Variability assessment in software product families

Sybren Deelstra a*, Marco Sinnema a, Jan Bosch b

a Océ Technologies B.V., P.O. Box 101, 5900 MA Venlo, The Netherlands
b Intuit, 2500 Garcia Avenue, Mountain View, CA 94043, United States

ARTICLE INFO
Article history:
Received 15 September 2006
Received in revised form 31 March 2008
Accepted 2 April 2008
Available online 10 April 2008

Keywords:
Software product families
Variability
Assessment
Evolution

ABSTRACT
Software variability management is a key factor in the success of software systems and software product families. An important aspect of software variability management is the evolution of variability in response to changing markets, business needs, and advances in technology. To be able to determine whether, when, and how variability should evolve, we have developed the COVAMOF software variability assessment method (COSVAM). The contribution of COSVAM is that it is a novel, and industry-strength assessment process that addresses the issues that are associated to the current variability assessment practice. In this paper, we present the successful validation of COSVAM in an industrial software product family.

1. Introduction

The widespread use of product family engineering in industry is driven by the potential increase in quality and productivity of software development that it offers [5,9,31]. To achieve this potential, a software product family maintains a product family architecture and set of reusable components to exploit the similarities of its members. The product family supports product diversification through variability, i.e., the ability of a software system or artifact to be extended, changed, customized or configured for use in a specific context.

The variability provided by a product family has to undergo continual and timely change, or the product family will risk using the ability to effectively exploit the similarities of its members. Being able to determine whether, when and how variability will risk responding to changing markets, business needs, and advances in technology, however, i.e., variability assessment, is a non-trivial task. To assist engineers in answering these questions, we developed the COVAMOF software variability assessment method (COSVAM).

COSVAM is based on the experiences and ideas formed during the ConIPF project [19], a research project that involved two industrial organizations, i.e., Robert Bosch and Thales Nederland. COSVAM is the first technique for assessing variability with respect to the needs of a set of product scenarios. The five steps of COSVAM (identify assessment goal, specify provided variability, specify required variability, evaluate variability, interpret assessment results) form a structured technique that can be tuned to address a variety of situations where the question of whether, how and when to evolve variability is applicable. Organizations can use COSVAM to produce a wide variety of results, such as an overview of mismatches, different solution scenarios and a selection of the optimal solutions. It defines the way in which information has to be selected and interpreted to produce those results. COSVAM further provides a means to externalize the provided variability of a product family to a variability model.

The contribution of this paper is that it presents a novel and industry-strength variability assessment method that solves the variability assessment issues in practice. The assessment method was successfully validated in an industrial software product line. In this paper, we provide descriptions and examples of each of the steps of COSVAM, and discuss the validation of COSVAM with a case study at Dacolian B.V.

To be able to present COSVAM in an understandable manner, however, this introduction first explains variability in terms of our variability management framework COVAMOF, and provides a more detailed motivation for the need of a variability assessment technique.

1.1. Variability management with COVAMOF

Properly managing variability is one of the key success factors of a software product family [6]. In practice, however, variability management suffers from a number of issues that prevent organizations from exploiting the full benefits of software product families [6,12]. To address these issues, we developed our variability

* Corresponding author.
E-mail addresses: sybren.deelstra@oce.com (S. Deelstra), marco.sinnema@oce.com (M. Sinnema), jan.bosch@intuit.com (J. Bosch).
management framework COVAMOF. As we show in Fig. 1, COVAMOF consists of a number of main parts, i.e., a meta-model, a language, a tool-suite, a derivation process, and the variability assessment method COSVAM.

These parts are built around the notion to model variability uniformly over the lifecycle phases and abstraction layers (e.g., features, architecture, component implementations), and modeling variation points and dependencies as first-class citizens. COVAMOF variability models specify views on the variability of a product family, and are created and configured with our COVAMOF-VS tool-suite. Below, we briefly discuss the main aspects of variability in terms of a number of entities in the COVAMOF Meta-Model (see Fig. 2). For a more elaborate discussion on the model and other parts of COVAMOF, see e.g., Sinnema et al. [26].

1.1.1. Variation points and variants

Variation points identify locations in software systems or artifacts at which a choice can be made between values, or zero or more variants. Variation points are recognized as elements that facilitate systematic documentation and traceability of variability, development for and with reuse [7,20], assessment, and evolution. As such, variation points are not by-products of design and implementation of variability, but are identified as central elements in managing variability.

Variation points can occur in all abstraction layers, such as features, architecture, and component implementations. Each variation point is associated with a value or zero or more variants, and are categorized to five basic types, i.e., optional (zero or one variant out of 1,...,m associated variants), alternative (1 out of 1,...,m), optional variant (0,...,n out of 1,...,m), variant (1,...,n out of 1,...,m), and value (a value that can be chosen within a predefined range).

1.1.2. Realizations

In COVAMOF, variation points in one abstraction layer can realize the variability in a higher abstraction layer, e.g., an optional architectural component that realizes the choice between two features in the feature tree. These relations are called Realization Relations. Variability that is not realized by variation points in a lower abstraction layer are implemented by a realization mechanism. Over the past few years, several of these mechanisms have been identified for different abstraction levels. The realization mechanisms impose a binding time on the variation point, i.e., the latest moment in the development lifecycle at which engineers can bind the variation point. Examples of realization mechanisms include parameterization, conditional compilation, and dynamically linked libraries (see, e.g., Svahnberg et al. [29]).

1.1.3. Dependencies

The dependency entities represent system properties such as quality attributes. Where possible, they specify a function of how the choices at the variation points in the product family influence a system property value, as well as the valid range for the system property value. For some system properties this influence can be described very well. For example, two variants that exclude each other, result in a function to true (when none or only one of the variants is selected in a product) or false (when both variants are selected in a product). Tools can automatically check the validity of those dependencies by comparing the variability model with the dependency and the configuration at hand. For other system properties, such as the system performance or the required memory, this influence is much more complicated to describe. This complication for specifying those dependencies is primarily caused by the fact that the knowledge that is available about these dependencies is incomplete, and imprecise. Engineers can therefore at most specify the influence of individual variation points in terms of changes that have a positive or negative effect on the system property of the dependency. COVAMOF addresses this issue by allowing engineers to specify the influence of variation points on dependencies as a combination of formal functions together with associations and dependency interactions (explained below).

1.1.4. Associations

The association entities specify which variation points have a significant impact on a dependency value, and how estimated system property values can be refined. For example, a directional Association specifies that we may not exactly know how the variation point influences the dependency, but at least know in which direction the value will change. The impact attribute of an association specifies the amount of the influence of an associated variation point on the system property value.

1.1.5. Dependency interactions

In addition to the unavailability of knowledge, dependencies typically suffer from dependency Interactions as well. Variation points can be involved in multiple dependencies. Meeting the constraints for a particular dependency value may therefore affect other dependency values. Optimizing dependency values is therefore particularly complicated, as this may invalidate the value of other dependencies. To address this issue, dependency interaction entities specify how engineers can cope with specific optimization problems.

1.1.6. Reference data

Reference data entities contain historical data about the dependencies. They capture specific choices that were made at the associated variation points, and store the system property value that was measured during testing. The reference data is used to determine a starting point for configuration, and for estimating system property values.
1.1.7. Products

Product entities specify the variants and values that have been selected for a product configuration, as well as the constraints that are imposed on the dependency values. These entities furthermore contain information about the customer of the product.

1.2. Variability assessment overview

Now that we have briefly explained variability, the next question we need to answer is: why is variability assessment necessary? To answer this question, we go back to the work of Lehman on software evolution. He noted that as the world around us continually changes, the resulting change in purpose and context may render software products useless. It was therefore that Lehman formulated the following law on software evolution: “A useful software system must undergo continual and timely change or it risks losing market share” [23]. This law applies to all products in a product family.

Although variability in the product family architecture and components anticipates some of the changes in space (different products) and time (different versions of products), not all future changes can be predicted or included in the product family. Consequently, once the product family is in place, at some point in the lifecycle, evolution will force the product family to handle new functionality and thus previously discarded unforeseen differences. In the same way that products need to undergo continual change, variability therefore has to undergo continual and timely change as well, or a product family will risk losing the ability to effectively exploit the similarities of its members.

The key challenge in this context is: “in what way can we determine whether, how, and when variability should evolve?”. A technique that deals with answering this question is what we refer to as variability assessment. Such a technique answers the question above by analyzing the mismatch between (1) the variability in the product family artifacts and (2) the variability that is demanded as necessary by the differences in functionality and quality in a set of product scenarios (see Fig. 3). We call the first type of variability, provided variability, and the second type, required variability. We call the mismatch between them a variability mismatch.

That variability assessment is indeed relevant becomes clear from examining five common activities for software product families in which the ‘whether-how-when’ question appears: (1) determine the ability of the product family to support a new product, (2) during product derivation, determine whether mismatches should be implemented in product specific artifacts or integrated in the product family, (3) collecting input data for release planning, (4) assess the impact of new features that cross-cut the existing product portfolio, (5) determine whether all provided variability is still necessary.

1.3. Remainder of this paper

We structured our discussion of COSVAM as follows. In the following section, we list a number of variability assessment issues, and discuss related work. In Section 3, we present our research methodology, and in Section 4, we present the case study organization. In Section 5, we present the activities of the five main steps of COSVAM. Each activity contains a description that specifies the expected input and output. We illustrate each activity with the results of our case study. In Section 5.5, we present the experiences of applying COSVAM in the industrial environment of the case study. In Section 5.6, we conclude this paper with a discussion on how COSVAM addresses the variability assessment issues, as well as a discussion on future work.

