IEEE Open Journal of the Systems Engineering

Received 19 August 2022; revised 6 July 2023; accepted 11 January 2024. Date of publication 23 January 2024; date of current version 26 February 2024.

Digital Object Identifier 10.1109/OJSE.2024.3357243

ALCEA: The Architecture Life-Cycle Effect Analysis Method

JAKOB AXELSSON [1,5], DAMIR BILIC [2,5], DANIEL BRAHNEBORG [3] (Member, IEEE), JOAKIM FRÖBERG [4], HENRIK GUSTAVSSON [5], ROBBERT JONGELING [5] (Member, IEEE), AND DANIEL SUNDMARK [5]

[1]RISE Research Institutes of Sweden, SE-164 29 Kista, Sweden
[2]Volvo Construction Equipment, SE-63185 Eskilstuna, Sweden
[3]Braxo AB, 118 64 Stockholm, Sweden
[4]Safety Integrity AB, SE-722 35 Västerås, Sweden
[5]Department of Innovation, Design, and Engineering, Mälardalen University, SE-721 23 Västerås, Sweden

CORRESPONDING AUTHOR: ROBBERT JONGELING (e-mail: robbert.jongeling@mdu.se.)

This work was supported by Mälardalen University.
This article has supplementary downloadable material available at https://doi.org/10.1109/OJSE.2024.3357243, provided by the authors.

ABSTRACT This article describes the architecture life cycle effect analysis (ALCEA) method, a structured method for evaluating proposed new architectures for software-intensive systems. The method evaluates a proposed architecture by quantifying its effect on the performance of system life-cycle phases. The method is instantiated by identifying the relevant life-cycle phases of the system under investigation and a set of evaluation functions that capture, in terms of basic factors, the effect of different architectural decisions on key life-cycle PAs, such as revenue, operating resources, and investments. The method results in a transparent cost and revenue structure, documented in a tabular form, based on quantifiable factors from the developing organization. The results of the method can be used directly as part of a business case, and their robustness can be estimated by sensitivity analysis. The ALCEA method is designed for system-level architectural analysis, covering both software and hardware aspects. In this article, we introduce the ALCEA method and provide a detailed example of how to apply it in the evolution of embedded systems. Moreover, we share early experiences of using the method in large-scale industrial settings.

INDEX TERMS Architecture analysis, computer architecture, full life-cycle support, systems engineering.

I. INTRODUCTION

Software-intensive embedded systems play an increasingly important role in many companies that develop technical products in industries such as automotive, defense, process automation, telecommunications, and transportation. The functionality of these systems is mostly provided by software, often running on a large number of computers, interconnected using both wired and wireless links. Each such computer, or processor node, can run software consisting of millions of lines of code and have many information-intensive actuators and sensors. The system needs to be packaged in whatever physical space is available in the product and needs to be equipped with an adequate power supply system. This packaging may require multiple modes of operation and electrical requirements that have implications for the software in different ways.

In addition, systems often must provide features and qualities to support service operations, testing efforts, and other phases of system usage. This means that a substantial part of the design effort deals with supporting other life-cycle phases beyond normal operations such as evolution or maintenance, and these processes can be both costly and provide additional revenue. It is, therefore, business-critical that the embedded system supports all these life-cycle phases of the product.

The emphasis of modern system architecture thus lies on supporting all life-cycle phases. Moreover, the various operating domains of these systems increase the need to create multiple variants of the system. Consequently, architecture requires an increasing amount of business attention in companies developing software-intensive systems. ISO/IEC 42010 defines an architecture as "the fundamental concepts or properties of a system in its environment embodied in its elements,

relationships, and in the principles of its design and evolution" [1]. In other words, it shows the following:

1) the high-level structure of important parts of the system;
2) the internal and external interfaces of the system;
3) the general design principles that are to be followed in both current and future versions of the system.

Given the importance of system architecture, an objective way to compare alternatives is required. An evaluation spanning all life-cycle phases would provide support when making decisions about how to build the system. Combined with existing work [2], we list the following generally desirable properties for an architectural evaluation method.

Lean: The effort should impose a reasonable footprint; the resources needed for the analysis should be relatively small compared to the value provided by the analysis.

Fact-based: As far as possible, the analysis should be based on facts and quantifiable measures rather than on a gut feeling of the individuals involved.

General: The method should not make assumptions about the system to allow it to be applicable to different types of systems. It should also be general enough to be applicable early in development, as well as possible to reuse in future architecture analysis when better estimates and measures become available during the system evolution.

Transparent: It should be possible to derive from the architecture analysis how the conclusions were reached and what the rationale was for a certain evaluation.

Several methods exist for architecture evaluation, as will be discussed in more detail in Section V. These methods are often based on quality attributes (QAs) such as modifiability and testability. However, we observed that many companies developing complex embedded systems have well-established life-cycle models, where companies collect data on the performance of the different phases. The properties of these life-cycle phases correspond well to the QAs. Therefore, we propose a method that can improve the state-of-the-art by benefiting from existing knowledge on life-cycle phases for architectural analysis.

The purpose of this research is to find an appropriate method that can be used for the evaluation of system architecture proposals. In addition to the four desirable properties mentioned previously, the method should cover the relevant aspects for software-intensive systems, including embedded systems where not only software, but also electronic hardware is of importance. Furthermore, the method should be applicable to product lines and the evolutionary development of a system.

We propose the architecture life-cycle effect analysis (ALCEA) method, where we analyze the architecture based on its performance in each life-cycle phase of the system. The analysis is quantified in economic terms, since the choice of architecture is an investment decision that requires a business case. Finally, our approach takes into account the whole system and not just the software. We exemplify the ALCEA method on architectures of software-intensive embedded systems, but possible applications are not limited to that domain.

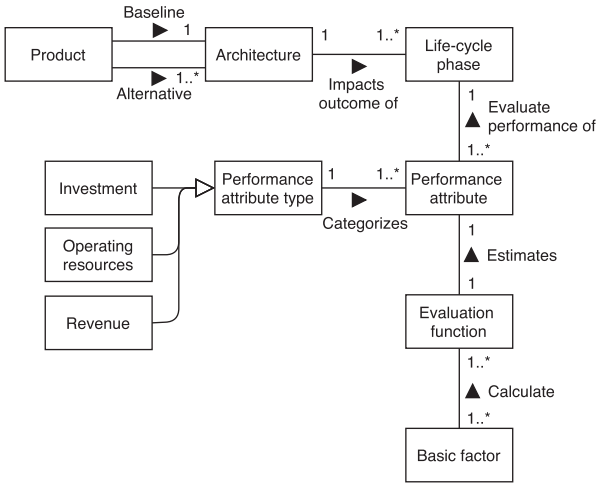

FIGURE 1. Information model (using UML notation) for architecture evaluations using ALCEA.

II. ALCEA METHOD

This section presents the ALCEA method in detail. We first present an overview of the basic principles of the ALCEA method and define a process for using it. We then elaborate on how to work with the method and provide forms for doing so. Finally, we show how conclusions can be drawn from an analysis and how to check the robustness of these conclusions.

A. BASIC PRINCIPLES

The ALCEA method fundamentally supports architectural decisions by comparing the costs and revenue incurred by each architectural alternative in all relevant life-cycle phases (see Section III for an example application of ALCEA). As described in Section I, the architecture analysis should be based on the degree to which it supports a well-functioning system. We emphasize that this is not limited to functioning during the operational phase of the system. Instead, for an analysis to be complete, it must span all life-cycle phases of importance, including, e.g., system development and maintenance.

