

Metric-based Evaluation of Software Architecture for an Engine Management System

Hariharan Venkitachalam
RWTH Aachen Univ.

Dirk von Wissel
Renault SA

Johannes Richenhagen
FEV GmbH

ABSTRACT

Powertrain software development for series production faces multifaceted challenges related to high functional complexity, high quality standards, reduced time to market and high development costs. Software architecture tackles the above mentioned challenges by breaking down the complexity of application software into modular components. Hence, design errors introduced during that phase cause significant cost and time deviations. Early and repeated analysis of new and modified architecture artifacts is required to detect design errors and the impact of the subsequent changes in the software architecture.

Engine management software has a high degree of functional complexity and large number of system variants depending upon market requirements. This paper deals with the methods to perform automated evaluation of Renault's EMS 2012 Engine Management Software in a Continuous Integration Framework. The adopted approach enables frontloading of development effort, avoids quality risks at a later stage of development and reduces time and cost for series software development.

CITATION: Venkitachalam, H., von Wissel, D., and Richenhagen, J., "Metric-based Evaluation of Software Architecture for an Engine Management System," *SAE Int. J. Engines* 9(3):2016, doi:10.4271/2016-01-0037.

1. INTRODUCTION

The first deployment of software in automotive industry was for the ignition control of engines [1]. Since then, due to emission regulations and market demand, the functional complexity of an engine management system is increasing exponentially [2]. Figure 1 provides a history of the number of calibration variables in the engine management software which demonstrates the rising complexity over years of development.

Modern engine management systems have complex functionality with large number of calibration variables, large number of system variants, communication with sensors, actuators, other control units and On-Board Diagnostics for efficient error recognition [3]. Market-specific requirements have created high degree of variability in the engine control software and hardware capabilities. These factors necessitate a high level of flexibility for the engine management software. Traditionally, the development time for automotive software was three to five years [4]. However, shorter time to market has reduced the time available for software development considerably.

Software architecture helps tackling the above mentioned concerns by breaking down the complexity of the engine control software. This is achieved by organizing the software into modular software components with limited interactions between them thereby improving quality attributes like maintainability, testability and portability. In addition to that, software architecture enables identification of reusable software assets across platforms to reduce development time and costs.

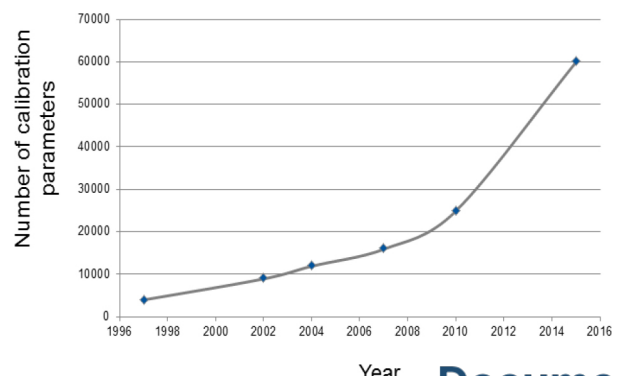


Figure 1. Increase in engine calibration variables over time [5]

Renault EMS 2010 software architecture focused on reuse of powertrain application software across vehicle platforms with great gain in costs, stability and quality. Renault EMS 2012 is the next generation platform of engine management systems with a focus on automatic code generation which reduces the costs of development further [6].

There exist various methods in control theory to reduce functional complexity using early verification and validation of control algorithms [7] [8] [9]. However, these methods are not focused on non-functional attributes of engine control software. This paper focuses on the evaluation of non-functional quality attributes of the Renault EMS 2012 engine management software architecture through a metric-based approach. The subsequent sections in this paper explain in detail the systematic approach used for the evaluation of architecture. Section 2 deals with the requirements for evaluation of powertrain software architecture. Section 3 contains details of the state-of-the-art in the domain of software architecture evaluation for automotive applications. Section 4 explains the EMS 2012 SW Architecture concept and the boundary conditions for this work. Section 5 deals with the metric-based evaluation of the EMS 2012 SW Architecture. Section 6 concludes the work with the benefits and limitations of the metric-based approach and future development goals. Section 7 deals with future development of the metric-based software evaluation concept.