2. Variability issues and related work

As software product families in industry have been widely adopted and evolve constantly, organizations that employ product families already perform some form of variability assessment. In this section, we discuss the issues with current approaches (both in practice and related work).

2.1. Variability assessment issues

Before a new product is derived, for example, a specification of the product functionality and quality is handed to, typically, software architects. The task of these architects is to assess how much effort will be associated in delivering the product with this specific set of functionality and quality. From what we have seen in several case studies that our group has participated over the years (e.g., as described in Deelstra et al. [12], or the Dacolian case described in this paper), current approaches that are used to determine whether, when, and how variability should evolve, are associated to a number of methodological and knowledge issues.

The methodological issues refer to the problems associated to the principles and procedures of current approaches:

- **Unstructured**: Variability assessment is often done by architects without explicit methodological guidance. Instead, they employ an informal process based on their own common sense and experience. These informal processes are typically highly unpredictable with respect to their outcome and required effort.
- **Reactive instead of proactive**: Assessments are furthermore often only applied in case of immediate problems or needs. As a consequence, these assessments suffer from time-pressure and lack of availability of experts; both for the assessment process, and for applying solutions.
- **Generalized instead of optimal decisions**: A third issue is that, in some cases, decisions with respect to evolving variability are generalized over a number of features. In Deelstra et al. [12], we present some extreme forms of generalization we found in industry. For example, one business unit would apply all necessary changes product specifically for each release of the product family, while another business unit would incorporate all necessary changes in the reusable product family artifacts. Both cases lead to problems. Where in the first case the full reuse potential
of the product family is not utilized (they re-implement similar functionality in each single product), the second case leads to an unnecessary increase of complexity.

- **Lack of removing obsolete variability**: After a while, the purpose of certain variation points and variants may disappear. Functionality specific to some products can become part of the core functionality of all product family members [6], or perceived alternatives may not be needed after all. Not removing them lead to a situation in which the complexity of the product family only increases during evolution, and the predictability and traceability only decreases. During the case studies we described in [12], for example, the interviewees indicated the existence of obsolete variation points from which no one knew what they are for, let alone what the optimal value was.

- **Addressing only one layer of abstraction**: Most existing assessment techniques only focus on one layer of abstraction, i.e., either the architecture (e.g., [8,15]), or its implementation in code (e.g., [4,22]). However, variability is a concern that cross-cuts all layers of abstraction. In case a detailed list of changes to the product family artifacts (both architecture and components) is required, this issue therefore drives the need for a technique that is able to address all these layers in a uniform fashion.

The knowledge issues refer to the problems associated to the information on which decisions in an assessment are based.

- **Implicit variability**: The last methodological issue (addressing only one layer of abstraction) suggests using a variability model that relates variability information across different abstraction layers. In many organizations, however, no complete and explicit model is available that covers all these layers. As the time and effort that is available for an assessment is limited, specifying a complete explicit model is often not an option. Not using a model at all also proofs problematic, however, as it is difficult to keep an overview of all variation points and their relations [12].

- **Neglecting implementation dependencies**: Even if particular options for functionality and quality are independent from a problem space perspective, the design and implementation of a product family can create additional dependencies between them. The consequence of dependencies as a result of implementation is twofold. First, not all combinations of options provided by a product family can be offered in one product without modification. This means that even if all required options are provided by the product family, the required combination of options may not be available. Second, effort estimates for new functionality and quality cannot be considered independent from other changes, as those may also have implementation dependencies.

- **Insufficient number of alternative solutions**: When a variability mismatch occurs, several solution strategies may exist to address this mismatch. For example, the software architect has to decide whether to solve a mismatch product specifically, in the reuse infrastructure, or not at all. In addition, he or she may choose from different mechanisms to solve the mismatch. The choice for a particular solution depends on a trade-off between pros and cons of the potential solutions. Examples of pros and cons of a solution are: whether it introduces incompatibilities in the asset base due to new dependencies, whether it imposes the use of immature or unstable technology, and how it affects the effort associated with other changes. The issue we address here involves software architects that only consider a very small number of alternatives, rather than carefully looking for the optimal solution [6].

These knowledge issues cause assessments to produce non-optimal and inaccurate results. The consequence is that, during product derivation, unexpected incompatibilities are identified. Deelstra et al. [12] explains that these incompatibilities have a profound impact on the total effort and time-to-market for the product at hand.

### 2.2. Related work

In the Introduction, we already mentioned that existing techniques have been suggested for variability assessment. In the discussion on related work below, we relate COSVAM to existing work on product families, and discuss why existing approaches are not suited to address all variability assessment issues we discussed above.

#### 2.2.1. Variability

Since the introduction of the notion of a variation point [20], variability has received substantial attention, especially in the product family community. Over the past few years, several variability realization techniques have been identified, e.g., by van Gurp and Savolainen [18], Fritsch et al. [16], and Jacobson et al. [20]. Approaches to variability modeling are available in the form of, for example, Asikainen et al. [1], Becker [3], and Czarnecki [10]. All these modeling approaches share some of the concepts and modeling facilities in our variability modeling framework COVAMOF. None of them addresses all issues that we have identified and solved in COVAMOF, however [27].

#### 2.2.2. FAST and SEI’s product line practice

In [31], Weiss and Lai formulate basic assumptions with respect to product families, from which two are particularly important in the context of this article: “it is possible to predict the changes that are likely to be needed to a system over its lifecycle”, and “it is possible to take advantage of these predicted changes”. When it comes to actually determining changes to variability, however, the book lacks precision (p. 198): “… You can adapt standard change management techniques to FAST projects, so the FAST PASTA model does not elaborate on those aspects in any great detail” [31]. Also in the SEI’s Product Line Practices and Patterns book, the evaluation of variability is quoted as an example of an important evaluation. The book, however, does not present a technique to perform these evaluations. Rather, it suggests modifying existing architecture assessment methods to accomplish this goal (pp. 77–83).

#### 2.2.3. Investment analysis

In [32], James Whitey provides an investment analysis approach that focuses on maximizing ROI of product line assets. Robertson and Ulrich [25] also evaluate economical aspects of a product family and deal with planning and scoping the product family architecture. In [11], however, DeBaud and Schmid provide a similar but more general approach. They also claim that product-centric commonality and variability analysis is better than a domain based view, as the latter provides a flawed economic model for making scoping decisions [11]. The approaches, however, are based on rough estimates, and focus on the question if certain features, assets or products should be part of the product family rather than how required variability should be realized in the product family artifacts.

#### 2.2.4. Assessment

Assessments typically consist of five steps, i.e., goal specification, specification of the provided aspect, specification of the required aspect, analyzing the difference between the provided and required aspect, and interpreting the results. Examples of these approaches are ATAM [8], ALMA [2], and SALUTA [15]. These three approaches, respectively, assess trade-offs between quality attributes, maintainability, and usability in software architectures. The approaches differ with respect to the required information,
elicitation, and specification of the scenarios. They focus on analyzing the architecture of single systems, rather than being suitable for all layers of abstraction in a product family.

An approach that does address variability, is the approach presented by Wijnstra [33]. The approach presents a high-level discussion on extracting variability information, and, based on this information, evaluates the provided variability with respect to best practices. In contrast to COSVAM, it is restricted to end-user variability, does not provide specific steps for building up a provided variability specification, and is not focused on specific variability needs of the product family members.

2.2.5. Change management

Change management is a process for ensuring that changes to a software system or product family are traceable, carefully planned, and motivated. The change management process is typically a high-level description of how a change should be handled, defines standard deliverables, as well as an organizational structure. COSVAM neatly fits into the change management process in product families. Where the descriptions in change management process usually do not go further then saying there are steps such as 'propose change', or 'evaluate change', COSVAM provides the details that are required to actually propose and evaluate changes to the variability.

3. Research methodology

Validation of software engineering research is, in our opinion, of high importance in order to further the field. During the design of COSVAM, we carefully evaluated alternative evaluation approaches. As we consider industrial validation to be of the highest importance for the validation, we selected an industrial case study approach as the preferred mechanism. The primary purpose of the case study was to evaluate the COSVAM process, rather than the specific outcome of applying COSVAM. In other words, we were interested in getting an answer to the question whether COSVAM could be applied in an industrial context, and whether it assists in producing better results, rather than evaluating the results themselves. The case study was designed as follows:

- **Preparation:** The case study organization faced the same problems as the ones we discussed in Section 2.1. Due to earlier contacts, the case study organization (mainly the assessment leader) was already familiar with the topic of variability, and the detailed steps of COSVAM.
- **Procedures:** In this case study, the researchers acted as observers. The organization was fully in charge of the assessment process, the researchers did not participate in the assessment, nor in producing the results of each of the process steps. The only questions that would be answered during the assessment would be specific questions on the process itself.
- **Data collection:** We collected the results produced by the assessment. In addition, eight months after the assessment, we interviewed the assessment leader for an evaluation of COSVAM. The interview was guided by a list of questions, with room for open, unstructured questions and discussions. Sixteen months after the assessment we conducted a second interview to verify whether the evaluation would be the same for the long term.

4. Case study organization: Dacolian

The COVAMOF variability assessment method (COSVAM) was applied at an industrial organization called Dacolian. Dacolian B.V. is the leading supplier of OEM software modules for intelligent traffic systems that utilize automatic license plate recognition (ALPR). These ALPR modules are developed using a product family called Intrada ALPR. Typical products consist of systems that deliver: based on real-time input images, abstract information on the actual contents of the images, e.g., the presence of traffic, vehicle type and brand, car registration numbers, etc. Intrada ALPR modules are used in systems for all three major ALPR application areas Tolling, Enforcement, and Access control/Parking. Each of these application areas provides its own distinct set of performance requirements and image characteristics.