Fig. 1 shows the basic information model used to perform architectural evaluations in the ALCEA method. To allow for cost comparisons of alternatives, the cost of performing a life-cycle phase is determined on the basis of its performance attributes (PAs). To better structure the analysis and to ensure that all costs are included and that no costs are counted more than once, we consider the following three types of PAs:

1) *initial investment* associated with creating or migrating to the new architecture;
2) the consumption of *operating resources* through the life-cycle phase;
3) *revenue* resulting from product sales or other sources such as sales of services and maintenance contracts.

For each of the PAs, an evaluation function is defined to calculate their value. Evaluation functions consist of arithmetic operations applied to one or more basic factors. Basic factors are the core ingredients of value calculations, e.g., the hourly cost of engineering.

To learn the life cycle and how it is affected by architectural decisions, it is important to involve the process owners of the different life-cycle phases. It is also important to set the scope for the system under consideration and assign costs that relate to the organization that is responsible for the life cycle. Our analysis focuses primarily on deciding, within the producing organization, which architecture will be used in the future. This means that all the effects of an architectural decision should be translated into a cost or revenue for the producer.

The actual evaluation of an architecture, based on the principles mentioned previously, is a question of identifying what *effects* (in terms of PAs) will be *caused* by using a system based on that architecture in all life-cycle phases. The causes should thus capture the mechanisms or parts of the architecture that contribute to the effects. Also, since architecture captures only certain properties of the system, and many others are the result of detailed design decisions, comparisons of different architectures must assume that the same decisions are made when creating the concrete system, regardless of which abstract architecture is the basis. In other words, the principle *ceteris paribus* ("all else equal") must be used for aspects that are not governed by the architecture.

B. USING ALCEA

Now, we describe the process of using ALCEA, how to evaluate the results, and how to deal with uncertainties in the analysis.

1) PROCESS OF USING ALCEA

We define a three-stage process to put the ALCEA method to practical use.

1) *Stage 1. Instantiation:* The instantiation stage includes the identification of relevant life-cycle phases and PAs. The purpose of this initial stage is to define a high-level description of the life-cycle process and its phases. For the system under analysis, the relevant life-cycle phases must be identified and understood. This includes identifying PAs and their corresponding evaluation functions. As an example, the annual cost associated with feature development could be described by the expression: (*average number of hours required for development of a feature*) × (*number of features to be developed per year*) × (*per-hour cost for engineering*).
   The PA captured by this evaluation function is of the type operating resources with three basic factors. Note that some basic factors, for example, the per hour cost for engineering, will most likely recur in several evaluation functions. The evaluation model and its evaluation functions are captured in an overall evaluation form (described in Section II-B2), and the basic factors are collected in a summary table of the basic factors (described in Section II-B3). In addition, there may be several PAs that are used to evaluate a single life-cycle phase. At this stage, variants of the input that can result in significantly different actions in the life-cycle phase must be adequately captured by sufficiently expressive evaluation functions.
2) *Stage 2. Preparation:* The preparation stage includes the identification of architectural alternatives and the collection of data. The ALCEA method requires that architectural alternatives be identified and defined. First, the *baseline* must be identified (often today's architecture). Then, descriptions of the architectural *alternatives* are created that will be evaluated. Once the evaluation model is created and validated by the process owners involved in the system life cycle, data are collected to quantify the *basic factors* that allow the calculation of the evaluation functions. For the baseline architecture, we assign values based on historical data, when such data are available. The relevant process owners then provide the assessment of the effect that the new architectural alternatives will have in terms of basic factors. It is assumed that the architectural alternatives do not affect the evaluation functions themselves, but only the values of the basic factors. In addition, the rationale for the assessment is documented by describing the cause-and-effect relationship from architectural features to life-cycle phase PAs.
3) *Stage 3. Comparison:* The comparison stage includes the comparison of alternatives, including their validation and sensitivity analysis. When completed, the assessment is summarized and a *business case* is developed for each alternative architecture (see Section II-C1). Note that an alternative architecture often affects aspects such as sales volume or pricing, which, in turn, affect the expected revenue for that alternative. Therefore, revenue must be considered in addition to costs.

To produce reliable comparisons, the baseline must be validated. This can be done by comparing the current budget for the life-cycle phases with the outcomes of the evaluation functions. Furthermore, a *sensitivity analysis* is performed to identify important uncertainties in the cost estimations and the resulting risks in the overall comparison. Using sensitivity analysis, it can be analyzed which of the basic factors are the most important and how much the business case is affected by outcomes that differ from estimates (see Section II-C3).

The instantiation stage (1) of the ALCEA method can be performed once, and then, reused in multiple analyses of the same system. Data collected in the preparation stage (2) may need to be updated to reflect events that have occurred since the last analysis, whereas the values for the baseline can typically be reused in multiple comparisons.

Data collection can be performed in a single well-prepared workshop in which all process owners participate. Alternatively, the analysis team could gather the relevant process owners of each life-cycle process in separate, shorter meetings to reduce the difficulties of finding a calendar slot that suits everyone. If such a distributed evaluation is chosen, it is essential that at least one person, e.g., an architect, is present

TABLE 1. Header of Form for Collecting Input to an ALCEA Analysis

Life-cycle phase	Performance attribute (PA)		Context	Effect		Rating				Comments	
	PA type	PA	Scenario	Effect on PA	Parts of arch. causing effect	Evaluation function	Baseline	Alternative	Difference	Improving actions	Analysis assumptions

in all sessions, to ensure that evaluations are done based on the same principles.

2) EVALUATION FORM

For practical evaluations, the information model in Fig. 1 has been transformed into a tabular format where each analysis item is on one line, and the other entities are in columns. The form is shown in Table 1.[1] The form is filled out during the process of using the method. During stage 1, the entities *life-cycle phase* and *PA* (and its type) are identified. For each PA, an *evaluation function* is defined, which takes *basic factors* as input. During stage 2, the basic factors are given concrete values, thus capturing the effects the architecture has on a PA through the evaluation functions. Thus, this evaluation model postulates that the performance of a process, and thereby, also the architecture, is only evaluated by measurable factors that are used to calculate estimates . This approach allows analyses to be based on real data, and by providing a predefined form (similar to that used in failure mode and effects analysis [3], which is well known to many practitioners), quickly gain momentum in their analyses. The form is structured in the following columns.

1) *Life-cycle phase:* The name of the life-cycle phase that is analyzed in this row.
2) *PA*, with the following subcategories.
   a) *Performance attribute (PA):* A description of the concrete PA.
   b) *PA type:* Investment, operating resource, or revenue.
3) *Context:* A label denoting the scenario(s) for which the input holds, e.g., "All," or "Variant 3."
4) *Effect*, with the following subcategories.
   a) *Effect on PA:* A textual description of the effect of the architectural alternative on the considered PA.
   b) *Parts of architecture causing effect:* A short reference to the parts of the architecture that are causing the effect mentioned in the previous column.
5) *Rating*, with the following subcategories.
   a) *Evaluation function:* A textual description of the evaluation function used to calculate the rating for different architectural alternatives of this PA, as a function of a set of basic factors.
   b) *Baseline:* The result of applying the evaluation function given the basic factor values of the baseline solution.
   c) *Alternative:* The result of applying the evaluation function given the basic factor values of the alternative.
   d) *Difference:* The difference between the rating for the baseline and the rating for the alternative.
6) *Comments*, with the following subcategories.
   a) *Improving actions:* If the evaluation team identifies things in the architecture that could be solved in other ways that would improve the assessment, those findings can be documented in this column.
   b) *Analysis assumptions:* The final column can be used to note assumptions made during the analysis, including, e.g., assumptions about what decisions will be made during the subsequent detailed design. Making these assumptions explicit makes it easier to ensure their consistent application for all alternatives (see the discussion of *ceteris paribus* in Section II-A).

