

Metric-based Evaluation of Powertrain Software Architecture

Hariharan Venkitachalam and Christian Granrath
RWTH Aachen Univ.

Balachandar Gopalakrishnan
Technische Hochschule Ingolstadt

Johannes Richenhagen
FEV GmbH

ABSTRACT

Ensuring software quality is one of the key challenges associated with the development of automotive embedded systems. Software architecture plays a pivotal role in realizing functional and non-functional requirements for automotive embedded systems. Software architecture is a work-product of the early stages of software development. The design errors introduced at the early stages of development will increase cost of rework. Hence, an early evaluation of software architecture is important.

PERSIST (Powertrain control architecture Enabling Reusable Software development for Intelligent System Tailoring) is a model-based software product line approach which focuses on cross-project standardization of powertrain software. The product line is characterized by common design guidelines and adherence to industry standards like ISO 25010, AUTOSAR and ISO 26262. This paper shows a systematic approach to ensure objective, early and repeated analysis of software architecture for the PERSIST product line using metrics. The systematic approach for architecture metrics involves the definition, implementation, measurement and, evaluation of the metrics for hybrid control unit software. With a new approach of architecture-driven development, this paper demonstrates how an early evaluation of the software architecture helps to frontload development efforts and improves software quality.

CITATION: Venkitachalam, H., Granrath, C., Gopalakrishnan, B., and Richenhagen, J., "Metric-based Evaluation of Hybrid Control Software Architecture," *SAE Int. J. Passeng. Cars – Electron. Electr. Syst.* 10(1):2017, doi:10.4271/2017-01-1615.

1. INTRODUCTION

The complexity of powertrain software is increasing exponentially with respect to time. The evolution of new technologies which enable communication between vehicles (Car-2-Car communication), information from the traffic infrastructure (Car-2-X communication), and cloud-based services has increased the number of data inputs to the powertrain [1]. With the availability of more data, complex control algorithms will be used to optimize the fuel consumption and reduce the exhaust gas emissions. In addition, there exist market-specific requirements and emission legislations which require a complex variant management for powertrain control software.

A shorter time-to-market has reduced the time available for software development. Hence, there is a need to reuse core software assets across various projects to meet stringent software development deadlines. Thus, modularity and reusability of software are key organizational goals which help to meet the demands of automotive software development within the time constraints. Moreover, the development process needs a high degree of automation to meet the requirements of automotive software development.

Given the reduced time-to-market and the high complexity of powertrain software, ensuring software quality is one of the key concerns of Original Equipment Manufacturers (OEMs) and Tier-1 suppliers. The control software has to fulfill various standards, functional and non-functional requirements (derived from standards and organizational goals) before it is deemed road-worthy. Software architecture helps in achieving the organizational and quality goals by ensuring by breaking down the complexity, improving reusability and other non-functional quality attributes such as maintainability, testability, and portability [2][3].

PERSIST (Powertrain control architecture Enabling Reusable Software development for Intelligent System Tailoring) [4] is a model-based product line for powertrain software. The design guidelines for PERSIST are based on industry-specific standards like ISO 26262 [5], AUTOSAR [6] and ISO 25010 [7]. It outlines architecture design guidelines for the development of powertrain application software. However, there has been little focus on the objective evaluation of software architecture based on the quality

goals and design guidelines. The metric-based approach objectively evaluates the architecture based on quality criteria like modularity, testability, and maintainability.

This paper deals with the objective evaluation of hybrid control software architecture using metrics. Section 2 deals with the requirements for architecture evaluation in the powertrain software domain. Section 3 compares the state-of-the-art methods existing for software architecture evaluation. Section 4 provides an overview of the hybrid control software. The solution approach for the definition, implementation, and evaluation of metrics is explained in section 5. Section 6 focuses on the results, validation methods and critical reflection of the metric-based approach for evaluation of software architecture. Section 7 concludes the work with the advantages of metric-based approaches for automotive software development. Section 8 deals with the future improvement required for the metric-based software architecture methods for powertrain applications.