The reusable asset base of the Intrada family evolved over the past 7 years into an asset base that consists of approximately 11 million lines of code. In Fig. 4, we depict the products that were sold during the 2 years prior to case study. In that period, the Intrada ALPR product family contained four different products groups (the space dimension), and all product groups evolved over several generations (the time dimension). When the assessment was initiated, the most recent version of the Intrada ALPR family was 4.0.0, but several customers still had products in the field that had been built with older releases of the Intrada ALPR family. In the case study we present in this paper, COSVAM was applied to plan the next release of the Intrada ALPR product family. In addition to the evaluation, we use the results of assessment to illustrate each activity in our method.

![Fig. 4. Intrada ALPR product evolution.](image-url)
5. COSVAM: the COVAMOF software variability assessment method

5.1. Key elements

To address the variability assessment issues from Section 2, we developed the COVAMOF software variability assessment method (COSVAM). Fig. 5 shows the main deliverables that are produced during COSVAM, i.e., the COVAMOF provided variability model, COVAMOF required variability model, mismatches, solution scenarios, and required modifications:

- **COVAMOF provided variability model**: This is a COVAMOF model of the variability in the product family artifacts (see Section 1.1).

- **COVAMOF required variability model**: This is also a COVAMOF model. However, this model specifies the variability that is required by the functionality and quality required for a number of product scenarios. In COSVAM, this model is specified in terms of configurations of the COVAMOF provided variability model.

- **Mismatches**: The mismatches are the differences between the provided and required variability, such as new variation points, conflicts between variants, and unmet quality attribute values. As the COVAMOF required variability model is specified in terms of the COVAMOF provided variability model, the mismatches are based on new entities and conflicts in the COVAMOF required variability model, rather than a comparison between the two models.

- **Solution scenarios**: The solution scenarios specify possible ways to solve the variability mismatches. Each solution scenario has a different impact on the product family.

- **Required modifications**: Required modifications specify a set of changes to the product family artifacts that address the needs and constraints of the product family. This set is based on a comparison of the pros and cons of the solution scenarios.

To produce these deliverables, we defined an iterative process. This process divides the assessment problem into five steps (see Fig. 6). COSVAM aims for manageability, and repeatability, and therefore distinguishes multiple goals, explicitly identifies the sources of information, and forces assumptions and decisions to be made explicit. Each step describes how results should be obtained, processed, and interpreted. Below, we discuss the key elements of each step.

5.1.1. Steps 1 and 5

As COSVAM is designed to be used in different situations, and with different goals in mind, the method starts with clearly demarcating the assessment goal, and ends with an interpretation step. The first step, i.e., identify assessment goal, identifies the assessment context, required outcome, scope, and required team members, and plans the remaining steps. The last step, i.e., interpretation, interprets the evaluation outcome with respect to the assessment goal. These steps allow structuring this process, viz. to tune the input, output and activities of each step to a particular situation. By forcing assumptions and decisions to be made ex-
plicit, COSVAM furthermore allows proposing changes to the provided variability that address the needs and constraints, where the rationale behind these changes can be traced.

5.1.2. Steps 2 and 3

The second and third step involve specifying the provided and the required variability, respectively. As in many product family organizations variability is implicit, the purpose of the second step in COSVAM is to specify the provided variability uniformly over several abstraction layers. As the time for an assessment is typically limited, COSVAM addresses the problem of not being able to create a complete specification by focusing on a particular scope (identified in the first step), and iterating between specifying provided and required variability.

The purpose of the third step is to specify the variability that is required to accommodate the combinations of functionality and quality in the set of product family members that are within the assessment scope (i.e., product scenarios). Depending on the goal of the assessment, this step also existing products as scenarios to be able to identify obsolete variability, and future products to be able to proactive evolve the product family.

Both steps use a COVAMOF model to specify the types of variability. To prevent a complicated comparison between models that potentially have a different decomposition and naming of entities, the COSVAM specifies the required variability in terms of the configuration of existing entities in the provided variability model, as well as differences to existing entities in that model. As a part of the required variability typically matches the provided variability, this also saves modeling effort in specifying required variability.

5.1.3. Step 4

The fourth step, i.e., variability evaluation, involves finding out how well the required variability is supported by the provided variability, and what changes can be made to accommodate mismatches between them. These mismatches are clustered, and for each cluster several alternative solutions are designed (i.e., solution scenarios). For each solution scenario the impact is determined in terms of, for example, required effort, as well as impact on design, time-to-market, tools, performance, and testing.

Each step of COSVAM consists of a number of activities. In this section, we discuss these activities in detail. We describe how each activity is performed, and how results are delivered. We illustrate each activity with our case study at Dacolian B.V. The focus of our discussion is on the situation where COSVAM is used to collect input data for release planning.

5.2. Step 1 identify assessment goal

Software variability assessment can pursue different goals, under very different circumstances. Both have an impact on how the assessment should be performed. The purpose of the first step of COSVAM is therefore to identify the goal and circumstances under which the assessment is performed. Making these goals and circumstances explicit ensures all members of the assessment team face the same direction. It also allows tuning the assessment input, output, and process, so that the assessment suits the goal, and addresses the circumstances.

The output of this step is an assessment goal document, which is maintained and used throughout the whole COSVAM assessment. It contains a description of the goal, circumstances, their influences, and organization of the assessment. To produce the assessment goal document, COSVAM defines five activities in this step (see Fig. 7), i.e., (Activity 1a), identify context (Activity 1b), define assessment outcome, (Activity 1b), select assessment scope (Activity 1c), identify assessment team (Activity 1d), and plan the assessment (Activity 1e). Since this step mainly builds upon how assessments are prepared in other assessment methods such as Clements et al. [8], and Maccari [24], we only briefly discuss each activity below.

5.2.1. Activity 1a – identify context

The reason why the assessment was initiated has an impact on the required input and output of the assessment. For example, it influences the time available, the level of cooperation, and acceptance of results afterwards. Due to the impact of the context on the assessment, identifying the assessment goal starts with identifying this context. The result of this activity is a description of the reason and way of initiating the assessment, as well as an overview of factors that may influence the assessment.

5.2.2. Activity 1b – define assessment outcome

Next to a precise definition of the assessment results and the impact on different parts of the assessment process, the definition of the assessment outcome also consists of a description of how the results will be communicated to the organization.

5.2.3. Activity 1c – select scope

As time and effort is limited, the assessment team typically has to limit the required and provided variability scope. Limiting the required variability scope limits the timeframe from which products and product releases are considered, as well as level of detail of features considered. Limiting the provided variability scope limits the layers or subsystems that are considered during the assessment. Next to reducing assessment effort, limiting the assessment scope introduces risks; the assessment process may require additional iterations in case of missing information (affecting predictability), and the results may introduce problems if they are incorrect. To ensure traceability of these decisions, the result of this activity therefore not only consists of a description of which provided and required variability scope was selected, but also the rationale, impact and risks of the selected scope.

![COSVAM Assessment Process](image)

**Fig. 7.** Identify assessment goal.
5.2.4. Activity 1d – identify assessment team

Selecting the participants that are required to be able to deliver the desired assessment outcome from Activity 1b, within the context and selected provided and required variability scopes from Activities 1a and 1c, respectively. The result of this activity is a list of experts and their role in the assessment.

5.2.5. Activity 1e – plan the assessment

The combined results from the activities above form the goal of the assessment and are specified in the assessment goal document. Once this goal has been specified, it is important to plan the assessment, obtain commitment from the organization for both the assessment and use of assessment results, and to assemble the assessment team.

5.2.5.1. Dacolian case study. As we discussed in Section 4, COSVAM was initiated at Dacolian to plan the new release of the Intrada ALPR product family, i.e., version 5.0.0. The aim of assessment for release planning was twofold. The first aim of the assessment was to be able to communicate to customers, as early as possible, which variability would be included in the product family. The second aim was to communicate to the organization, which changes we have to make to the product family artifacts.

The assessment window spanned 18 months. This meant Dacolian took into account the products that they expected to sell in the next one and a half year. Dacolian focused the assessment on products in the Tolling, and Law Enforcement market segments, and not on the access control segment since the latter segment only represented a small portion of their product portfolio.

The effort for each of the remaining steps of the assessment was roughly divided as follows. The specification of required variability was estimated to take a maximum of 1 week. The analysis of mismatches in the evaluation step would be supported by tools. The analysis was estimated to take anywhere from 2 h to 4 days per configuration, depending on the mismatches that would be identified during the assessment. The automated analysis would allow Dacolian to run multiple analyses in parallel. Any analysis that would likely take more than 3 days would be simplified to a quicker, but less accurate derivative analysis, or expert judgment. The design and impact analysis of the solutions was estimated to take a maximum of 3 weeks. One additional day was estimated for the interpretation step. As a result, each of the steps would fit in the 5 weeks set for the assessment.

5.3. Step 2 – specify provided variability

In many organizations, variability is implicit, or specified in different types of models that have no explicit relations across abstraction layers. The purpose of this step is therefore to specify the provided variability of a product family in a variability model where variability is linked across abstraction layers. COSVAM uses a COVAMOF model to create such a specification (see Section 1 for a brief discussion on the main entities in COVAMOF).