TABLE 2. Header of the Form Summarizing Basic Factors to Allow Reuse in Multiple Evaluation Functions

Basic factor	Unit	Baseline value	Alternative value (min)	Alternative value (max)	Comment

3) BASIC FACTORS AND NUMERICAL ASSESSMENTS

The evaluation functions indicate the economic importance expressed in currency units per year, taking a positive value for income and a negative value for costs for the system producer. Its parameters are a set of basic factors that should be provided with values for the baseline architecture and for the proposed alternative architecture. Typically, the same basic factor serves as input to the evaluation of several analysis items, and some factors are even used in several or all processes. Therefore, the basic factors are summarized in a separate form with the structure shown in Table 2.[2] A completed version of this table is shown later in the example in Section III-C1.

Going from left to right, the following columns are used.

1) *Basic factor:* A textual description of the factor.
2) *Unit:* Unit of measurement used.

[1]The interested reader is referred to the supplementary materials, where the form is provided as a spreadsheet.

[2]The interested reader is referred to the supplementary materials, where the form is provided as a fillable spreadsheet.

3) *Baseline value:* The value of the factor in the baseline.
4) *Alternative value:* The value of the factor in the alternative solution. Here, we add two columns to provide minimum and maximum bounds for the estimated costs. These Min–Max intervals are intended to capture uncertainties in the estimates and are later used in the comparison stage.
5) *Comment:* A textual description that captures, e.g., assumptions on which the value is based or other relevant information.

Note that the focus of the evaluation function is on the costs and revenues of the producer organization. Since certain life-cycle phases might be owned by another organization than the producer (e.g., maintenance), values perceived by them need to be translated into costs or revenues for the producer organization. It is important to document the reasoning behind this translation in the analysis sheet.

When determining values on the basic factors for the baseline (if using the current product), useful input includes the department budgets for those departments working with the products, the part price for the parts included in it, etc. The values of the alternative can be estimated by comparing the differences in architecture between the two solutions.

C. USING THE DATA FOR ANALYSIS

We now discuss how to use the data resulting from applying the method for analysis of alternative architectures.

1) SUMMARIZING THE OUTCOME OF LIFE-CYCLE PHASES

After finalizing the evaluation by filling out the form line by line, it is time to step back and look at the overall picture. Often, the evaluation is used as a basis to decide whether or not to invest in the development of a new or changed architecture. This business case can be created simply by summing the overall ratings for each line. However, usually, it is beneficial to dig a bit deeper into the data created, since this improves the understanding of why a certain effect was achieved, which can be used for either further improvements or a broader evaluation taking non-economic factors into account. When the effects of different PAs are examined, the following observations can be made.

1) *Investment* indicates how much initial investment is needed to migrate to the new architecture.
2) *Operating resources* shows the effects on the yearly cost.
3) *Revenue* shows the effects on revenue from sales or other sources, such as maintenance contracts.

The difference between the baseline and the alternative can be used to filter out the most important items in the analysis, and depending on the sign (positive or negative), they summarize the strength and weaknesses of the proposal. By grouping the most important items by the life-cycle phase, it is also possible to show where in the life cycle the gains and losses can be found from the new architecture. Similarly, one could group by the parts of the architecture that cause the effects to see what the key parts are.

2) BUSINESS CASE AND REVENUE

Based on the defined analysis, we now have a projection of the effects, in terms of costs, on the complete life cycle when comparing the baseline to the alternative architecture. To choose between alternative architectures, their business cases need to be constructed and compared. To create a business case, it is necessary to consider the financial benefit of a product, which includes not only changes in costs but also changes in revenue.

a) Revenue: In its simplest form, revenue is given by the expression (*the number of systems sold*) $\times$ (*the price per system*). An organization can generate additional revenue from related services, such as maintenance contracts.

In the initial cost analysis, we compare the baseline with a system based on a new architecture, while assuming that the production volume is unchanged. However, an architectural change can affect sales and, therefore, production volume. During the cost analysis, a change in production volume would be reflected only as an increase in cost and would not necessarily support a decision toward change unless we take the revenue into account.

By summing up all revenues expressed as performance attributes in all phases of the life cycle, we can observe the effect of the new architecture in terms of revenues in addition to the cost analysis. Here, we get the overall projection on how revenue will change for the new system.

b) Business case: A business case must cover the entire life-cycle process of a product to avoid premature conclusions based on the effects on individual units. In fact, problems can arise when the business case of an organizational unit responsible for one life-cycle phase does not reflect the costs or savings of parts of an organizational unit responsible for the other life-cycle phases. For example, a cost change in maintenance due to an architectural change will not necessarily be visible in the business case of the organizational unit responsible for the development.

Moreover, the life cycle of a product might even span multiple organizations. For example, a bus operator organization is usually responsible for the maintenance life-cycle phase of their buses. In such cases, from a producer perspective, maintenance can be considered as a cost, e.g., due to warranty guarantees, but also as a source of revenue, e.g., through sales of extended maintenance contracts.

To build a complete business case that spans all life-cycle phases across multiple units within an organization or even multiple organizations, it is necessary to describe where the organizational boundaries are and assign life-cycle phases to the corresponding organizations and their units. This allows us to estimate revenue streams across internal organizational borders and build a business case that takes into account the complete life cycle of the system.

3) DEALING WITH UNCERTAINTIES

The ALCEA method aims to perform an analysis based as much as possible on facts. However, there is still a large element of subjective judgment involved in deciding the values of the basic factors and the design of the evaluation functions. To strengthen the analysis, we propose three techniques for dealing with these uncertainties: baseline validation, min–max intervals, and sensitivity analysis.

a) Baseline validation: To validate that all costs are accounted for by the basic factors and evaluation functions, it is necessary to assess them on the baseline budget. The evaluation functions should produce a result close to the true cost of the baseline architecture according to the current budget. Once the baseline is validated, we can consider uncertainties in the values of the basic factors themselves.

b) Min–Max intervals: A straightforward way to deal with uncertainties in the values of the basic factors is to ask the process owners to provide intervals within which the estimate of the basic factor is likely to fall, providing both minimum and maximum values for the estimations. This allows the analysis to account for variability in the estimations and to derive best- and worst-case scenarios for the alternative and the baseline architecture.

c) Sensitivity analysis: Another way to manage uncertainties in estimates is to perform a sensitivity analysis on the ratings to see which estimates could potentially distort the main conclusions. As discussed by Axelsson [4], this is an important technique to deal with the inevitable uncertainties involved in architecting. The decision of whether to base future products on a certain architecture alternative is made from the business case, which should then be better than the other alternatives; that is, the sum of the overall ratings should be higher.

The procedure for the sensitivity analysis is to select a basic factor and then to find the value of that factor that causes the total difference between the baseline and the alternative to be zero. This will be the point beyond which the conclusion of which alternative is the best will change. Since we have not put any particular restrictions on the form of the evaluation functions, the equation has to be solved numerically. However, in our experience, the functions tend to take a fairly simple form and be solvable very rapidly even with the simplest numerical algorithms.

The aforementioned procedure should be repeated for all factors that contain any uncertainty, both for the baseline and the alternative solution. When the breakpoint value has been found, it can be compared with the original value to assess the sensitivity of the conclusion. At this point, the architects' judgment has to be used to determine whether the difference is small enough to constitute a major risk or not, since different factors have different levels of inherent uncertainty. If it turns out to be a risk, several mitigation approaches can be used. One is to try to gain more information by a deeper investigation to see if the uncertainty is reducible. Another is to try to modify the architecture to make it less sensitive to unknown events.

III. ALCEA APPLICATION EXAMPLE

To illustrate the usage of the ALCEA method and to provide additional guidance to practitioners who want to use the method, we present an example of architecture analysis.[3] The example is based on reality, but has been simplified and adapted to allow for a short presentation and to avoid revealing proprietary data from the company involved. The example concerns the issue of revising the hardware and software structure of a complex distributed embedded system. To simplify the presentation, we describe the product on which the example is based, in Section III-A.

