Software Product Family Evaluation

Frank van der Linden,[1] Jan Bosch,[2] Erik Kamsties,[3]
Kari Känsälä,[4] and Henk Obbink[5]

[1]Philips Medical Systems,
frank.van.der.linden@philips.com
[2]University of Groningen,
Jan.Bosch@cs.rug.nl
[3]University of Duisburg-Essen,
kamsties@sse.uni-essen.de
[4]Nokia Research Center,
Kari.Kansala@nokia.com
[5]Philips Research Laboratories
henk.obbink@philips.com

Abstract. This paper proposes a four-dimensional evaluation framework for software product family engineering. The four dimensions relate to the software engineering concerns of business, architecture, organisation, and process. The evaluation framework is intended for use within software developing organisations to determine the status of their own software product family engineering and the priorities for improving. The results of the evaluation can be used for benchmarking, roadmapping, and developing improvement plans. An initial evaluation of a real industrial case is presented to show the validity of the framework.

1 Introduction

The main arguments for introducing software product family engineering are to increase productivity, improve predictability, decrease the time to market, and increase quality (dependability). To improve the overall system family engineering capability in Europe, a series of Information Technology for European Advancement (ITEA) projects on this topic is being executed, namely if99005 ESAPS (1999-2001), if00004 CAFÉ (2001-2003), and if02009 FAMILIES (2003-2005) 7. We are involved in all or some of these projects. The initial evaluation framework for software product family engineering was prepared during the ITEA project called From Concepts to Application in System-Family Engineering (CAFÉ) 9. In this paper, we provide an improved framework as well as a more extensive explanation of the different development concerns in relation to software family engineering. Within the recently started FAMILIES project, an improved and refined version will be produced and tested. The framework has been developed based on experiences with a wide variety of software product families and is illustrated in this paper using a case within Philips Medical Systems.

The focus of this paper is on embedded systems. Software product family engineering originated from embedded systems development, where product families already existed. Software in embedded systems was introduced originally to improve the flexibility and later for the introduction of more functionality. When the amount of software was growing, the family engineering approach had to be applied on software as well. As software behaves differently than other product assets, it was initially not clear how to deal with it. In this paper, we look at all the concerns around *software* product family engineering. Below, we often denote products built by software product family engineering. We refer then to embedded systems that have software inside produced in a family-wise way. Our approach works, however, also for pure software systems, but we have not seen many examples of such systems.

The remainder of the paper is organised a follows. In the next section, the Business, Architecture, Process, Organisation (BAPO) model is introduced. BAPO is used as the basis of the evaluation framework. Subsequently, the four evaluation dimensions are discussed. Then, we give a validation example, discuss related work, and finally, provide our conclusions.

2 The Business, Architecture, Process, Organisation (BAPO) Model

Within the ITEA project called Engineering Software Architectures, Processes, and Platforms for system family Engineering (ESAPS), we identified four interdependent software development concerns, BAPO 1:

1. **B**usiness, how to make profit from your products
2. **A**rchitecture, technical means to build the software
3. **P**rocess, roles, responsibilities, and relationships within software development
4. **O**rganisation, actual mapping of roles and responsibilities to organisational structures

Figure 1 gives an overview of the BAPO concerns. Links denote the interrelationships between the concerns that exist between all pairs. In principle, applying changes in one concern is though, because it induces changes in the others. Arrows denote a natural order to traverse the concerns, giving an order to the acronym as well. The Business is the most influential factor: it has to be set up right in the first place. The architecture reflects the business concerns in software structure and rules. The process is set up to be able to build the products determined by the architecture. Finally, the organisation should host the process.

Through the clarification of these dimensions, the ESAPS, CAFÉ, and FAMILIES projects consider all the concerns in the context of software product family engineering. In fact, although architecture is an important topic for family engineering, the process had a much larger emphasis in these projects, since it was often neglected in earlier approaches. Due to the realisation that is it crucial for software family engineering to address the business and organisation well, effort was also directed to these dimensions, resulting in a more complete view of what is necessary for software product family engineering.