As the time and effort for an assessment is typically limited, it is usually impossible to create a COVAMOF model that completely represents all variability provided by a product family. In COSVAM, this problem is addressed in two ways. First, in Activity 1c, the provided variability scope has been limited through the abstraction layer and subsystem scopes. Second, the assessment team continually weighs how much information will be externalized throughout this step, and how much will left implicit. Stopping criteria for activities in this step are when the experts in the assessment team are confident that externalizing more information will not significantly impact the assessment results.

The result of this step is thus a COVAMOF model that partially covers the provided variability of a product family. This specification models variability in terms of a COVAMOF model, where variation points and dependencies are first-class and related across lifecycle phases. To produce such a model, the provided variability specification step is structured in five activities: Identify and obtain information sources for provided variability (Activity 2a), identify variation points (Activity 2b), unify set of variation points (Activity 2c), identify variants (Activity 2d), and determine realization rules and dependencies (Activity 2e). Fig. 8 summarizes these activities.

5.3.1. Activity 2a – identify and obtain information sources for provided variability

The specification step starts with identifying information sources that can be used to model the variability provided by the product family artifacts. Potential information sources include: expert knowledge, feature diagrams, the product family architecture, component documentation, product configurations, make-files, and source code. In case this activity identifies that a provided variability model is already available in the right form and detail, the other activities can be skipped completely, and the assessment can proceed with the next step of the assessment, i.e., specifying required variability (Table 1).

5.3.1.1. Dacolian case study. At the time of the assessment there was no COVAMOF variability model of all their ALPR product family artifacts yet. They therefore developed a new model during this step, for which they identified a total of 10 different types of input. A part of the output of this activity is shown in Table 2.

5.3.2. Activity 2b – identify variation points

Once the provided variability sources are obtained in Activity 2a, the next activity is to use these sources to identify the first of the two main aspects of variability, i.e., the variation points (see...
Table 3). These variation points identify locations in product family artifacts that are within the provided variability scope, and that provide the ability to vary functionality and quality between different products. A number of approaches can be used find these variation points:

- Some existing notations, such as for feature diagrams, already specify variability. The assessment team can identify variation points directly from these notations.
- Interview provided variability experts, and let them draw from experience to identify important variation points.
- Compare the architecture, selected components, and parameter settings in different product configurations to identify differences.
- Search for patterns in design and implementation that may indicate variability. These patterns consist of design patterns [17] such as factories and abstract classes, but also code statements such as if-statements and pre-compiler directives.
- Search for comments that deal with choices in design and code. Although specifying criteria for an automated search with full-coverage may prove complicated, comments can be identified by experts, and often serve as valuable addition to the other means for externalizing variability above.

5.3.2.1. Dacolian case study. The experts were asked to identify variability in features and draw different architectures. They used a search for code statements such as pre-compiler, if-, and switch-statements in the source files and c-interface specifications to identify variation points in the source code. The internal module documentation, component make-files and partial component configurations were used to identify variation in the component structure. The reference manuals for customers and tools were used to identify post-deployment variability. The MDA-based tool was furthermore inspected to identify variability with respect to the generated source code. This activity identified a total of 6872 variation points (see Table 4).

5.3.3. Activity 2c – unify set of variation points

The identification of variation points results in a set of variation points that originate from different sources. The purpose of unification is to identify multiple representations of the same variation point, to relate variation points with Realization Relations, to remove too low-level variation points, and to determine their attributes. The unification process furthermore identifies settings that are identical across multiple product configurations.

In this activity, assessment team members develop the initial COVAMOF variability model (see Table 5) from the identified variation points (Activity 2b) and additional information from the pro-
provided variability sources (Activity 2a). For each unique identified variation point in the identified variation points they create a new variation point entity in this model, and specify, where known, its attributes. The most important attributes are the name, the description, the references to the associated rows in the “Identified Variation Points” table, the type, the abstraction layer, the binding time, whether it’s opened or closed, and its realization technique. Similar, for each identified realization, they create a new COVAMOF Realization Relation entity in the variability model, which is linked to the variation points (see Section 1.1).

5.3.3.1. Dacolian case study. During the previous activity, the assessment team made sure no multiple representations from the same variation point were identified. Most of the variation points identified during the previous activity (94.6%) were value variation points that were identified in two components (called Neural-Network and Matcher) that are generated by MDA-based tools, and commented variables in the other source files. The assessment team decided not to externalize all information regarding these component implementations, as the variability provided these components was considered to be well isolated from other variability. In addition, it was expected that the variability provided by these components would be enough to address any changes in the ALPR domain during the time scope. This reduced the total amount of value variation points to be explicitly considered during the assessment to approximately 1%. The remaining variation points were almost equally distributed among the other types (1.7% optional, 1.2% alternative, 1% optional variant, and 1.5% variant). Most of the value variation points had their binding time and closing time at compilation time. For a sample of the resulting COVAMOF model of Step 2, see Fig. 9.

5.3.4. Activity 2d – identify variants

The unified set of variation points in the initial variability model is then populated with variants (see Table 6). A number of variants have already been encountered earlier, as these variants served to identify variation points in the first place. This set of variants is not always complete however, as only a subset of product configurations may have been used, or because the product configurations only use a subset of the available variants. The set of variants can therefore be extended by using additional information from the provided variability sources, such as interviewing experts, inspecting additional product configurations, and analyzing the set of shared product family assets. This process is guided by the variation points in the COVAMOF model.

5.3.4.1. Dacolian case study. The previous step primarily yielded closed variation points with the latest binding time at post-deployment. As a consequence all variants of these variation points were specified in the source code, and identified during the previous activities. For the compile-time variation points there are only a limited amount of variants as well, so all of those variants had also been identified in the previous activities. For a sample of the resulting COVAMOF model of Step 2, see Fig. 9.

5.3.5. Activity 2e – determine realization rules and dependencies

After the unification of the set of variation points, the Realization Relations specify which variation points are related to each other. Second, the variability model does not specify which combinations of variants are permitted, and third, it does not specify how the selection of variants affects system properties such as quality attributes (see also Section 1.1). This activity therefore deals with adding realization rules and dependencies to the provided variability model. After this activity, the COVAMOF provided variability model is ready to be used in subsequent steps (see Table 7).

Besides interviewing experts and studying documentation, there are specific techniques that can be used to determine realization rules and dependencies. A fairly simple technique is to modify the settings for choices in existing product configurations, and to test these modified configurations. A technique that takes this approach a step further, and that uses test results from existing product configurations is called sensitivity analysis. This approach originates from Taguchi’s robust design method [30]. In Taguchi’s method it is used to determine the influence of single parameters from a set of data points where multiple parameters were changed between measurements. We can translate this to the problem in product families. The values of the system properties of existing product configurations are our measurements. These product configurations thus define data points, where multiple choices were changed between measurements.

Finally, the assessment team enriches the knowledge on dependencies in the variability model with dependency interactions, typically by using derivation knowledge of product family experts. Note that specifying the dependencies is a time-consuming task. Here, the assessment team thus also has to weigh how detailed the analysis of the dependencies should be. In Sinnema et al. [26] we describe how COVAMOF allows the team to limit the level of detail of the knowledge on the influence of variation points on system properties.
5.3.5.1. Dacolian case study. During the unification of the set of variation points, the assessment team related the variation points with Realization Relations. They did not address the realization rules explicitly in this activity, however. The main reason for not explicitly specifying realization rules was that the experts indicated that the variability in the Intrada ALPR product family was organized in such a fashion that choices in one layer straightforwardly map to choices in another layer. According to these experts, the provided variability also hardly contained any implementation dependencies that would cause combinations of variants not being supported by variation points in lower layers. In addition, most realization rules were already formalized in the source code.

The primary action during this activity therefore comprised of identifying natural dependencies between features (through interviews), and the variation points that influence the main quality attributes (through interviews, and product configurations). To identify how the variation points influence the main quality attributes, several configurations were tested during the lifecycle of the previous release. These comprised of running 100 typical configurations on 70 datasets (each containing millions of images). For a sample of the resulting COVAMOF model see Figs. 9 and 10. Fig. 10 shows the result of incrementally building up the COVAMOF model. It features a screenshot where the COVAMOF-VS Add-ins visualize a part of the variability in the ALPR product family.

5.4. Step 3 – specify required variability

The previous step results in a model of the provided variability. In the third step of COSVAM, the participants of the assessment (specified in Step 1) use this model to specify the required variability. The required variability model captures the variability that is required to accommodate the combinations of functionality and quality in the set of product family members that are within the required variability scope. This required variability scope was already defined during the first step of COSVAM (see Activity 1c).

This step consists of three activities, i.e., identify and obtain information sources for required variability (Activity 3a), construct product scenarios (Activity 3b), and construct model (Activity 3c). We summarize these activities in Fig. 11. The impact on process and influencing factors from Step 1, and the provided variability model from Activity 2e serve as input, and the required variability model is the output of this step. In the next step (Step 4 – variability evaluation), the required variability is compared to the provided variability. To prevent a complicated comparison between models that potentially have a different decomposition and naming of entities, the COSVAM specifies the required variability in terms of the configuration of existing entities in the provided variability model, as well as differences to existing entities in that model. An additional benefit of this solution is that effort is saved in modeling the required variability. For a more detailed explanation of this solution, see Activity 3c.

5.4.1. Activity 3a – identify and obtain information sources for required variability

Specifying required variability starts with selecting the information sources that are necessary to determine the required variability (see Table 8). Example sources include existing and future product specifications, market analysis, different roadmaps, internal variability requirements, and a catalog of requirements requested by customers. The selection of these sources depends on the required variability scopes (see Activity 1c), as well as the availability of information. The required variability scope limits the required information sources to a particular subset of products, and a particular subset of features. The availability of information determines what information is part of the input of the assessor.
ment, and what information has to be constructed during the activities in this step. When future product specifications are already available, for example, they are part of the information sources. Otherwise, specifying future products is part of the activities in this step, and thus market analysis, customer requests, and roadmaps are necessary inputs. For more examples of sources and pointers to literature, see Clements and Northrop [9].

