the total effort.

TABLE II. SW FUNCTIONALITY COMPLEXITY

Sw Layer	**First level**	**Second level**	**Name**	**Tasks Number**	**Functional Complexity Breakdown**					**Development Time Estimation (Hours)**					**Total Effort Estimation**
					1	*2*	*3*	*4*	*5*	*1*	*2*	*3*	*4*	*5*	*Hours*
ASW	Engine Function	Air System	AirMod	6		3	5			0	3x1,8y	5x 3,6y		0	
ASW	Vehicle Function	Active Surger Dumper	ASDCtl	2		1	1			0	1,8y	3,6y	0	0	
ASW	Engine Function	Air System	BstCtl	3		1	2			0	1,8y	2x 3,6y	0	0	
ASW	Engine Function	Air System	ChrCtl	5		3	2			0	3x1,8y	2x 3,6y	0	0	
ASW	Engine Function	Air System	ChrSet	4			1	1	2	0		3,6y	4,8y	2x6,4y	
ASW	Engine Function	Coordinator Engine	CoEng	5		2	3			0	2x1,8y	3x 3,6y	0	0	
ASW	Communicat ion	Vehicle	ComVe h	87	25	50	12			25x	50x1,8y	12x 3,6y	0	0	
ASW	Vehicle Function	Cruise Control	DrAs	9		4	5			0	4x1,8y	5x 3,6y	0	0	
….	……	………..	……												
……	…………..	……………… ……	………..	.	.	.	.	.							

TABLE II illustrates three important indicators for each functionality of the Application Software:

- Functional Complexity Breakdown: For each software functionality, we followed the same approach. We interviewed an adequate number of experts (Engine and Vehicle System Engineers). We defined each functionality as the composition of sub functionalities (tasks) with different complexity, starting from the experts experience with the most recent one and going back as far as they could.
- Development Time estimation: For each software functionality, the Software Manager will identify the estimation effort based on TABLE I.
- Total effort estimation: for each functionality, we show the effort in terms of Hours, days and months.

where

1= Very Simple, 2= Simple, 3=Medium, 4=Complex and 5=Very complex

IV. PROJECT DEFINITION ACCURACY

Accurate project estimation is one of the most challenging aspects of a project. The estimation becomes increasingly difficult as the project's complexity and uncertainty increases. Effort estimation accuracy depends on the available information. Usually, there is less information at the start the project (presales) and more information while working on the project, for example, after the requirement consolidation. In order to increase the accuracy level, the PERT (Program Evaluation and Review Technique) three-point estimates is used. It provides a range of project estimates and calculates the weighted average of that range. In order to use the PERT project estimation technique, we provide 3 data points, the "**best case**", "**most likely case**" and the "**worst case**".

The optimistic scenario (best case) is usually the shortest duration and/or the least costly estimate based on the notion that all will go well on the project. The pessimistic scenario (worst case) is the longest duration and/or the most costly estimate, based on the notion that problems may be encountered during the project. The most likely scenario falls somewhere in between the pessimistic and optimistic estimates, based on the notion that the project will progress under normal conditions.

In general, the experts will be asked to first provide their worst case estimate and then the best case estimate. Once these 2 points are agreed upon, it is easier for them to determine the most likely case, knowing their upper and lower limits.

Based on the above data, we obtain the PERT estimate:

$$E = (o + 4m + p) / 6. \qquad (1)$$

where E is Estimate; o = optimistic estimate; p = pessimistic estimate; m = most likely estimate.

Standard Deviation:

$$SD = (p - o) / 6 \qquad (2)$$

where SD is Standard Deviation; p = pessimistic estimate; o = optimistic estimate

E and SD values are then used to convert the project estimates to confidence levels as follows:

1. Confidence level in E value is approximately 75%
2. Confidence level in E value +/- SD is approximately 85%
3. Confidence level in E value +/- 2 × SD is approximately 95%
4. Confidence level in E value +/- 3 × SD is approximately 99.5%

TABLE III describes the approach.

TABLE III. SOFTWARE COST ACCURACY

Sw Layer	**First level**	**Second level**	**Name**	**Best case**	**Most Likely case**	**Worst Case**	**Standard Deviation**	**0.75 Confidence**	**0.85 Confidence**	**0.95 Confidence**	**0.995 Confidence**
				Effort Hrs	***Effort Hrs***	***Effort Hrs***		***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***
ASW	Engine Function	Air System	AirMod								
ASW	Vehicle Function	Active Surger Dumper	ASDCtl								
ASW	Engine Function	Air System	BstCtl								
ASW	Engine Function	Air System	ChrCtl								
ASW	Engine Function	Air System	ChrSet								
ASW	Engine Function	Coordinator Engine	CoEng								
ASW	Communication	Vehicle	ComVeh								
ASW	Vehicle Function	Cruise Control/Speed Limiter	DrAs								
….	………… …	………… ………..	…….	………	…						
…...											

This technique works great for several reasons:

- Psychologically, it is easier to provide a number when you can provide a wide range
- Starting with the worst case often leads to less resistance
- Once worst case and best case are identified, it becomes easier to provide the most likely case
- Reduces the natural instinct to inflate estimates

V. SOFTWARE DEVELOPMENT PROCESS COST ESTIMATION

As introduced in the previous sections, the cost of development activities is primarily the development effort costs. This is the most difficult to estimate and control, and has the most significant effect on the overall project cost. Software cost estimation is a continuing activity which starts at the proposal stage and continues throughout the lifetime of a project. Projects normally have a budget, and continual cost estimation is necessary to ensure that spending is in line with the budget. Therefore, it is very important to estimate the software development effort as accurately as possible. A basic cost equation can be defined as:

$$\text{Total\_SW\_Project} = \text{SW\_Development\_Labor} + \text{Other\_Labor}$$

In fact, we may have to consider other labor costs, such as:

- Software project management, performed by the project Manager, to plan and direct the software project
- Facility Administration, for example software configuration management and tools maintenance. The software development facility (SDF) is composed, generally, of hardware, software, and services utilized by the MBD Engineering Team to generate and test code, perform the engineering analysis, generate all of the required documents and manage the software development.
- SW Process Enhancement & Innovation.