2. PROBLEM STATEMENT

Software Architecture is a work product of the early design phase. The impact of architecture on a project is profound [10]. The design errors introduced in software architecture are detected only at the later stages of development. This results in rework of software artifacts and hence additional costs. To reduce cost and time for development, we need to evaluate the architecture at an early stage before proceeding to the next stage of development. This leads to the research question (RQ1) as mentioned below

RQ1: How to evaluate engine management software architecture at the early stages of development?

The number of electronically controlled devices and functional requirements associated with the powertrain is increasing. As a result, the functional complexity of the powertrain software is also increasing. Hence, new software components need to be integrated into the architecture with legacy components which have been developed over many years. Modern engine management systems consist of thousands of software interfaces, which must be compatible with each other. With increasing complexity, verification of the compatibility of interfaces between the software components becomes a complex and time consuming task. A manual verification of such a high number of interfaces is not feasible. Hence the architecture evaluation must have a highly automated tool chain to support the evaluation of components and their interdependencies. An automated evaluation of the software architecture can be only

achieved if the component interfaces are defined in a standard format. The architecture evaluation method must also be capable of ensuring the completeness of the specification.

Market-specific requirements like emission legislations and diagnostic requirements have created a high degree of variability in the software components developed for engine management systems. Hence, architecture evaluation for engine management software must be capable of handling variants of software components.

An overarching part of architecture evaluation is to prioritize specific quality goals that the architecture must meet in order to be considered suitable. Ideally, these quality goals (functional and non-functional) are defined in detail for each control unit. However, this notion fails due to the lack of complete requirements. In most cases, only the system requirements are available and some of the quality requirements are not explicitly stated [10]. The important engine management software quality characteristics are safety, reliability, resource utilization, modifiability and reusability [3]. With increasing complexity of engine management software, rigorous processes need to be implemented to ensure quality. At the earliest stages of design, the standards and design guidelines [11] to be fulfilled by software components are fixed by the software architect. Over the span of a project, the extent to which the architecture of an engine management system fulfils the quality goals outlined by the standards with respect to time is not monitored. This leads to deviations in software quality leading to additional development costs. To avoid this condition, a continuous and systematic method for analysis of software architecture is required. The second research question (RQ2) deals with the automation of architecture analysis as shown below

RQ2: How to automate the evaluation of architecture in order to deal with high complexity and evolution of engine management systems?

In powertrain software domain, the components are logically grouped together based on functional requirements. The functional decomposition of software into various components has an impact on the non-functional attributes like maintainability, testability, and analyzability. An objective feedback is required to determine the impact of functional decomposition on these quality attributes. This leads to research question 3 (RQ3) as shown below

RQ3: How to obtain an objective feedback regarding the quality attributes associated with engine management software architecture?

Architecture development in powertrain software domain is driven by the focus on handling functional complexity, reducing development time and costs, improving software quality, handling system variants and facilitating software sharing between various project stakeholders. The method used for evaluating the software architecture must also be aligned to meet the challenges.

Table 1 summarizes the decision matrix between factors driving architecture development and requirements on software architecture.

Table 1. Decision matrix for SW Architecture Evaluation

Factors driving architecture Development	Requirements on SW Architecture Evaluation					
	Objective Evaluation of Quality Goals	High level of toolchain automation	Early detection of errors	Variant Handling for Architecture Evaluation	Well defined interfaces	Analysis of legacy systems
Increasing functional complexity	X	X	X	X	X	
Shorter Development times		X	X		X	
Lower Costs		X	X		X	
SW Quality Requirements	X				X	
Handling of System Variants				X	X	
Reusability of SW Components	X				X	X
SW Sharing between stakeholders					X	

Another important task which follows such an architecture evaluation is to identify the problems in the software architecture and to suggest changes in the software architecture.

RQ4: How to refactor software components based on architecture evaluation?

3. STATE-OF-THE-ART

The popular method for architecture evaluation is the scenario based method where stakeholders develop scenarios or use-cases to evaluate the fulfilment of the architecture goals. The key quality goals are articulated by the stakeholders and conflicting goals are prioritized. This method uncovers possibilities of cross-project reuse and evaluates the design guidelines used for architecture development. The scenario based evaluation is performed at the start of the project. Among the existing scenario based evaluation methods, there are multiple instances of industrial deployment for ATAM (Architecture Trade-off Analysis Method). The approximate size of a medium size ATAM is estimated to be 70 Man Days [10]. However, a scenario-based approach does not fit the requirements for architecture evaluation in the powertrain software as mentioned in section 2. Hence a metric-based approach to evaluate software architecture is preferred.