5.4.1.1. Dacolian case study. Three sources were identified for specifying required variability, i.e., a roadmap from innovation, product descriptions of products that had been built on top of version 3.0 and 4.0 of the Intrada ALPR family (see Fig. 4), and a change request catalog, that is divided into internal and external change request. For the previous release, the change request catalog contained 50 change requests. Of these 50 requests, 35 were internal request, and 12 originated from existing users. For the new release, the catalog contained 100 requests that were mapped onto 12 feature branches. The number of internal requests and requests from existing users remained constant over the previous and new release. The increase in requests is thus primarily due to the increase in visibility and reputation of the ALPR products, which yielded a substantial increase in potential customers. This also meant that a number of the request were outside of the existing domain of the Intrada ALPR product family (e.g., from the existing image recognition that involve country names, to image recognition that involve container numbers). The output of this activity is described in Table 9.

5.4.2. Activity 3b – construct product scenarios

The next activity is using the required variability sources to construct the product scenarios. Product scenarios consist of an overview of the functionality and quality of products that exist or are perceived within the required variability scope of the assessment (i.e., scope in space and time). The scenario construction activity breaks down into the following aspects (see Table 10).

- The assessment team defines (or generates) the feature dictionary that initially contains the variant and dependency entities and their descriptions from the provided variability model. This dictionary is maintained throughout the whole specify required variability step and ensures consistent meanings of features and quality attributes (see also domain analysis methods such as FODA [21]).
- The team then identify the functionality and quality attributes that are present in the domain in the timeframe specified by the required variability scope (for example, based on existing products, change requests, technology roadmaps, and market analysis). These are added to the feature dictionary. From this set, the assessment team identifies which functionality and quality attributes fit in the selected functionality and quality scope.
- The assessment teams subsequently identifies the product releases that are in the space and time scope (i.e., the product scenarios). An overview of these scenarios is drawn in a similar fashion to the product portfolio evolution diagram in Fig. 4.
- Finally, for each product identified above, the team selects a combination of functionality and quality that is likely to be needed for the product scenarios, and describes this scenario in the Product scenario description document. In order to make sense during the assessment, the quality attributes must be measurable at some point in the product lifecycle. Required quality can be required range, value, or a minimum/maximum. They also identify the variation within each product scenarios: the type of variation, and the earliest and latest time each variant should be bound (e.g., compile-time or runtime).

Note that depending on how much information is already available, one or more results of this construction process may already be part of the input of this activity (see also Activity 3a).

5.4.2.1. Dacolian case study. The number of product scenarios that would have to be created during this activity was estimated to be approximately 500 (based on featured products, requested new products, and desired products). Rather than specifying each individual product scenario, the assessment team chose to specify a number of generic product scenarios, as well as a number of specific scenarios. The specific scenarios comprised of products configurations that were based on request from existing customers. The specific scenarios therefore consisted of the features of the customers previously requested, as well as the product specific changes for these existing customers.

In Table 11, we present two examples of required variability that are related to the samples of provided variability in Fig. 9. First, if we look at the image type variation point, Intrada ALPR should support more image file formats. One of the product scenarios requested the support for RAW Bayer (i.e., a RAW variant with alternating blue, green and red pixels). All input images of Intrada ALPR version 4.0 furthermore contained eight bits per channel per pixel. For customers from Texas (US) and Taiwan this was not suf-
Activity 3b: Construct product scenarios.

Product assessment team performs for each product scenario the sub-activities (see Fig. 13). To build up such a required variability model, the COVAMOF meta-model (see Section 1.1) is extended with information from the product scenarios from Activity 3b, and the internal variability requirements in the product scenario entity is created in COVAMOF-VS. This scenario specifies the product group it belongs to, and the time at which the product that is defined by this scenario was, or should be delivered.

The second example of required variability in the product scenarios involved constraints on the recognition performance of IntraData ALPR. Where in the scenarios for Law Enforcement the recognition rate and error rate should be at least 80% and at most 1%, respectively, for Tolling products, these values are 90% and 0.1%.

5.4.3. Activity 3c – construct model

Next, the assessment team builds up the Required variability model (see Table 12), by using the provided variability model from Activity 2e, the Scoped feature dictionary and product scenarios from Activity 3b, and the internal variability requirements in the required variability sources from Activity 3a. During this activity, the assessment team performs for each product scenario the sub-activities we describe below. Throughout these sub-activities we explain the meaning and usage of the new entities and relations in the meta-model (these new entities marked by the dashed lines in Fig. 13):

- **Create product scenario:** First, for each product scenario, a product scenario entity is created in COVAMOF-VS. This scenario specifies the product group it belongs to, and the time at which the product that is defined by this scenario was, or should be delivered.

- **Identify new variation points and variants:** For the features in the product scenario that are not already a variant at a variation point in the feature layer of the current required variability specification, a new variant has to be specified. If this new variant cannot be a variant of an existing variation point, a new variation point has to be created as well. These new entities are tagged as “New” in the required variability model (see also Fig. 13). Note that new variants and variation points required for multiple product scenarios only have to be created once in the model.

- **Mark variants, and specify values:** Each required feature then translates in a variant or value that has to be selected for a product scenario. To record these selections, product scenarios are configured using the COVAMOF-VS configuration assistant. Both the new and variation points and variants from the provided variability model can be configured.

- **Specify variation point attributes:** Based on the variability within the product scenarios, the earliest and latest required binding time, and required type per variation point have to be specified. COVAMOF-VS allows for specifying these attributes in the new required change entities. These entities contain a reference to the associated COVAMOF entity together with the property-name and the required value.

- **Specify required quality:** In COVAMOF, the important quality attributes are specified using dependency entities (see also Section 1.1). To specify any requirements on these quality attributes, required quality entities have to be created. These entities specify, for one dependency, the required value, range, maximum or minimum quality for the associated product scenario. If the quality attribute was not already part of the model, a new dependency has to be created first.

In addition to the individual product scenarios, also the internal variability requirements are integrated into the required variability

<table>
<thead>
<tr>
<th>Output name</th>
<th>Description</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roadmap from innovation</td>
<td>The roadmap from innovation is included as it contains, for example, new technical insights from of ongoing internal research, and scientific publications</td>
<td>Computer of System Architect 1</td>
</tr>
<tr>
<td>Product descriptions</td>
<td>The product descriptions contained descriptions of product specific changes, and therefore would identify changes that might qualify for inclusion in the product family</td>
<td>File Cabinet</td>
</tr>
<tr>
<td>External change requests</td>
<td>The external change requests originate from two sources, i.e., existing users and potential customers</td>
<td>Change request catalog</td>
</tr>
<tr>
<td>Internal change requests</td>
<td>The internal change requests originate from three sources, i.e., market analysis, the IntraData Systems family, and the product developers. Market analysis for the ALPR family analyzes the market for improvements that are required to maintain a competitive advantage. The IntraData Systems family serves as internal customer for the ALPR family. They are separated from external customers as external requests have an economic priority over internal requests. The requests from the product developers involve change requests that will improve the product derivation process. Change requests are translated to a uniform ontology, and mapped to a feature branch to cluster changes before entering the catalog. The change request catalog preserves the source, the priority of the source, and frequency in order to determine whether a request is required, desired, goodwill to existing customers or will increase sale.</td>
<td>Change request catalog</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output name</th>
<th>Representation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Feature dictionary</td>
<td>Table that contains the features and quality from the feature dictionary that are in the scope of the assessment</td>
<td>Activity 3c</td>
</tr>
<tr>
<td>Product scenario overview</td>
<td>A diagram that presents an overview of the scenarios that exist, or are perceived, within the time and space scopes</td>
<td>Activities 4d, 4e, 5b, and 5c</td>
</tr>
<tr>
<td>Product scenarios description</td>
<td>For each product scenario, a section in the product scenarios description document, with references to sources and terminology from the features overview. Each section describes where the product that is defined by the scenario is located in time and space, and least contains a description of the functionality, variation, and quality in the product</td>
<td>Activity 3c</td>
</tr>
</tbody>
</table>
model. These internal variability requirements originate from Activity 3a and are used to specify variability that is not necessarily of interest to the customers, but is required to improve the business processes such as product derivation and maintenance. These requirements often cross-cut the set of product scenarios, and are specified directly in the required variability model as required change entities that are grouped in a Direct changes entity (see also Fig. 13). These entities specify new and obsolete variation points and variants, type, binding time, mechanism, and closing time.

Note that the result of this activity is a model of the required variability in terms of the configuration of the provided variability with the addition of a delta: the required variability is specified in terms of provided variants that need to be bound in the individual product family members, plus a difference to the provided variability in terms of new and obsolete variation points and variants, and changes to the attributes of existing variation points.

5.4.3.1. Dacolian case study. Fig. 14 shows where the product scenarios from the previous activity required changes to samples from the provided variability model presented in Fig. 9. Dacolian accommodated the Image type variation point with the new required image formats identified in the previous activity (i.e., RAW Bayer and JPEG2000). Similarly, they extended the set of languages with the new Taiwan (RC) and Swiss (CH) variant entities, and extended the operating system variation point with the new variants for the four additional Linux distributions. Finally, they modeled the choice between using 8 or 12 bits per channel by introducing the new runtime variation point bits per channel together with its two variants 8 bits and 12 bits.