The Innovation activity is a great way to improve the SW development process and the quality of the software product. Enhancement actions of software development helps organizations to establish a mature and disciplined engineering practice that produces secure, reliable software in less time and at lower costs. It gives us a potential better way of doing business. Normally, innovation is associated with higher costs but that's exactly the wrong way to looking at it. This is especially true if the company and finally the customer do not appreciate the change. It is more common for an automotive company, for example during a crisis period, to look at the innovation investment, as an item to be subject to an eventual optimization (reduction) cost process. In order to protect and to reduce the risk of reduction of innovation, it could be useful to allocate the above costs among all the SW development projects.

TABLE IV. SW LIFE CYCLE COST

Software Engineering Development Effort Element	**0.75 Confidence**	**0.85 Confidence**	**0.95 Confidence**	**0.995 Confidence**	**Comments**
	Effort Hrs	***Effort Hrs***	***Effort Hrs***	***Effort Hrs***	
Project management					3.2 % of Total Project Estimate
Facility Administration					1.5 % of development Estimate
SW Process Enhancement & Innovation					2 % of Development phase Estimate
SW Development Dependent phases					as from previous sheet;
SW Engineering Total Effort Estimation					

TABLE IV defines the cost for other Organizational support processes, as Project management, Facility Administration, Sw Process Enhancement & Innovation, depending of the direct SW development cost. The above costs are hardly specific of the Organization structure. The values, reported in the comments Column, are based on our experience. The Cost estimation is based on the assumption that the team will be composed of experienced (more 1+ year) MBD Software Engineers. In case of inexperienced MBD SW engineers, we need to consider the learning process cost.

VI. CONCLUSIONS

Today, many software development Managers have problems in providing accurate and reliable cost estimates, and, therefore sometimes do not undertake estimation at all. Besides, the existing cost estimation methods and tools are more complex and not customized for each specific software development process. Our approach aims at yielding more reliable estimates, based on the experience of all actors involved in the software development life cycle and it is based on the phases of the software life cycle. The estimation process improves continuously with the availability of more data and it continuously adjusts itself to the evolution of the software development phases. The first time cost estimation can be done is at the beginning of the project after the requirements have been outlined. Cost estimation may even be done more than once at the beginning of the project. For example, several companies may bid on a contract based on some preliminary or initial requirements, and then once a company wins the bid, a second round of estimation could be done with more refined and detailed requirements. Doing cost estimation during the entire life cycle allows for the refinement of the estimate because there is more data available. Periodic re-estimation is a way to gauge the progress of the project and whether deadlines will be able to be met.

Effective monitoring and control of the software costs is required for the verification and improvement of the accuracy of the estimates. Tools are available to help organize and manage the cost estimates and the data that is captured during the development process. People are less likely to gather data if the process is cumbersome or tedious, and so using tools that are efficient and easy to use will save time. It is not always the most expensive tool that will be the best tool to buy, but rather the tool that is most suited to the development environment. Therefore, the success of our proposal is not necessarily the accuracy of the initial estimates, but rather the rate at which the estimates converge to the actual cost. We are using the proposed approach for our Software development projects for All Vehicle ECUs (i.e., Engine Control unit, Vehicle computer Module). Therefore, we have a very valuable database reflecting our distribution cost in all phases of the software life cycle. These data are used to develop a software cost estimation model tailored to all CNH Industrial applications.

REFERENCES

[1] D. Cortese, "New Model-Based Paradigm: Developing Embedded Software to the Functional Safety Standards, as ISO 26262, ISO 25119 and ISO 13849 through an efficient automation of Sw Development Life-Cycle" SAE Technical Paper 2014-01-2394, doi: 10.4271/2014-01-2394

[2] D. Cortese, "ISO 26262 and ISO IEC 12207: The International Standards Tailoring Process to the whole Sw Automotive Development Life Cycle by Model-Based Approach" SAE Technical Paper 2011-01-0053, 2011, doi:10.4271/2011-01-0053

[3] D. Cortese, "Model-based Approach for the realization of a Collaborative repository of All Vehicle Functionalities" FISITA Technical Paper F2008-05-039, 2008

[4] Road Vehicles – Functional Safety - International Standard ISO 26262 : 2011

[5] Tractor and machinery for agriculture and forestry – Safety-related parts of control systems – International Standard ISO 25119: 2010

[6] Safety of machinery – Safety-related parts of control systems – International Standard ISO 13849: 2006

[7] CNH Industrial Web site: http://www.cnhindustrial.com/it-IT/Pages/homepage.aspx

[8] D. Cortese, Iveco Develops a Shift Range Inhibitor System for Mechanical 9- and 16-Speed Transmissions in Six Weeks, “https://www.mathworks.com/tagteam/71432_91989v00_IVECO_UserStory_final.pdf

[9] D. Cortese, “Developing Embedded Software to International Standards and On-board Vehicle Software Architectural Standardization” Course for PH.D. Program in Computer Science, 2013, http://dott-informatica.campusnet.unito.it/do/corsi.pl/Show?_id=1f5e