The measurement process for the EMS 2012 engine management software must be in accordance with the ISO 15939 standard [12] [13]. The metric data must fulfil certain quality characteristics to be considered for automotive software development like simplicity, precision, objectivity, robustness and configurability [14]. Metrics express the degree of fulfilment of a quality attribute through a quantitative measurement of software artifacts. There exist various approaches to define metrics like Factor-Criteria-Metric [15] and Goal-Question-Metric [16]. ISO 25010 [17] standard explains the quality attributes that affect commercial off-the-shelf software. The previous works of the authors benchmark the definition of quality attributes based on the ISO 25010 [17] and the ISO 26262 [18] standards and internal design guidelines [19]. The Goal-Question-Metric approach was adopted by the authors to define metrics in the past [20].

Model based development is gaining prominence in powertrain software development due to the possibility of auto-code generation from models thereby reducing development costs. Metrics to evaluate the quality attributes related to Simulink models already exist in the automotive domain [21] [22] [23]. According to the V-Model for software development, development and evaluation of model quality happens at later stages of development. Any design error detected during model

quality evaluation will cause additional effort of rework of the preceding stages of development (i.e. requirements or architecture). Hence an early evaluation at the architecture level is important.

4. APPLICATION DOMAIN

Embedded software architecture in automotive domain can be broadly divided into application and basic software. Application software contains the control laws and strategies which support the system functionality. The basic software manages the hardware resources and communication services. EMS 2012 is modular software architecture from Renault focused on reuse of application software components thereby reducing cost of development. The EMS 2012 software architecture has more than 250 components in the application software, which were developed in a distributed development environment. It is based on the EMS 2010 platform with the additional possibility of auto-code generation from models [24]. Hence from an architectural standpoint, the platform architecture concepts of EMS 2010 and 2012 are similar. Hence all the concepts applied in this paper are valid for both architectures. Figure 2 shows the architecture of EMS 2010.

EMS 2012 has three levels of composition. The highest level is the function followed by sub-functions which are composed of modules. Modules are atomic elements of software architecture with a well-defined functionality and interfaces [6].

This metric-based architecture evaluation focuses on applying static metrics for the reusable modules of the EMS 2012 platform. There are some boundary conditions imposed by the third party evaluation of software architecture. The libraries, interfaces to the EEPROM, and scheduler calls have not been considered as a part of the study. The project specific modules which undergo adaptation for various applications have been considered to understand signal flows between modules. However, the metric evaluation of these modules is not the focus of this study and hence will not be discussed in this paper. Additional stubs were inserted for modules which were missing in the interface specification based on discussion with functional experts.

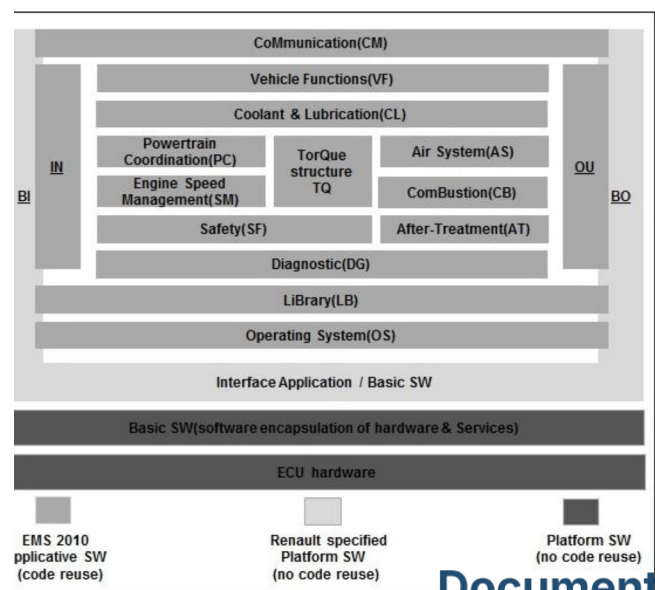


Figure 2. EMS 2010 Software Architecture [Wise Document Sharing Hub]