2. PROBLEM STATEMENT

Automotive software needs to fulfill various quality requirements and standards in order to be considered worthy of production. Hence, ensuring software quality is one of the key concerns of vehicle manufacturers and suppliers. Architecture definition is one of the early activities in the software development lifecycle. Software architecture development involves decomposition of a complex functionality into smaller and manageable parts. Architectural decisions made at the early stages of a project have a deep impact on the structural and dynamic aspects of the software. 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. Hence, in order to reduce time and development cost, we need to evaluate the software architecture early.

An overarching part of architecture evaluation is to prioritize specific quality goals that the architecture must meet in order to be considered suitable [8]. Ideally, these quality goals (functional and non-functional) are defined in detail for each electronic control unit. However, this notion fails due to the lack of complete requirements. In the automotive domain, only parts of the quality requirements are explicitly stated. Hence, a comprehensive view of software quality is necessary to meet the quality demanded by automotive standards. This leads to the first research question (RQ1) as mentioned below

RQ1: What are the quality requirements of software architecture in powertrain domain?

The quality attributes of software architecture need to be measured in order to evaluate the fulfillment of quality goals. A repeatable and data-driven approach is required to improve the objectivity and reproducibility of architecture evaluation. Metric-based evaluation methods provide an objective feedback by measuring quality characteristics. According to IEEE, software quality metric is a function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given attribute that affects its quality [9]. A metric defined must fulfill certain fundamental qualities like

simplicity, precision, objectivity, robustness and configurability to be considered for adoption into the automotive software development [10]. The objectivity of the measurements depends on identifying causal relationships between various architectural parameters that constitute a metric and the quality attributes. Such causal relationships are necessary to establish the credibility of the metric-based approach. This leads us to the second research question (RQ2) mentioned below

RQ2: How should metrics be defined in order to evaluate the quality of software architecture?

There exist various system topologies and control strategies for hybrid vehicles [11]. Hence variants of software components will be used to meet the diversity of control requirements for hybrid vehicles. Hence, the method used for the architecture evaluation must be capable of handling software variants.

In powertrain software development, a three-level monitoring concept is implemented which consists of function, function monitoring and controller monitoring. Level 1 contains control functions like implementation of the requested engine torque, input and output diagnostics and system reactions to a fault. Level 2 detects the defective process of functional software by monitoring the calculated torque values or unintended acceleration. In case of fault, system reactions are triggered. The monitoring controller which represents level 3 is independent of the function controller which tests the correctly executed program [27]. The same monitoring concept is also used for the control of hybrid vehicles. The architecture metrics discussed in this paper deal with the evaluation of quality attributes associated with the Level 1 and Level 2 software components.

The complexity of the hybrid control algorithm is increasing with the addition of data to the powertrain from various external sources like traffic infrastructure and inter-vehicle communication [1]. The data availability from external sources leads to modification of functional requirements. As a result, new software functionalities are included leading to a continuous evolution of hybrid architecture. Hence, the architecture evaluation method must be capable of handling the complexity and evolution of software architecture.

The extent to which the architecture of hybrid control software fulfills the quality goals (for e.g. testability) is not monitored. In addition to this, powertrain software development involves the integration of software from various suppliers. Changes introduced by individual suppliers for a given revision can have an impact on overall software quality. The lack of continuous architecture checks leads to the degradation of software quality and subsequent increase in development costs. Hence, continuous evaluation of the software quality attributes is necessary.

In addition to this, the measured values of the quality attributes must be compared against a threshold value to determine outliers. The determination of the right thresholds for individual metrics holds the key to identifying quality problems in the software architecture. Hence, a systematic method needs to be employed to determine the threshold values for the architecture metrics. The measured and

evaluation of software quality involve identification of architecture metrics and their corresponding thresholds. This leads us to the third research question (RQ3) as mentioned below

RQ3: How to evaluate the quality of software architecture continuously?

The ease of integration of metrics into the development process is another key acceptance criterion for powertrain software development. As mentioned in [Section 1](#), development cost is one of the key concerns of automotive software suppliers. Hence, the metrics to measure architectural quality must be seamlessly integrated into the development process. This leads us to the fourth research question (RQ4) mentioned below

RQ4: How to improve the ease of integration of architecture metrics in the powertrain software domain?