A. EXAMPLE SYSTEM

The example system is a distributed control system for high-power electric equipment, such as motors or auxiliary power, similar to what is used in power systems, process automation, heavy-duty automotive powertrains, or drive systems for railway trains. The system is a platform used for different product variants in a product line, and the product line architecture (rather than an individual product architecture) is subject to change. In particular, we consider, and in detail describe, the application of ALCEA on a proposed architectural change related to the restructuring of the hardware and software.

As described in Section II, the ALCEA method is performed in the following three stages: The *instantiation stage*, where the life-cycle is mapped and the evaluation model is created; the *preparation stage*, where the architectural alternatives are identified and data are collected; and the *comparison stage*, where the actual analysis is performed. In the remainder of this section, we provide an in-depth description of the application of ALCEA for analyzing an example proposed architecture alternative. Note here that the first two stages are done once per system (and life cycle), and thus, can be reused for other proposed architectural changes.

B. STAGE 1: INSTANTIATION

The instantiation stage of the ALCEA method starts with the mapping of the life cycle of the system to be analyzed. In our example, the life-cycle consists of the following six phases.

1) *Platform development:* Development of the base platform from which concrete products are derived. The platform is revised iteratively, with a major release once per year.
2) *Sales:* Negotiation with potential customers about application requirements and agreement on the terms of sale. This phase occurs in parallel to platform development and provides application requirements as input to application development.
3) *Application development:* Development of the system solution for a particular application or a particular customer segment. Application development is based on the platform, but with a certain set of configurations and additional application-specific software and I/O.

[3]The interested reader is referred to the supplementary materials, where we include complete filled-out spreadsheets for this and an additional example.

TABLE 3. PAs for Each Life-Cycle Phase

Life-cycle phase	Performance attribute (PA)	
	PA	PA type
Platform development	Running changes	Operating resources
	Initial development	Investment
	Development tools	Investment
Sales	System sales	Revenue
Application development	New product development	Operating resources
	Development tools	Investment
Production	Probability of faults	Operating resources
	Cost of parts	Operating resources
	Production staff time	Operating resources
	Production equipment	Investment
Operation	Reliability	Operating resources
	Operation staff time	Operating resources
Maintenance	Spare parts	Operating resources
	Maintenance staff time	Operating resources
	Tools	Investment

4) *Production:* Production of hardware units and assembly into the overall product.
5) *Operation:* Starting up and supporting the use of the product by the customer.
6) *Maintenance:* Upon product failure during operation, performing fault tracing and repair to put the product back in operation.

Note here that the objective of the ALCEA analysis is to evaluate the effect of the architectural alternative for the producer of the system, not the customer or other process owners. Therefore, the process phases are described and modeled using only the details that are relevant to the producer, and the evaluation shows the results in terms of the producer processes. In other examples, there may be additional life-cycle phases such as a disposal phase in cases where the system remains property of the producer throughout operation.

The instantiation stage involves describing these life-cycle phases in more detail. Specifically, for each phase, process variants, PAs, and evaluation functions should be derived. In Table 3, PAs for each phase are listed. For some process phases, there are variants in how the phase is run, typically depending on the different models of the product. In the example, there are three variants of the system that make up three different system assemblies. Variant 1 is a small system with one motor and one auxiliary power system; Variant 2 is a medium-size system with four copies of the system in Variant 1, organized in two groups; and Variant 3 is a large system with three copies of Variant 2. The variants correspond to 30%, 60%, and 10% of the sales volume, respectively.

To describe the mechanics of the instantiation, let us look at the first life-cycle phase: platform development. Investigating the procedures that the producer employs, we may find the key PAs that reveal the costs in this phase. In the example, we see that bug fixing and support is needed from a platform engineering team ("running changes"). Furthermore, architectural change requires investments in the form of development and the purchase of new tools ("initial development" and "development tools"). In this way, we simplify and model each phase with key PAs.

After defining the PAs, the evaluation functions need to be defined. Table 4 shows a few such evaluation functions, exemplifying each PA type, and Table 5 shows the basic factors in these evaluation functions. Let us again elaborate on how this is done in practice and look at the first evaluation function. From the experts in the producer organization, we get an estimate on how much work is needed to accomplish the proposal, engineering hourly cost, and the expected lifetime of the platform. From this estimate, we derive a formula to express the investment. In this way, we model the evaluation function for each PA.

C. STAGE 2: PREPARATION

The second stage begins with the identification of a baseline architecture. In our example, the functionality of the systems built on the platform controls the electrical equipment, e.g., ensuring that the motor speed remains at the level currently given by a set point from the operator. For an auxiliary power system, the set point is fixed to a certain voltage, which has to be upheld regardless of the current being drawn from the system. The control functions use sensors to measure voltage, current, and temperature; and its principal actuators are electrical converters that control the shape of the voltage curves. In addition, secondary functionality related to diagnostics, supervision, and operator interfaces is supported. To address the variability of configurations of the control system, a product line approach is used in which the architecture consists of several building blocks that can be connected as required.

The main layout of the hardware architecture is illustrated in Fig. 2, in black and red. In the architecture, there are three types of electronic control units (ECUs), which are connected over various networks.

1) *Converter control unit (CCU):* An ASIC-based ECU, responsible for the detailed, micro-second level control of the converters.
2) *General control unit (GCU):* A general ECU controlling a set of CCUs; typically placed close to each other and solving parts of a common overall task. The GCU executes different software depending on its application, e.g., controlling motors or auxiliary units. It also has numerous I/O capabilities to handle customer or application-specific requirements. One of the GCUs also runs software to coordinate the set of GCUs.
3) *Operator control unit (OCU):* A control unit where operator commands are collected and where system status information is displayed.

The CCU and GCU software are developed in-house, whereas the OCU is developed and produced by a supplier. Internally, the GCU and OCU are realized using basic software (BSW, i.e., operating system, communication, etc.) and application software (ASW). The BSW is different between the OCU and the GCU, but is the same in all GCUs. The ASW also differs between different GCUs, depending on the application they are used for.

TABLE 4. Evaluation Functions for Selected PAs; Referring to Basic Factors in Table 5

Life-cycle phase	Performance attribute (PA)		Evaluation function
	PA	PA type	
Platform development	Initial development	Investment	A * B * C / D
Sales	System sales	Revenue	E * F
Application development	New product development	Operating resources	G * B * C
Production	Cost of parts variant 'a', a= 1,2,3	Operating resources	H_a * F * I_a
Operation	Reliability	Operating resources	J * K * L

TABLE 5. Basic Factor Values for the PAs for the Proposal in Table 4

Basic factor		Unit	Baseline value	Alternative value
A	Initial development effort new architecture	person-years	0	**40***
B	Annual working hours	h/year	1600	1600
C	White-collar average cost per hour	$/h	120	120
D	Investment payoff time	Years	5	5
E	Average sales price	$	11 232	**12 168***
F	Number of units produced	1/year	5000	**5500***
G	Annual application development effort	person-years	20	**19.5***
H_1	Variant 1 Product cost	$	1600	**1100***
H_2	Variant 2 Product cost	$	6400	**3400***
H_3	Variant 3 Product cost	$	19 200	**10 200***
I_1	Variant 1 Part of total volume	%	30	30
I_2	Variant 2 Part of total volume	%	60	60
I_3	Variant 3 Part of total volume	%	10	10
J	Number of system repair actions	1/year	3900	**4537.5***
K	Standstill cost per hour	$/h	2000	2000
L	Average standstill time during repair	h	3	3

*alternative value differing from the baseline value.