Note that, although they expected...
a relation between this new variation point and the error rate, this relation is not specified during this activity. Instead, the presence of such association is determined during the next step, where the assessment team develops solutions to realize these new variation points.

5.5. Step 4 – variability evaluation

The fourth step in the assessment is the variability evaluation. The purpose of this step is to determine the impact of changes that are required to solve mismatches between the provided and required variability. The assessment team starts this step by identifying the mismatches (Activities 4a and b), which are clustered into impact analysis sets (Activity 4c). Impact analysis sets consist of mismatches for which the solutions are related. For each of the impact analysis sets, they then devise several solutions (Activity 4d). We refer to these as solution scenarios. Finally, the team determines the impact of each of these solution scenarios (Activity 4e).

5.5.1. Activity 4a – identify direct mismatches

The purpose of the first activity in this step is to identify the direct mismatches in the required variability model the assessment team created in Activity 3c. These mismatches correspond to the additional entities and required property changes that are explicitly specified in this model. The mismatches do not require propagating choices through the variability model. Direct mismatches that originate from the product scenarios are new variation points and variants. Direct mismatches that originate from internal requirements are obsolete variation points and variants, mismatches between binding time, closing time, type, and mechanism (Table 13).

5.5.2. Activity 4b – identify indirect mismatches

Where the previous activity involved the identification of the straightforward, direct mismatches, the purpose of this activity is to identify indirect mismatches, i.e., the mismatches that require propagating choices through the variability model. This activity breaks down in a number of separate actions that are supported by the COVAMOF-VS inference and validation engine:

- The assessment team has to trace Realization Relations between variation points to identify how choices at the feature level translate to choices at architecture and component levels. From

<table>
<thead>
<tr>
<th>Output name</th>
<th>Representation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mismatches</td>
<td>Table that identifies and describes the direct mismatches, and for each mismatch the products that are responsible for the mismatch. This can be directly generated from the required variability model</td>
<td>Activity 4c</td>
</tr>
</tbody>
</table>

Table 13: Output of Step 4

<table>
<thead>
<tr>
<th>Output name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Direct mismatches</td>
<td>Bits per channel: Nonexistent variation point Product L5, Product H5.</td>
</tr>
<tr>
<td></td>
<td>8bits: Nonexistent variant Product L5</td>
</tr>
<tr>
<td></td>
<td>12bits: Nonexistent variant Product H5</td>
</tr>
<tr>
<td></td>
<td>CH: Nonexistent variant Product H5</td>
</tr>
</tbody>
</table>

Table 14: Samples from the output of Activity 4a in the Dacolian case study

Fig. 13. Meta-model of the COVAMOF required variability model. The entities related to required variability are marked by the dashed lines.

Fig. 14. A sample from the required variability model from the Dacolian B.V. case study.
the realization rules at these Realization Relations, the assessment team (or in this case, tool) can deduce the mismatch where no variant is implemented for a certain combination of variants at a higher level. Note that as a result, required variants are identified at lower levels. These are added to the required variability model. This activity therefore causes iterations between the variability evaluation and specifying required variability steps (see Fig. 15).

- The team furthermore inspects dependencies to determine whether conflicts occur between variants, or whether required quality is met after propagating the choices. While conflicts can typically be directly calculated from the COVAMOF model, determining whether required quality is met is more complicated. Depending on how much knowledge is externalized in Step 2, the COVAMOF model contains reference data, associations, and dependency interaction entities that can assist engineers in estimating the dependency value (see also Section 1.1). If there is not enough information in the model yet, this activity can also cause iterations between the evaluation and specifying provided variability steps (see also Fig. 15).

- Finally, the assessment team inspects whether all product scenarios use the same variant at variation points, to identify potentially obsolete variation points (in contrast to variation points that are required to be obsolete).

### 5.5.2.1. Dacolian case study

The identification of indirect mismatches primarily resulted in mismatches between provided and required quality for the combinations of country variants in the product scenarios. For a number of these product scenarios, the analysis was already available. These analyses were performed during the lifecycle of the previous release, when potential customers required a particular product for which the quality was not met. If it would have been met, these products would already have been sold. Now, they ended up in a product scenario (Table 15).

Whether the quality could be met for the remaining five product scenarios was determined by testing a configuration that approximated the product scenarios, and assessing the product family architecture. Obviously, this approximation did not incorporate new required features. The test and architecture assessment revealed a number of mismatches. For example, for the scenarios that required multiple country variants, it would not be possible to derive a product from the product family, where the error-rate was below the required 0.1% with a recognition rate of 90%. In addition, the maximum processing time could also not be achieved for those product scenarios. For a sample of the output of this activity, see Table 16.

### 5.5.3. Activity 4c – cluster and prioritize mismatches

The results of Activities 4a and b are lists of mismatches, i.e., the direct mismatches and indirect mismatches. The goal of Step 4 is to find solutions for these mismatches. On the one hand, the task of developing one big solution for the whole set of mismatches would become too complex for the team. On the other hand, solving all mismatches separately is not possible either, as the solutions for the mismatches may influence or even conflict with each other. We refer to mismatches whose solutions will probably influence each other as interacting mismatches.

The goal of this activity is to find a trade-off between the approaches presented above, by clustering the set of mismatches into separate disjunctive subsets, called impact analysis sets (see Table 17). More precise, the clustering strives for creating impact analysis sets, where mismatches in one impact analysis set can interact with each other, but mismatches in separate sets do not. To determine whether mismatches interact, the assessment team uses the following pointers:

---

**Table 15**

<table>
<thead>
<tr>
<th>Output name</th>
<th>Representation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect mismatches</td>
<td>Table that identifies and describes the indirect mismatches, and for each mismatch the products that are responsible for the mismatch. This can be generated by propagating choices through the variability model</td>
<td>Activity 4c</td>
</tr>
</tbody>
</table>

**Table 16**

<table>
<thead>
<tr>
<th>Output name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Indirect mismatches</td>
<td>Mismatch Type Product scenarios</td>
</tr>
<tr>
<td>Error rate of &lt;0.1% cannot be obtained with 90% recognition rate for the product scenarios that require multiple country variants in one product</td>
<td>Required quality Product scenario L5, H5, ...</td>
</tr>
<tr>
<td>Processing time cannot be met when a product scenario requires more than one country variant</td>
<td>Required quality Product scenario L5, H5, ...</td>
</tr>
</tbody>
</table>

---

Fig. 15. Variability evaluation.
5.5.3.1. Dacolian case study. The mismatches from the previous two activities were clustered into 12 impact analysis sets (IAS). Based on the variation points that were involved, and the impact of solutions on other solutions, the assessment team first clustered the mismatches related to error-rate, the required image types and bit-levels, plate styles, obsolete variation points, processing time, and country variant mismatches in different impact analysis sets. The country variant IAS was further split in four impact analysis sets (such as Arabic, North-American, or Western European), and thus require different developers to address these mismatches.

Table 18 shows a sample from the output of this activity.

<table>
<thead>
<tr>
<th>Impact analysis sets</th>
<th>Id</th>
<th>Priority</th>
<th>Mismatches</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAS1</td>
<td>1</td>
<td></td>
<td>Error-rate too high for multiple country variants</td>
<td>The solutions we propose for these mismatches will most likely have an affect on almost all solutions we propose for a few of the other Impact analysis sets</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>IAS2</td>
<td>4</td>
<td></td>
<td>Bits per channel, 8 bit, 12 bit</td>
<td>We expect this IAS can be solved independent from the other sets</td>
</tr>
<tr>
<td>IAS9</td>
<td>4</td>
<td></td>
<td>New Image input JPEG2000</td>
<td>We expect this IAS can be solved independent from the other sets</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

5.5.4. Activity 4d – devise a set of solution scenarios. In most cases, several solutions exist for accommodating the variability mismatches in the impact analysis sets. In COSVAM, possible solutions are called solution scenarios. Important differences in these solutions scenarios are, for example, the decision to solve mismatches product specifically, or to solve them in the product family. Other important differences are the binding time and realization mechanism that are chosen, as well as differences in terms of dependency values that can be achieved or dependencies that are removed or created. The purpose of this step is therefore to identify, for each impact analysis set, the affected entities (product family artifacts, and entities from the variability model) under different solution scenarios for the IAS, and to determine the necessary changes on these entities (see Table 19).

As with most design processes, this activity is highly dependent on expert knowledge and creativity of architects and designers. It can be supported by several techniques, however. Source code analysis, for example, can be used to determine affected entities when replacing stable with new components, or changing variability realization mechanisms. In addition, several examples of realizations of variability have appeared, such as Schnieders and Puhlmann [28]. In addition, Svahnberg et al. [29], and van Gurp and Savolainen [18], present a pattern-like catalog of variability mechanisms, with the intent, motivation, solution, constraints, consequences, and examples. Fritsch et al. [16] on the other hand, present a process with which an organization can create their own catalog of variability mechanisms and their qualities.

Next to the existing body of knowledge, also the variability models that are produced in earlier steps of COSVAM can be used during this activity. For variant mismatches at existing variation points or binding time mismatches, for example, the provided variability model can be used to trace Realization Relations to identify the locations where the variation point is implemented, as well as which realization mechanisms are used. Dependencies in the provided variability model are furthermore used to identify which parts of the product family in the abstraction layer and subsystem scope are affected by a change.

Table 19 shows a sample from the output of this activity.