The metric values obtained by analyzing the architecture must be used to further refine the architecture based on the prioritization of quality goals. This leads us to the next research question

RQ5: How should quality attributes of individual software components be improved based on metric values?

3. STATE-OF-THE-ART

Software architectural information can be specified using informal, semi-formal or formal methods. Depending upon the method used to specify, the quality and the level of architectural information at the early stages of development varies. In the automotive industry, informal or semi-formal methods are used to specify software architecture. Formal methods are in general used for research purposes.

Evaluation of software architecture is divided into two broad concepts namely scenario-based and metric-based methods.

3.1. Scenario-based Evaluation Methods

The popular method used for architecture evaluation is the scenario-based method. In this method, the stakeholders develop scenarios to evaluate the software architecture. The key quality goals are articulated and prioritized by the stakeholders within the organization. Various scenarios are derived based on the relevant quality requirements prioritized by the stakeholders through discussions. It can also be verified if an architecture design guideline fits the goals of the organization using this method.

The scenario-based evaluation is generally performed at the start of the project. Among the existing scenario-based evaluation methods, there are multiple instances of industrial deployment for Architecture Trade-off Analysis Method (ATAM). The approximate effort estimated for a medium size ATAM is estimated to be 70 Man Days [8]. Since the ATAM involves multiple stakeholders in varying capacities, the time required by each stakeholder cannot be easily estimated.

Given the fact that the powertrain software is evolving continuously, the evaluation method must be capable of handling architectural evolution. Secondly, the scenarios depend on the stakeholders who are involved in the architecture evaluation. Hence, the outcome of a scenario-based evaluation cannot be reproduced.

3.2. Metric-based Evaluation Methods

Metrics express the degree of fulfillment of a quality attribute through a quantitative measurement of software artifacts. The strength of the metric-based approach lies in the objectivity, reproducibility and tool-based support which improve the possibility of tracking the evolution of software architecture. The software architecture description must be formalized in a machine readable format to analyze metric data. However, the metric-based approach is not intended to replace the scenario-based evaluation methods of SW architecture. Instead, the metric and scenario-based approaches can co-exist as complementary solutions for powertrain software architecture.

A metric can be defined by using various structured approaches. The most popular approaches available are Factor-Criteria-Metric [12] and Goal-Question-Metric [13]. The authors have chosen the Goal-Question-Metric [13] approach which serves as a logical connection between the goal and the metric.

The PERSIST product line is defined based on various standards like ISO 25010 [7], ISO 26262[5] and AUTOSAR. In addition, PERSIST also specifies certain design guidelines and it's prioritization for software architects. The previous works of the authors benchmark the definition of quality attributes based on the ISO 25010 [7] and the ISO 26262 [5] standards and internal design guidelines [4]. The PERSIST tool framework is based on the concept of Continuous Integration, based on agile development principles.

Metrics to evaluate the quality attributes related to Simulink models and static code analysis already exist in the automotive domain [14] [15][16]. Some of these metrics can also be evaluated at early stages of development as the information is already available. Such an effort to transfer evaluation of software at an earlier stage is termed as frontloading. The key advantage of architecture metrics lies in frontloading the effort of evaluation thereby reducing the cost of rework.

4. APPLICATION

The software architecture of automotive embedded systems can be classified into two broad layers namely application layer and basic software layer. Basic software layer deals with the communication management, hardware abstraction, diagnostics, state management of the control unit and operating system. There are already ongoing efforts to standardize the basic software layer like AUTOSAR to achieve scalability and transferability of software [6]. The basic software is generally provided by the supplier of the control unit hardware. Due to confidentiality clauses, the internal working of the basic software is abstracted from the application software developer. Software interfaces between the application and basic software facilitate information exchange between the two broad layers.

However, the basic software still remains a black box for most of the application software developers. The architecture metric evaluation of the application software also considers the basic software as a black box component.