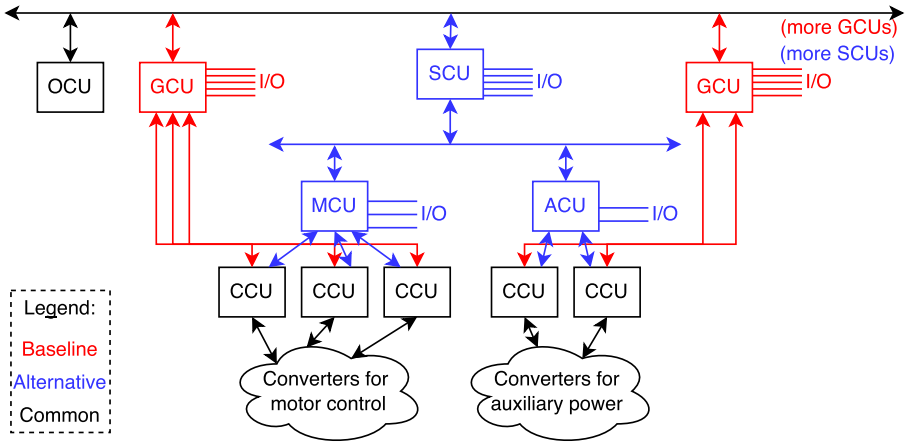

FIGURE 2. Baseline hardware architecture used in the example (black+red) and the proposed new architecture (black+blue).

1) PROPOSED ARCHITECTURE CHANGE—RESTRUCTURING OF HARDWARE AND SOFTWARE ARCHITECTURE

In our proposed change, the hardware and software architecture will be restructured. The background is that the solution with general GCUs that run different kinds of software depending on use has proven to be quite expensive since the hardware has to be powerful enough to handle all possible situations. However, in most concrete applications, it turns out that only a fraction of the processing power, memory, and I/O is used, which is suspected to drive the component cost. Therefore, the idea is to replace the GCUs with a hierarchical structure, where a specific hardware unit is developed and optimized for each application type, i.e., a motor control unit (MCU), an auxiliary control unit (ACU), etc. Moreover, a new subsystem control unit (SCU) is introduced for coordination, a task that was previously allocated to one of the GCUs. In this way, much of the customer-specific adaptations can be allocated to the SCU. The functionality of the system is unchanged, but some hardware units are new and the allocation of software is changed. The proposed hardware architecture change is shown in Fig. 2; common elements are shown in black, current elements (to be replaced) in red, and new elements to be added in blue.

Having defined the new proposed alternative architecture, it is now time to populate the basic factors table for the proposal with numerical values, which will give us the PA values for the evaluation. As mentioned in Section II-B3, the baseline basic factor values are fetched from existing company data. The

TABLE 6. Baseline and Alternative Architecture Values for Initial Development of the Proposal

PA	Baseline value	Alternative value
Initial development	0	-7.5 M$

TABLE 7. Baseline and Alternative Architecture Values for New Product Application Development for the Proposal

PA	Baseline value	Alternative value
New product development	-3.8 M$	-3.7 M$

corresponding values for the proposed alternative architecture are estimated by studying the architectural differences between the two solutions. The baseline and alternative values of the basic factors presented in Section III-B are shown in Table 5. All factors from Table 4 are included for completeness; however, many factors are unaffected by the choice of architecture. For factors where the alternative architecture influences it, the alternative value is highlighted in boldface characters. The rationale behind these alternative values is further explained when we perform the comparisons throughout Section III-D. To preserve readability of the tables, we present the examples with a single alternative value and not a min and max value.

D. STAGE 3: COMPARISON

In the third stage, the actual evaluation of the alternative architecture is done to understand what financial effect the alternative architecture will have in each product life-cycle phase. Hence, the analysis starts by inserting the values of all the applicable basic factors—see the subset of basic factors in Section III-C1—into the PA evaluation functions for deriving the PA values.

We now go through and calculate some of the PAs in the evaluation table to exemplify the analysis. We round the values of the PAs to one decimal point for the brevity of the presentation. To avoid tedious repetition, we do not explicitly reference each of the tables summarizing the contributions of the PAs.

1) PLATFORM DEVELOPMENT: INITIAL DEVELOPMENT

In the platform development life-cycle phase, one of the PAs is initial development. The new ECUs need to be developed and the cost is simply the manpower needed, which is estimated at 40 person-years, or a total cost of -7.5 M$. With an economic lifetime of 5 years, the annual figure is -1.5 M$ for the alternative solution (and 0 for the baseline) shown in Table 6.

2) APPLICATION DEVELOPMENT: NEW PRODUCT DEVELOPMENT

The annual application development effort is (*average manpower cost per project*) × (*number of application projects per year*). For the baseline, the average project requires 2 person-years to complete, and there are ten projects per year. Due to the new structure of the architecture where application changes are separated in the SCU, the effort per project is assumed to be reduced to 1.5 person-years. On the other hand, it is assumed that the number of projects will increase by 30% as flexibility will allow the company to be competitive for more niche projects. Therefore, the total cost for application development only changes from -3.8 M$ to -3.7 M$ shown in Table 7.

TABLE 8. Baseline and Alternative Architecture Values for the Cost of Parts in Variants 1, 2, and 3 of the Proposal

PA	Baseline value	Alternative value
Cost of parts Variant 1	-2.4 M$	-1.8 M$
Cost of parts Variant 2	-19.2 M$	-11.2 M$
Cost of parts Variant 3	-9.6 M$	-5.6 M$
Cost of parts Aggregated	-31.2 M$	-18.6 M$

TABLE 9. Baseline and Alternative Architecture Values for System Sales of the Proposal

PA	Baseline value	Alternative value
System sales	56.2 M$	66.9 M$

TABLE 10. Baseline and Alternative Architecture Values for the Reliability of the Proposal

PA	Baseline value	Alternative value
Reliability	-23.4 M$	-27.2 M$

3) PRODUCTION: COST OF PARTS

The cost of the parts is simply (*cost of part for a variant*) × (*annual production of the variant*). The analysis is divided into Variant 1, 2, and 3, as described in Section III-B. Variant 1 consists of two GCUs; Variant 2 of eight GCUs; and Variant 3 of 24 GCUs in the baseline. For the alternative, Variant 1 consists of one SCU, one MCU, and one1 ACU; Variant 2 of two SCUs, four MCUs, and four ACUs; and Variant 3 of six SCUs, 12 MCUs, and 12 ACUs. The baseline thus contains fewer components in all variants, but on the other hand, the GCU is more expensive than the other, more specialized, components. Therefore, the average cost (weighted by the variant production mix) is reduced from 6240 $ to 3300 $. On the other hand, the production volume increases from 5000 systems to 5500 systems per year. Therefore, the total annual cost of production parts is reduced from -31.2 M$ to -18.6 M$, shown in Table 8.

4) SALES: SYSTEM SALES

Revenue from product sales is calculated by adding sales prices (weighted by the variants production mix.) The production volume increases from 5000 systems to 5500 systems and the price change yields an increase in revenue from 56.2 M$ to 66.9 M$ per year, shown in Table 9.