<table>
<thead>
<tr>
<th>Output name</th>
<th>Representation</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Solution scenarios</td>
<td>A set of design documents for each impact analysis set. Each design document in the set is identifiable, and specifically focused on a particular solution</td>
<td>Activity 4e</td>
</tr>
</tbody>
</table>

Note that due to, for example, dependencies that span multiple variation Points, it may not be possible to construct totally independent impact analysis sets. After the assessment team has finished clustering, they therefore prioritize the impact analysis sets according to the likelihood that the solutions that are proposed for the analysis sets, affect solutions for the other sets.

Also note that clustering and prioritization may need to continue during the evaluation, when new relations are discovered and created between variation points. Impact Analysis sets may furthermore be combined or split if, during the evaluation, it turns out they are, respectively, more or less related than initially expected.

5.5.4.1. Dacolian case study. For almost all impact analysis sets created during the previous activity, the assessment team designed three or four solutions. Most time, about 3 weeks, was spent on investigating different solutions for the IAS with priority 1, i.e.,
error-rate. Only for the Impact Analysis with the lowest priority two solutions were proposed, as the assessment team did not have time to explore other solutions. A sample from the output of this activity is depicted in Table 20. It describes the solutions proposed for impact analysis set 7, 8, and 9 (see also Table 18). The technical details of these solutions are left out of this paper.

### 5.5.5. Activity 4e – determine the impact of solution scenarios

The solution scenarios specified in Activity 4d imply changes to the product family artifacts. These changes have an impact in terms of the persons or business units that are involved in applying the changes, as well as the effort and time-to-market of the product scenarios. The goal of this activity is to determine these impacts for all solution scenarios, so that they can be used to decide which solution scenarios should be selected in the next step (see Table 21).

For the development of the product family, the assessment team determines the product family artifacts that are affected by the changes, as well as the effort and time-to-market of the product scenarios. The objective of this activity is to determine these impacts for all solution scenarios, so that they can be used to decide which solution scenarios should be selected in the next step (see Table 21).

In addition to the product family artifacts themselves, the team specifies the impact on the product described by the product scenarios. To this purpose, the assessment team determines the time and effort that is required to derive the product, given a product family where a particular solution scenario has been applied. Where solutions scenarios that completely cover the functionality required in product only imply configuration effort, solutions scenarios that do not support the required functionality imply product specific adaptations or exclude the product completely.

Note that while COSVAM only prescribes affected entities, time, and effort as impact, the list can be extended with organization specific impacts, such as the impact on tools for validation and testing.

#### 5.5.5.1. Dacolian case study

For each of the solution scenarios, the assessment team created a table of impacts such as depicted in Table 22. This Table shows the impact for the solution scenarios of IAS7, 8, and 9 (see also Tables 18 and 20). This impact analysis set contained mismatches related to the image formats and bits per channel variants. In addition to effort, resources and affected artifacts, the assessment team determined the impact of solution scenarios on solution scenarios of other impact analysis sets. For the error-rate impact analysis set (IAS1), for example, they determined that, depending on the solution, the solution scenarios of the four country impact analysis sets (IAS3–6) were affected by a one person week effort.

### 5.6. Step 5 – interpret the evaluation results

The purpose of the interpretation is to draw conclusions based on the evaluation. As we explained in the introduction of this section, we limit the discussion in this paper to the situation in which the optimal solution scenarios need to be selected for release planning (see Activity 1a). As we shown in Fig. 16, in this final step of COSVAM, assessment team identifies relevant business drivers and constraints (Activity 5a), identifies pros and cons of different solutions (Activity 5b), and selects solutions (Activity 5c).

#### 5.6.1. Activity 5a – identify business goals and constraints

The interpretation starts with identifying aspects that have an influence on which solution should be chosen, e.g., with respect
to the expected results specified in Activity 1b. For identifying the optimal solution scenarios these are aspects that constrain possible solutions, or that help in achieving certain goals (Table 23). The goals and constraints can have a technical, economical, or organizational background, but usually involve a combination of all three. Examples of constraints are the size of development teams, and the maturity of technology. Examples of goals are reduction of testing effort, and improved performance, ease of maintenance, time-to-market, and market share. See, for example, Bosch [5], and Clements and Northrop [9].

5.6.1.1. Dacolian case study. This activity identified eight relevant goals and constraints; they are depicted in Table 24. The order in this Table suggests a weighing of the goals and constraints, viz. the Highest performance, development constraints, and large projects are the most important goals and constraints, while open variation points and own code are less important.

5.6.2. Activity 5b – identify pros and cons of the different solutions
A set of solutions scenarios that contains exactly one solution scenario for each of the impact analysis sets forms a possible outcome of the assessment. In COSVAM, these sets are referred to as solution sets. The purpose of this step is to find the solution set that optimizes the business goals and constraints identified in Activity 5a. This and the next activity address the following two problems. First, judging the optimum for a set of goals and constraints is a multi-objective decision problem, i.e., solutions will generally be good for one set of goals and constraints, but bad for the others. Second, determining the optimum for all possible solution sets suffers from combinatorial explosion, i.e., since each impact analysis sets has multiple solution scenarios, the amount of possible solution sets increases exponentially.

COSVAM deals with these problems in the following way. In this activity, each solution scenario is rated independently on each business goal and constraint identified in Activity 5a, instead of as combinations of solution scenarios (Table 25). This rating is, amongst others, based on the solution scenario impacts determined in Activity 4e. The rationale for first rating the solution scenarios independently, is that from the solution scenarios for each impact analysis sets, typically only a small amount of solution scenarios will sufficiently address the goals and constraints to be likely candidates for a solution set.

5.6.2.1. Dacolian case study. In the case study at Dacolian, the assessment team created an overview of pros and cons for the solution scenarios of all impact analysis sets (see Table 26). To this purpose, the assessment team either used an exact value or a scale ranging from -- to 0, to ++. The --- denotes a situation where the solution scenario negatively influences the goal or conflicts the constraint, while ++ denotes a situation where the solution
5.6.3. Activity 5c – select solutions

In this final activity, the overview of pros and cons identified in previous activity are weighed, and the solutions that optimize the constraints and business goals are selected. To this purpose, the assessment team first constructs candidate solution sets by selecting the solution scenario candidates for each impact analysis set. This selection is based on whether the solution scenarios have an acceptable score on the pros and cons. These solution scenario candidates represent sub-optimal solutions, however. Different combinations of solution scenarios from different impact analysis sets represent different overall pros and cons of solution sets. In addition, the combination of solution scenarios may introduce scheduling issues, and have a different impact on which product scenarios can be facilitated (which translates in a different amount of revenue).

Depending on the number of solution scenario candidates per impact analysis set, the assessment team can either determine the impact, scheduling, and pros and cons of all solution sets at once, or first select the solution scenarios that have the best score for their impact analysis set, and then iteratively determine the optimal solution set by exchanging solution scenarios in the solution set.

The result of this activity is a solution set that optimizes the business goals and constraints (see Activity 5a), a description of its impact on the product scenarios, and the motivation of the selection (Table 27). The assessment is now almost finished. The only remaining task is communicating the output according to the communication output of Activity 1b.

### 5.6.3.1. Dacolian case study

A sample of the output of the selection process at Dacolian is depicted in Table 28. This table describes the motivation for the selection of a number of solution scenarios from this selection, and the descriptions of the solution scenarios, Dacolian constructed the documents to communicate the results internally and externally. We discussed the communication of these results in Activity 1b.

#### 6. Evaluation

As described in the research methodology (Section 3), we asked the assessment leader of Dacolian for their experiences with COS-VAM eight and sixteen months after. Below, we first present the unfiltered strengths and weaknesses they identified, and then discuss how these experiences relate to the variability assessment issues of Section 2.1.
6.1. Experiences

Quoting the assessment leader: “The assessment took us 5 weeks, of which 3 weeks were spent on the analysis of solution scenarios that we did not select for the assessment outcome. In the end, however, we do feel that assessing our product family according to the COSVAM method was a good investment. The experiences we have on working with the new release show that we indeed predicted the required variability very well in Step 3, and that we chose the right solution scenarios to address them. The different alternatives we investigated are certainly not a waste of effort, as we feel they helped us in choosing a set of solutions that really addressed our business goals, while at the same time making sure that we could implement them within the organizational, economical, and technical constraints. When we evaluated our experiences with COSVAM, we identified five main strengths and one weakness”.

6.1.1. Strengths

- **Structure**: “In the original situation, we had no structured assessment process to evolve our Intrada ALPR product family to a new version. Instead, the assessment and the actual implementation of necessary changes were not clearly separated. The ad-hoc nature of our assessments often caused situations where we would implement changes and then simply run out of time. COSVAM really helped us in explicitly planning the assessment tasks, by breaking the them up in small, manageable pieces, and showing how different inputs and outputs could be used to be able to select the right set of solutions. As the outcome of COSVAM is furthermore structured in separate solutions for the impact analysis sets, we can easily plan the implementation of these smaller parts”.

- **Decisions and assumptions made explicit**: “Another benefit of COSVAM is that after the assessment, there is still a clear list of required variability that will not yet be implemented in the product family (such as some required country variants). For those aspects, we did have the solution scenarios prepared, tough. In the end, this allowed us to quickly implement some scenarios that were rejected based on scheduling issues, but that could be implemented when it turned out some developers required a bit less effort than estimated”.

- **Optimal solutions for actual problems**: “We used to identify required variability in an ad-hoc manner. In addition, the decision to implement a particular change was mainly based on technical challenges we would like to tackle, because, honestly, those were the changes the developers thought to be fun. With the instructions provided by COSVAM, we were forced to, and assisted in making decisions that represent value to our business, rather than our personal preferences.”