The application software architecture for the hybrid powertrain is provided in Figure 1. The PERSIST hybrid architecture is hierarchically structured into layers, compositions, components, runnables and functions. The concept of layers in PERSIST is explained in the previous paragraph. The composition is a virtual grouping of high-level software functionality existing across various software components. The focus of PERSIST architecture is to identify the potential for reuse at the level of a software component. A software component is a virtual structure comprised of various runnable entities which interact with each other. Each runnable consists of individual units which realize one or more vehicular functional requirements.

The control software architecture of the hybrid vehicle consists of various software components which perform interpretation of inputs from the driver, coordination of torque and operating modes of the powertrain, thermal management, application-level error management and coordination of electric supply.

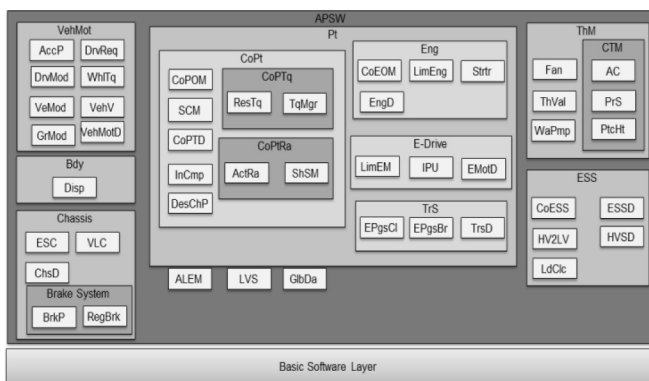


Figure 1. Application software architecture of the hybrid control unit

5. SOLUTION APPROACH

The key motivation to adopt a metric-based evaluation is to perform early, objective and repeated evaluation of software architecture. The solution approach for the evaluation of hybrid control software architecture is divided into four general parts. The individual parts of the solution are explained below.

5.1. Definition of a Quality Model and Metrics

This section of the solution approach addresses the following aspects, namely definition of the quality model, mapping of architectural parameters with quality criteria and definition of the metrics. The key research questions addressed in this sub-section are listed below.

- RQ1: What are the quality requirements of software architecture in powertrain domain?
- RQ2: How should metrics be defined in order to evaluate the quality of software architecture?

5.1.1. Definition of Quality Model

Powertrain software has to comply with various automotive standards and guidelines to ensure quality. At the stage of architecture definition, the focus is to prevent design errors by complying with standards and guidelines. Architecture goals are derived from standards like ISO 25010, ISO 26262 and design guidelines. Figure 2 describes the quality model for the automotive software architecture. The overall structure of the quality model is based on the guidelines provided by the ISO 25010.

The ISO 26262 standard also describes various quality attributes for software architecture as given below.

- Bidirectional traceability
- Restricted size of software components
- Restricted size of interfaces
- Low coupling and high cohesion

Bidirectional traceability establishes a relationship between heterogeneous artifacts in the automotive software development and hence ensures a better maintainability. Modularity already covers aspects related to high cohesion and low coupling of a software component. Hence, the quality characteristics outlined by the ISO 26262 are already covered by ISO 25010. Simplicity ensures that a software component will have a restricted size and that it will be easy to maintain. Hence, these characteristics are included in the quality model.

The definition of a comprehensive quality model (in Figure 2) addresses the Research Question 1 (RQ1) mentioned in section 2. The output of this step will contain a mapping of the standards and the individual quality attributes required for powertrain software development. The quality attributes which are measurable from a static architecture specification are identified at this stage. The static architecture specification consists of the various software components and the data flows between them.