5) OPERATION: RELIABILITY

The reliability of the system is linked to a financial loss that comes from the system being unable to operate due to fault occurrence and the corresponding obligations of the producer to perform repairs under the warranty period. The increased

TABLE 11. Summary Per Life-Cycle Phase of the Proposal

Life-cycle phase	Baseline	Alternative	Difference	Contrib.
Platform dev.	-2.7 M$	-4.2 M$	-1.5 M$	-8 %
Sales	56.2 M$	66.9 M$	10.7 M$	55 %
Appl. dev.	-3.8 M$	-3.7 M$	0.1 M$	0 %
Production	-31.5 M$	-18.9 M$	12.6 M$	64 %
Operation	-23.4 M$	-27.2 M$	-3.8 M$	-19 %
Maintenance	-3.2 M$	-1.7 M$	1.5 M$	8 %
Total life-cycle	-8.5 M$	11.1 M$	19.6 M$	100 %

TABLE 12. Summary Per PA Type in the Proposal

PA type	Baseline	Alternative	Difference	Contrib.
Revenue	56.2 M$	66.9 M$	10.8 M$	55 %
Operating res.	-64.7 M$	-54.3 M$	10.4 M$	53 %
Investment	0 $	-1.53 M$	-1.53 M$	-8 %
Total	-8.5 M$	11.1 M$	19.6 M$	100 %

number of components is expected to result in a change from 3900 to 4537.5 annual fault repair actions which yields a cost increase from -23.4 M$ to -27.2 M$ per year. Note that in this example, we do not consider the addition of diagnostics to identify facilitate repairs, which may increase the probability of failure. ALCEA analysis wishing to incorporate these effects may identify them as additional PAs, shown in Table 10.

6) SUMMARY OF EVALUATION

The five aforementioned PAs show the principles of how the analysis is performed. Table 11 summarizes the annual profits and losses in each life-cycle phase and in the total life cycle, after completion of all analysis stages. The table shows the baseline architecture and alternative architecture values, and the difference between them. The table also shows which relative percentage contribution each life-cycle phase has on the total annual business improvement.

As indicated in Section II-C1 interesting information about the evaluation data can also be found by summarizing the data in other ways, e.g., categorized per PA type. This is illustrated in Table 12. By observing Tables 11 and 12 together, we can deduce that the change in operating resources, supposedly primarily in the production and maintenance phases, is the main contributor to the annual business improvement. This supposition should be confirmed by a look at the full PA table.

We are now ready to make observations about which financial impact the alternative architecture will have on the business and where the big contributors can be found. In this case, the result is that the new architecture would, per year, improve the business from a loss of -8.5 M$ to a profit of 11.1 M$. The main contributors to this 19.6 M$ improvement are increased sales (10.7 M$) and reduced production cost (12.6 M$).

Looking at the analysis and the business case, we can analyze uncertainties, using data validation for the baseline architecture, basic factor min–max interval definition, and sensitivity analysis.

The first step in dealing with uncertainties is to validate the calculated baseline data by comparing it against actual budget data. In a real case, the sum of the costs should roughly equal the budget for the system. The next step is to find realistic min–max intervals for each of the basic factors. Finally, we do the sensitivity analysis, deciding for each basic factor what the break-point value is, i.e., the value that yields a zero difference in profit between the baseline architecture and the alternative architecture. We find the sensitive basic factors in the analysis by identifying the ones whose break-point values are located within the min–max interval, i.e., the result is sensitive to that factor and the result can realistically end up in that range.

In this example, we found that the conclusion of the analysis is quite robust to changes of the values of basic factors within their min–max intervals. The basic factor to which the analysis is most sensitive is the cost of the GCU, and if that would shrink from the current 800 $ to around 450 $, the current architecture would be competitive to the alternative. However, the price of the GCU is a known fact, not an estimate, so it appears unlikely that such a cost reduction would occur (without at the same time reducing the other ECUs in the alternative with similar amounts). Another sensitive factor is the average repair time. If that would go up from 3 to 5.2 h for the alternative (but remain at 3 h for the baseline), the conclusion would change and the proposed architecture is no longer favorable. Such a scenario is possible, for instance, due to the increased number of spare part types that must be kept or more complex fault tracing, and the likelihood of this happening should be investigated before embarking on the new architecture development.

IV. DISCUSSION

The aforementioned example analysis of a proposed architectural change above highlights several benefits of the ALCEA method. First, it shows how the same analysis structure can be reused for quite different purposes, since the instantiation stage of the method as done for the shown proposal can be reused for other analyses, too, thereby reducing the effort for other analyses since both structure and some data could be reused.

Second, the sensitivity analysis is important, both as a validation of the results, but also as a constructive pointer to what actions can be taken to improve the situation, the latter point hints at a different way of using the model. Once it has been instantiated for a given system in a company, it can be used to guide the process of identifying refactoring by pointing out areas of improvement that would have a large effect on the business. That is, the method is applied before an alternative architecture has been sketched and serves as input to the process of deriving the alternative.

A. INDICATORS FOR USEFULNESS

In Section I, we presented four desirable properties that the evaluation model should have; it should be lean, fact-based, general, and transparent. We will now discuss how the ALCEA method fulfills those properties.

1) LEAN

The model reduces time consumption in several ways. First, many parts of the analysis can be reused many times. The whole instantiation can be done once for a certain type of system at a certain company, and then, used for analyzing many different alternatives, or for iterative solutions. Second, it is not necessary to have all process owners participate throughout the analysis, but only those with knowledge about a certain process need to participate in the analysis of that process. We believe that thereby the cost of performing an ALCEA analysis is minimized.

2) FACT BASED

All of the performance indicators are directly related to measurable things that companies often already measure, including resource consumption for different processes, frequency of events, etc.

3) GENERAL

The model described is completely general and can be used to evaluate any architecture in any organization. Company-specific details are provided during the instantiation and preparation phases.

4) TRANSPARENT

The cause-effect chains for different ratings are provided in the evaluation form, as is an importance rating for different analysis items in terms of their relative contributions to the overall result.

B. EXPERIENCES FROM REAL USE

The previous parts of the section have shown an example of applying ALCEA. The example is partial, and it might be hard to deduce how much effort goes into this kind of analysis. Our own experience from doing the real-life analyses[4] that underlie the example shows that the effort is quite limited. Assessing each analysis item (row in the matrix) typically required about 10–20 min by a few knowledgeable people. With about ten life-cycle processes and about five analysis items per process, there will be about 50 items to consider, i.e., a total duration of about 8–16 hs. With five people involved, this would mean a total analysis cost of 5–10 working days for a first analysis.

Subsequent analyses of other alternatives for the same system would be much faster. To this comes the one-time cost of instantiating the model, which is a similar effort. In addition, the effort to get the relevant data is needed, and this varies depending on how accessible the data are within the particular organization. Our experience from similar analyses using other methods that are based on abstract QAs is that there tend to be lengthy discussions on the interpretation of attributes and on personal opinions that are not based on data. Such discussions are more or less absent when applying ALCEA, and therefore, the whole process becomes less resource-intensive and more predictable.

[4]These real analyses could not be included in this article because of intellectual properties.

C. POTENTIAL APPLICATION CONTEXTS OF ALCEA

The ALCEA method is designed to perform lean, fact-based, and transparent analyses of architectural alternatives during the development and maintenance of software-intensive systems. Although not required, it is likely easier to apply the method in settings with a preexisting architecture, since then more values of basic factors can be estimated with greater confidence. When used to analyze possible decisions in an agile setting, ALCEA can benefit from reuse of identified PAs and evaluation functions. For a completely new system, the required inputs may necessarily need to be more estimated, and hence, the min–max intervals on them will be greater. It is then important to perform a careful sensitivity analysis to ensure that the analysis is still fact based. Given the domain-specific expertise needed to define the PAs and the estimations of basic factors, the method is probably more useful for companies that have previously developed complex systems.

To aid the application of the method in greenfield settings, we intend to build a corpus of common PAs and basic factors. The corpus is intended to be a living document, added to by the community. A starting point of such a corpus is provided in the supplementary material and is based on the attributes and factors identified in the real-world examples presented in Section III. The preexistence of this information can help the adoption of the method in practice, although still many of these factors are of course specific to the setting in which the method is applied, and therefore, need to be identified on a case-by-case basis.