- **Maturity**: “Based on the instructions we received on the method, we were confident enough to apply the method by ourselves. After completing the assessment, we still think COSVAM is a mature and complete technique that makes sure no necessary steps are forgotten, and that clearly specifies the required inputs and outputs”.

- **COVAMOF**: “The final main strength of COSVAM is its approach to modeling variability. In the past, experiences with having no variability model frequently resulted in situations where mismatches were not found. Having such a model in place makes it a lot easier to identify variability mismatches, as it helps to keep an overview of all choices and dependencies. Another benefit of using the COVAMOF externalization process is that taking the information from different sources revealed that different experts actually had different insights into the dependencies between choices, and that some experts had assumptions about the provided variability that turned out to be wrong. Creating such a model thus improved our common understanding of the provided variability of the Intrada ALPR product family”.

6.1.2. Weaknesses

- **Focused on larger organizations**: “When we look at COSVAM as a whole, we think that Step 1 is now a bit superfluous for small organizations like Dacolian. While we do see the benefit in case an external assessment team is involved, in particular in situations where internal people perform the assessment, the identification of the assessment team, influencing factors, etc. in Step 1 feels a bit like overkill. We would assume these aspects to be known to all people involved in the assessment”.

“The evaluation thus concluded that experiences of applying COSVAM, as well as the experiences with its results, strengthen our decision to continue to apply this method for planning our product family releases. Sixteen months later, our opinions on COSVAM have not changed. As we still see the same benefits of this assessment technique, we now even changed our working methods to collect the information needed for the assessment in a much more structured way throughout development.”

6.2. Discussion

The examples and experiences presented in this paper are related to the variability assessment issues we discussed in Section 2.1. Below, we discuss how the examples and experiences cover these issues.
6.2.1. Methodological issues

- **Unstructured**: The assessment team concluded that COSVAM divides the assessment problem into manageable pieces, where the processes and their results can be carefully planned (see strength: structure).

- **Reactive instead of proactive**: The experiences of Dacolian showed that by taking future products into account, the assessment team was able to create a product family release that turned out to match the requirements of the products that had been built on top of that release (see, for example, strength: optimal solutions for actual problems).

- **Generalized decisions**: A few months after the assessment, the experiences at Dacolian showed that, now, solution scenarios were selected that actually represent value to their business (see, for example, strength: optimal solutions for actual problems).

- **Lack of removing obsolete variability**: The assessment team was able to identify two obsolete variation points at the feature level, and to make an educated decision whether to remove them (see, for example, Activity 4b).

6.2.2. Knowledge issues

- **Addressing only one layer of abstraction**: The assessment at Dacolian encompassed an assessment on all layers of abstraction (see Activity 1c). This allowed the assessment team to identify detailed changes that were required to the variability provided by the product family artifacts.

- **Implicit variability**: The experiences at Dacolian show that even the complete model that was constructed during the limited time of the assessment, created a better understanding of the dependencies between variation points (see strength: COVAMOF).

- **Neglecting implementation dependencies between features**: The implementation dependencies that were captured by the provided variability model allowed the assessment team to cluster the mismatches, so that they could explicitly relate the impact of solution scenarios to each other (see, for example, the impact of the error-rate impact analysis set in Activity 4e).

- **Insufficiently exploring alternative solutions**: Finally, the assessment concluded that explicitly considering multiple solution scenarios helped them in choosing a set of solutions that really addressed their business goals (see Section 5.5.1).

Note that this evaluation does not allow us to verify whether the way Dacolian used to work would actually have produced better results. An important aspect of the validity of our conclusions, however, lays in the fact that Dacolian still uses COSVAM sixteen months later. This shows that COSVAM not only addresses the variability assessment issues, but also that it is applicable in practice.

7. Conclusion

The predominant challenge, in most software product families, is the management of the variability required to facilitate the product differences. Over time, the variability required from the product family evolves, but assessing how the variability provided by the product family artifacts needs to evolve in response, is a non-trivial problem. Currently, however, there is a lack of techniques that specifically support the assessment of software variability in product family artifacts. In response to this, we have developed COSVAM, the COVAMOF software variability assessment method.

In this paper, we described and illustrated each of the five steps of COSVAM, i.e., identify assessment goal, specify provided variability, specify required variability, evaluate variability, and interpret the assessment results. While we presented the main ideas behind COSVAM in earlier work [13,14], this paper discusses the different steps of COSVAM and the case study in detail. In addition, this paper presents the experiences of applying COSVAM in an industrial setting. We conclude our paper with a discussion on the contribution and future work of COSVAM.

7.1. Contribution

The main contribution of COSVAM is that it is the first technique for assessing variability with respect to the needs of a set of product scenarios. The five steps of COSVAM form a technique that can be tuned to address a variety of situations where the question of whether, how and when to evolve variability is applicable. It defines different results that can be produced, such as an overview of mismatches, different solution scenarios and a selection of the optimal solutions, as well as defines the way in which information has to be selected and interpreted to produce those results. It furthermore provides a means to externalize the provided variability of the product family.

The activities of the five steps of COSVAM address the issues we presented in Section 3 as follows:

- **Unstructured**: COSVAM addresses this issue by distinguishing multiple goals, defining repeatable steps and activities, and forcing assumptions and decisions to be made explicit.

- **Reactive instead of proactive**: One of the main ideas behind COSVAM is that, for evolving the variability of the product family artifacts, future product specifications are incorporated in the product scenario set (see Activities 3a and b). COSVAM is therefore not only suitable as reactive assessment for benchmarking, identifying mismatches, and determining product specific adaptations, but also as proactive assessment for release planning.

- **Generalized decisions**: Decisions with respect to mismatches between the required and provided variability are grouped in impact analysis sets, i.e., sets of mismatches that are related as they involve the same component, for example. For each impact analysis set, multiple solution strategies are devised that each have a different impact (see Step 4). The interpretation step (Step 5) then identifies the pros and cons of those solution scenarios, with respect to the goals and constraints of the product family. This allows the selection of set of solution scenarios that addresses the requirements posed on the product family.

- **Lack of removing obsolete variability**: The COSVAM is a need-based assessment method that compares the variation points, variants, and quality attributes that are required within a set of product scenarios with the variability provided by the product family. Obsolete variability is identified indirectly when none of the product scenarios requires particular variation points, variants or quality, or when they are directly identified by the experts (see Activities 3c, 4a, and 4b). Obsolete variability is marked as potentially obsolete, and during the evaluation and interpretation steps it is decided whether the obsolete variability will be removed, or whether the removal will be postponed.

- **Addressing only one layer of abstraction**: The COSVAM is built around a notion of variability that treats variation points and dependencies uniformly as first-class citizens that are related across abstraction layers (see Step 2). The COSVAM is aimed at all abstraction layers, but when required, allows for focusing on a subset of layers and artifacts by selecting the appropriate assessment scope. To minimize the inaccuracy and risks associated with a limited scope, the assessment allows for iterations between different steps (Steps 2–4), and weighs risks in the interpretation step (Step 5).
• Implicit variability: COSVAM provides an iterative process that is focused on minimizing effort with respect to externalizing variability that is irrelevant for the assessment (see Step 2). COSVAM explicitly deals with missing information by forcing the documentation of which parts are deliberately left implicit and weighing missing information in the interpretation step (Step 5). COSVAM furthermore provides a step-wise process for externalizing provided variability to a variability specification that models variability evolution.

• Neglecting implementation dependencies between features: During the evaluation (Step 4), it is verified whether the dependencies in the provided variability allow for offering the required combinations, how different solutions address these implementation dependencies, and how solutions have an impact on each other.

• Insufficiently exploring alternative solutions: The COSVAM explicitly deals with this issue by assisting engineers in devising multiple solution scenarios, and weighing the pros and cons of these solution scenarios (see also above: generalized decisions).

With the case study presented in this paper, we showed that COSVAM indeed addresses the variability assessment issues presented in Section 2.1. In other words, we showed that COSVAM represents a practical and structured technique to answer the question of whether, how, and when variability should evolve.

7.2. Future work

There are always points where a method can be improved or extended. The situation is not different for COSVAM. Below, we discuss a number of points our method can be improved:

First, a number of steps in this process were performed manually, while some could have been supported by tools. While our modeling tool COVAMOF-VS is capable of modeling variability, and finding variability mismatches, the COVAMOF-VS tool requires a number of extensions, such as maintaining the mismatch lists, and impacts of solutions scenarios, before our assessment method is fully integrated in the COVAMOF-VS tool-suite.

In addition, we intend to investigate and incorporate existing quantitative and qualitative models for weighing the pros and cons of different solutions scenarios. With a stronger connection between our method and other of techniques, we think we can achieve a situation where also the weighing, scheduling, and selection of solutions can be supported by tools.

So far, we have also only briefly discussed the inaccuracy of provided variability. In case the assessment involves future products, e.g. during release planning, the required variability is also subject to inaccuracy, however. Predicted features may not be needed, for example, or new products may require a different set of features than predicted. We are currently extending the COSVAM to deal with inaccuracy of predictions, as well as ways to quantify the inaccuracy of provided and required variability.

Finally, although the technique has been applied on one case, i.e. the Intrada ALPR product family of Dacolian B.V., it requires several additional cases in the future. That way, we can further validate COSVAM, and extend it with a body of best practices, for example on the construction of product scenarios, the required level of detail of impacts of solution scenarios, a ranking of realization mechanisms with respect to common mismatches, and a collection of important goals and constraints with sensible weights.

Despite the benefits already experienced by our case study organization, COSVAM thus represents a first large step into an area where many research challenges still remain.

Acknowledgement

We thank the participants from Dacolian for their valuable input.

References