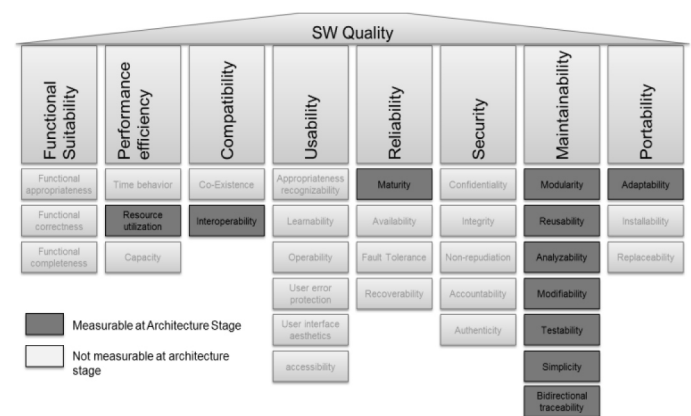


Figure 2. Quality Model for Automotive SW Architecture

5.1.2. Mapping between Quality Attributes and Architectural Parameters

The quality model shown in Figure 2 outlines the attributes that constitute quality of powertrain software architecture. Once the quality attributes are defined, the measurable parameters of the architecture which influence them are identified. The measurable parameters could be as simple as the number of inputs, outputs, units or coupling between components. These parameters provide an objective feedback regarding the fulfillment of the quality characteristics. Based on the quality model, the architectural parameters that affect the quality attributes are outlined. Following this, a mapping of architectural parameters and their corresponding quality attributes is performed. A combination of various architectural parameters that affect a quality attribute can be used to develop a metric.

The architectural parameters can be evaluated at various abstraction levels based on the architecture model. Figure 3 shows the structural elements of the architecture. The smallest functional element of the software is termed as a unit. A set of inter-related units with a definite execution time are grouped in a runnable. This is however in contrast to the concept of runnable in AUTOSAR [6], where the runnable is the atomic element of the architecture. The units are connected with each other through the inter-functional interface. A component consists of multiple runnables and external interfaces (inputs and outputs) to other components. A composition consists of various inter-related components. For complex vehicle control domains like the powertrain, a set of nested compositions are used to create a hierarchical and understandable structure of the software. The application software is comprised of many such compositions.

Table 1 provides examples of mapping between quality characteristics, abstraction levels, and architectural parameters. Similarly, hundreds of architectural parameters and their relationship with various quality criteria were analyzed.

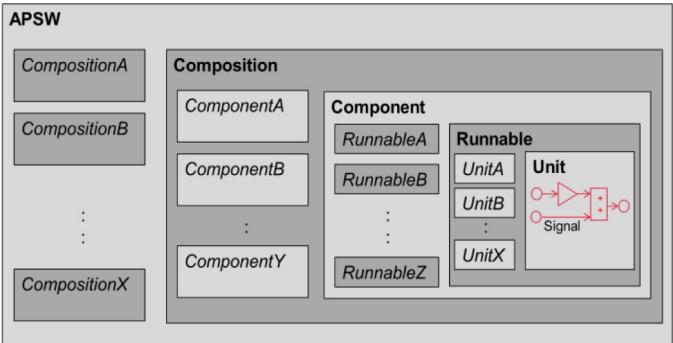


Figure 3. Overview of the abstraction levels of the metric based on architecture meta-model

Table 1. Example of mapping between quality attributes and architectural parameters

Quality Attributes	Abstraction level	Architectural Parameters
Simplicity	Software component	Number of units in a component
		Number of input interfaces
		Number of output interfaces
		Number of calibration variables
		Number of inter-unit interfaces
Cohesion	Software component	Number of units in a component
		Number of functional requirements connected to the software component
		Number of inter-unit interactions
Maintainability	Signal	Number of components using a signal

The output of this step will contain the following information

- Abstraction level of the metric
- Architectural parameters
- Mapping of the architectural parameters to the quality attributes

5.1.3. Definition of Metrics

Goal-Question-Metric [13] approach is used to define metrics based on the attributes defined in the quality model. The individual quality attributes are the goals against which the architecture is measured. The identification of architectural parameters in section 5.1.2 is an important step in defining the metric.

In automotive software development, there are some artifacts which are better maintained along the lifecycle of a project and are machine readable. Such artifacts must be chosen as the source to extract the architectural parameters. A metric is expressed as a mathematical equation consisting of various architectural parameters. The metric can have a direct or an inverse relationship with a given quality criteria based on which the mathematical equation is defined. The