V. RELATED WORK

Before the concept of architectural analysis had matured, Abowd et al. [5] published a report on recommended best practices based on two workshops in 1995 and 1996, respectively. They discussed the cost of doing the evaluation, which should be compared to the benefits in terms of increased understanding of the system and the requirements, which in turn can lead to lowered project costs. They identified two main categories of analysis processes, which were distinguished on whether they focused on qualitative questions, e.g., discussing scenarios in various contexts, or if they were based on quantitative measurements. They also gave recommendations on when to do the analysis, by whom it should be done, how to select a good process, the expected output, and more. They mentioned software architecture analysis method (SAAM) [6] (discussed in the following) briefly, but the article is otherwise method agnostic.

There now exists a wide selection of published methods for architectural analysis [6], [7], [8], [9], [10], [11], most of them based on analyzing scenarios. Their focus varies, and can either consider QAs in general [9], [11], or take a more narrow approach, focusing on just cost/benefit [10], [12], modifiability [6], [13], or maintenance effort [8]. Some of these methods are described briefly in the following. Despite this variety of

methods, to the best of our knowledge, there does not yet exist an analysis method that explicitly considers the system's life-cycle phases.

A. SAAM, ARCHITECTURE TRADEOFF ANALYSIS METHOD (ATAM), AND RELATED METHODS

One of the earliest surveys on software architecture analysis methods was done in 2002 by Dobrica and Niemelä [14]. Their goal was to help software architects and designers select a suitable analysis method, based on whether the methods were intended to be used early or late in the design process, on few or many QAs, using an easy or complex process. Assuming that early in the process is better than late, many attributes are better than few, and an easy process is better than a complex one, they recommended the ATAM [11] from the Software Engineering Institute at the Carnegie Mellon University (CMU/SEI).

When using the ATAM [11], the desired functionality and QAs of the target system are first elicited and discussed, and then, related to the suggested architectural elements (also called architectural strategies, AS). If a particular element is required to fulfill a QA requirement, it is called a *sensitivity point*. If it also affects one or more other QAs, particularly in a different direction, it is called a *tradeoff point*. By letting the stakeholders prioritize the QAs involved in tradeoff points, any architectural elements that may need to be changed are easily found [15]. The process is iterated as needed. The QAs can be selected freely, with some typical and reasonable starting points being either ISO/IEC 25010 [16] or the list presented by Bass et al. [17].

Also included in the survey by Dobrica and Niemelä [14] is the predecessor to the ATAM, the SAAM) [6]. The SAAM does not consider tradeoff points and primarily focuses on *modifiability* as compared to the full set of QAs used in the ATAM. In addition to the ATAM, the SAAM was extended at least by: SAAMCS [18], focusing on complex scenarios; ESAAMI [19], adding a reusable knowledge base; and SAAMER [20], adding considerations for evolution and reusability,.

The ATAM was later refined into the cost benefit analysis method (CBAM) [10], [12], which can be used to prioritize possible architectural changes. Just as in the ATAM, the CBAM creates links between possible ASs and selected QA. It then goes further, calculating a combined benefit for each AS, adding the product of the weight of each QA and the utility of the reached QA value as a value between 0 and 100%. By dividing the benefit of the QAs with the cost for the AS, each AS gets a return on investment (ROI) score, which in turn can be used for a preliminary ordering. The strategies with the best ROI scores then get their costs and benefits converted to ranges, based on the estimated risks. These ranges are then compared pairwise, resulting in the final ordering.

Of the analysis methods in the SAAM family tree, CBAM is the one closest to ALCEA, as CBAM also considers the financial aspect in a quantitative way. The main differences are that ALCEA explicitly separates the costs and benefits according to the product life cycle, making the values easier to elicit, and that ALCEA uses monetary units, providing a more direct business case.

The survey by Ionita et al. [21] compared much the same set of methods as Dobrica and Niemelä, in particular the SAAM ones in the family tree. They concluded that most methods lead to "*. . . improved communications between stakeholders and meaningful architectural discoveries and improvements if the assessment is performed early in the development phase*."

Mattson et al. [22] searched for analysis methods explicitly targeted on performance, maintainability, testability, and portability. The method they find most useful is ATAM, just as Dobrica and Niemelä did earlier.

Later surveys have taken different perspectives, e.g., looking at methods specific for product lines [23], evaluation tools [24], or architecture languages [25]. Typically, SAAM and/or ATAM appear in these surveys as well.

B. OTHER METHODS

In addition to the methods in the SAAM tree discussed previously, Bengtsson and Bosch described the scenario-based software architecture reengineering method (SBAR) [7]. Despite its name, the main difference between SBAR and other methods is not the scenario viewpoint, which is quite common. Instead, the difference is that SBAR considers simulations, mathematical models, and experience-based reasoning if those techniques would be more appropriate. The result of an SBAR analysis is a list of whether each examined QA is satisfied, but it is not quantified further.

Next, Bengtsson and Bosch discussed architecture level prediction of system maintenance (ALPSM) [8]. ALPSM uses historical data, estimations, and other factors to predict the sizes of future maintenance efforts. The unit of these sizes is not defined. ALPSM is focused on the maintenance phase of a system, while ALCEA also considers all other relevant phases in the system life cycle.

The software architecture evaluation model (SAEM) [9] uses a quantitative approach to find out which QAs are fulfilled by a particular version of the architecture, and to what degree. For the measurements, a technique such as goal-question-metric [26] can be used. In SAEM, no particular analysis process is suggested.

Babar et al. [27] presented a framework for evaluating software architecture methods, applying it on what effectively amounts to the SAAM family tree plus architecture-level modifiability analysis (ALMA) [13]. ALMA focuses on modifiability initiated by external factors, e.g., changes in the environment or the requirements, thereby ignoring changes such as bug fixes. In particular, the goal of the analysis may be to predict the cost of the elicited modifications, their risk, or the effect of different architectural selections. The method is described as a combination of ALPSMA [8] and SAAMCS [18].

VI. CONCLUSION

In this article, we have presented the ALCEA method and illustrated its application on an example proposed architecture change. The basic idea behind the method is that the value of an architecture lies in how well it supports the creation of well-functioning systems that perform throughout all of their life-cycle phases. Therefore, it is natural to base the analysis on the system life cycle and look at the principal PAs of investments, operating resource consumption, and revenue, instead of looking at abstract QAs that have been the basis for most existing methods. This approach makes it possible to base the analysis on real data, and by providing a predefined form they can quickly gain momentum in their analyses. The method also supports sensitivity analyses for the uncertainties that are always present in the early phases of the system design. Our research focus is primarily on software-intensive embedded systems, and that is the area where the method has been applied so far, as illustrated by the examples mentioned previously. However, we believe the method is applicable in other domains too, such as software-only systems on servers or desktops, or even for mechanical systems with no software in them at all. Verifying this generalization is future work.

REFERENCES

[1] *ISO/IEC/IEEE 42010:2011 Systems and Software Engineering–Architecture description*, Standard ISO/IEC/IEEE 42010:2011, 2011.

[2] J. Fröberg, S. Larsson, S. Dersten, and P.-Å. Nordlander, "Defining a method for identifying architectural candidates as part of engineering a system architecture," in *Proc. Int. Syst. Conf.*, 2014, pp. 266–271.

[3] S.-H. Teng and S.-Y. Ho, "Failure mode and effects analysis: An integrated approach for product design and process control," *Int. J. Qual. Rel. Manage.*, vol. 13, no. 5, pp. 8–26, 1996.

[4] J. Axelsson, "On how to deal with uncertainty when architecting embedded software and systems," in *Proc. Eur. Conf. Softw. Archit.*, 2011, pp. 199–202.

[5] G. Abowd, L. Bass, P. Clements, R. Kazman, L. Northrop, and A. Zaremski, "Recommended best industrial practice for software architecture evaluation," Software Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-96-TR-025, 1997.

[6] R. Kazman, L. Bass, G. Abowd, and M. Webb, "SAAM: A method for analyzing the properties of software architectures," in *Proc. Int. Conf. Softw. Eng.*, 1994, pp. 81–90.

[7] P. Bengtsson and J. Bosch, "Scenario-based software architecture reengineering," in *Proc. Int. Conf. Softw. Reuse*, 1998, pp. 308–317.

[8] P. Bengtsson and J. Bosch, "Architecture level prediction of software maintenance," in *Proc. Eur. Conf. Softw. Maintenance Reengineering*, 1999, pp. 139–147.

[9] J. C. Dueñas, W. L. de Oliveira, and A. Juan, "A software architecture evaluation model," in *Proc. Int. Workshop Architectural Reasoning Embedded Syst.*, 1998, pp. 148–157.

[10] R. Kazman, J. Asundi, and M. Klein, "Quantifying the costs and benefits of architectural decisions," in *Proc. Int. Conf. Softw. Eng.*, 2001, pp. 297–306.

[11] R. Kazman, M. Klein, M. Barbacci, T. Longstaff, H. Lipson, and J. Carriere, "The architecture tradeoff analysis method," in *Proc. Int. Conf. Eng. Complex Comput. Syst.*, 1998, pp. 68–78.

[12] M. Moore, R. Kazman, M. Klein, and J. Asundi, "Quantifying the value of architecture design decisions: Lessons from the field," in *Proc. Int. Conf. Softw. Eng.*, 2003, pp. 557–562.

[13] P. Bengtsson, N. Lassing, J. Bosch, and H. van Vliet, "Architecture-level modifiability analysis (ALMA)," *J. Syst. Softw.*, vol. 69, no. 1/2, pp. 129–147, 2004.

[14] L. Dobrica and E. Niemelá, "A survey on software architecture analysis methods," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 638–653, Jul. 2002.

[15] D. Brahneborg and W. Afzal, "A lightweight architecture analysis of a monolithic messaging gateway," in *Proc. Int. Conf. Softw. Archit.*, 2020, pp. 25–32.

[16] *ISO/IEC 25010*, 2020. Accessed: Jun. 7, 2020. [Online]. Available: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[17] L. Bass, P. Clements, and R. Kazman, *Software Architecture in Practice*, 3rd ed. Reading, MA, USA: Addison-Wesley Professional, 2013.

[18] N. Lassing, D. Rijsenbrij, and H. van Vliet, "On software architecture analysis of flexibility, complexity of changes: Size isn't everything," in *Proc. Nordic Softw. Archit. Workshop*, 1999, pp. 1–6.

[19] G. Molter, "Integrating SAAM in domain-centric and reuse-based development processes," in *Proc. Nordic Workshop Softw. Archit.*, 1999, pp. 1–10.

[20] C.-H. Lung, S. Bot, K. Kalaichelvan, and R. Kazman, "An approach to software architecture analysis for evolution and reusability," in *Proc. Conf. Centre Adv. Stud. Collaborative Res.*, 1997, pp. 144–154.

[21] M. T. Ionita, D. K. Hammer, and H. Obbink, "Scenario-based software architecture evaluation methods: An overview," in *Proc. Workshop Methods Techn. Softw. Archit. Rev. Assessment*, 2002, pp. 1–12.

[22] M. Mattsson, H. Grahn, and F. Mårtensson, "Software architecture evaluation methods for performance, maintainability, testability, and portability," in *Proc. Int. Conf. Qual. Softw. Architectures*, 2006, pp. 18–28.

[23] L. Etxeberria and G. Sagardui, "Product-line architecture: New issues for evaluation," in *Proc. Int. Conf. Softw. Product Lines*, 2005, pp. 174–185.

[24] E. Anjos and M. Zenha-Rela, "A framework for classifying and comparing software architecture tools for quality evaluation," in *Proc. Int. Conf. Comput. Sci. Appl.*, 2011, pp. 270–282.

[25] M. A. Abbasi, D.-E.-B. B. Batool, R. Butt, and T. M. Anjum, "Comparative analysis of software architecture documentation and architecture languages," in *Proc. Asia-Pacific World Congr. Comput. Sci. Eng.*, 2016, pp. 199–204.

[26] V. R. Basili and H. D. Rombach, "The TAME project: Towards Improvement-oriented software environments," *IEEE Trans. Softw. Eng.*, vol. 14, no. 6, pp. 758–773, Jun. 1988.

[27] M. A. Babar, L. Zhu, and R. Jeffery, "A framework for classifying and comparing software architecture evaluation methods," in *Proc. Australian Softw. Eng. Conf.*, 2004, pp. 309–318.

JAKOB AXELSSON received the Ph.D. degree in computer systems from Linköping University, Linköping, Sweden, in 1997.

He spent about 15 years in the automotive industry within the Volvo Group and with Volvo Cars. He is currently a Full Professor in computer science with Mälardalen University, Västerås, Sweden, and is also a Senior Research Leader in systems-of-systems with RISE Research Institutes of Sweden, Kista, Sweden. He is the author of more than 100 research publications. His research interests include all aspects of system-of-systems engineering as well as system architecture for cyber-physical systems.

Dr. Axelsson was the recipient of Best Paper Awards at international conferences on four occasions.

DAMIR BILIC is currently working toward the industrial Ph.D. degree in software engineering with Mälardalen University, Västerås, Sweden.

He is having experience as a System Engineer in the automotive domain. His research interests include model-driven development and product line engineering, with focus on the evolution and consistency management of models in product lines.

DANIEL BRAHNEBORG (Member, IEEE) received the Ph.D. degree in computer science from Mälardalen University, Västerås, Sweden, in 2022. His Ph.D. dissertation was Improving the Efficiency and Reliability of Text Messaging Gateways, "Improving the Quality Attributes of Messaging Gateways."

He is self-employed with Braxo AB, Stockholm, Sweden, where he spends most of his time on further development of the company's flagship product and the Short Message Service (SMS) gateway Enterprise Messaging Gateway (EMG).

JOAKIM FRÖBERG received the Ph.D. degree in computer science from Mälardalen University, Västerås, Sweden, in 2007 .

He has more than 20 years of industry experience in developing software-intensive embedded systems. He is a Safety Assessor and Researcher with Safety Integrity AB, Västerås, Sweden. He has participated in many research and development projects including automotive electric drive, autonomous, remote-control, and platooning systems. He has worked with safety analysis and architecture analysis in many application areas including construction, automotive, transport, agriculture, and mining. He is active as an Assistant Supervisor for a Ph.D. student in the area safety analysis for system-of-systems. His research interests include systems engineering, safety analysis, and evaluation of system architecture.

HENRIK GUSTAVSSON is currently working toward the industrial Ph.D. degree in software engineering with Mälardalen University, Västerås, Sweden .

He is having 25 years experience in industry specializing in requirements management and integration of electrical and embedded systems, including safety-related applications. His research interest include product line engineering requirements, use cases, and architecture.

ROBBERT JONGELING (Member, IEEE) received the Ph.D. degree in software engineering in 2022. His Ph.D. dissertation was Mälardalen University, Västerås, Sweden, "Lightweight Consistency Checking for Advancing Continuous Model-Based Development in Industry."

He is an Associate Senior Lecturer in software engineering with the Department of Innovation, Design, and Engineering, Mälardalen University, Västerås, Sweden. His research interests include continuous model-based development and flexible modeling for software and systems.

DANIEL SUNDMARK received the Ph.D. degree in software engineering with Mälardalen University, Västerås, Sweden, in 2008. He is a Professor of computer science, focusing on engineering of embedded software and systems with Mälardalen University, Västerås, Sweden, where he serves as a Leader with the Software Testing Laboratory. His research interests include empirical studies of industrial software engineering, with a particular focus on different aspects of testing of embedded systems and embedded software engineering from a life-cycle perspective .