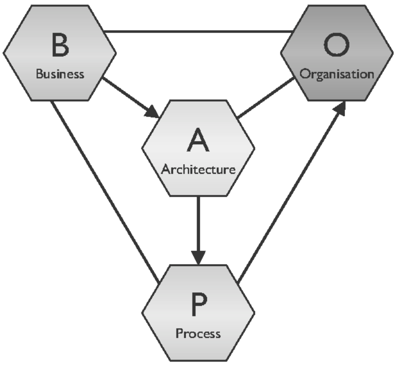

Fig. 1. The BAPO concerns

We will use this separation of concerns to provide four dimensions of the family evaluation framework. An organisation will have a separate evaluation level for each BAPO concern. The interdependence between those concerns becomes obvious as soon as one studies the effects of changes. Changes in one dimension virtually always have consequences for the other dimensions as well. In fact, actions to improve the evaluation result for one concern may give rise to a lower evaluation result for some of the others. Therefore, improvement actions have to consider all BAPO concerns.

Through BAPO, our evaluation framework collects and structures characteristics of a software production unit, division, or company that are proven by experience to be effective. It is based on the experience of the companies involved in the aforementioned ITEA projects, and, as such, it consolidates a large body of knowledge. The purpose of the model is to

1. serve as a benchmark for effective software product family engineering
2. support the assessments of software product family engineering for capability evaluations of software production units, divisions, or companies
3. support the improvement of software product family engineering, which involves producing assessments and improvements plans

The result of the evaluation (i.e., an evaluation profile) is the representation of a software product family engineering evaluation for an organisation, represented in four separate evaluation scales for Business, Architecture, Process, and Organisation evaluation.

Setting the target evaluation profiles for improving software product family engineering has to include the consideration of possible dependencies and tradeoffs between the evaluation dimensions.

In the next several sections, we describe the different evaluation levels for each BAPO concern. Subsequently, we describe the relationships and influences between these concerns. Finally, the evaluation framework is validated and illustrated using a case.

3 Business Dimension (BAPO-B)

The business evaluation dimension deals with the ability of an organisation to manage, predict, and steer the cost and profit of the development. The cost is dependent on the architecture process and organisation chosen. The profit relates to customer satisfaction and the market position.

For our purposes, the main concern lies in the way an organisation can determine the costs and payoffs of a software product family engineering approach. The organisation should be able to determine whether and how it should invest in software family engineering, for which products, and in which order. It should be able to understand and quantify the necessary investments in software family engineering. A well-organised organisation has only one mission and a well-defined set of objectives. The software family engineering should fit in them.

When a software family engineering approach is chosen, the business goals should be in alignment with the software family engineering goals in order to improve the competitiveness. One of the questions addressed by the business aspect is *scoping* 12, which deals with the question of what products should be subject to software family engineering and why, based on market and technology expectations.

In the evaluation framework, four main aspects are used in the evaluation of the business dimension. The selection of these aspects is a first guess, but they are related with important aspects recognised in business literature. These aspects are partially dependent of each other; often, a higher level for one aspect corresponds to higher levels for the others as well.

1. **Identity**: How well has the organisation formulated an identity relating to software family engineering? Low levels for this aspect correspond to software product family engineering that is not visible in the identity. At higher levels for this aspect, there has to be a strong relationship between the development of the products and the identity. For instance, the similarity between the products is visible at the marketing level and used in the communication of the management to the employees and customers.
2. **Vision**: How well does the organisation aim for a future where software product family engineering fits? At lower levels for this aspect, the software product family engineering is not present in the vision. At higher levels for this aspect, the presence of software family engineering is available in the determination of the future goals and communicated to the customers.
3. **Objectives**: How well does the organisation determine its future goals, aimed at marketing and selling what the software family engineering produces? At lower levels, no specific marketing for similar products based on reusable platforms is available. At higher levels, products in the family will be marketed as such, aiming to sell family members instead of one-of-a-kind systems.
4. **Strategic planning**: How well does the organisation plan the family business and development? At lower levels, no specific planning is available for the software family engineering. At higher levels, roadmaps of what will be developed are available and used.

Below, we present a five-level structure that is more refined than the one presented by van der Linden and associates 9. The levels are

1. **Reactive -** The business does not actively influence the software family engineering. Instead, it *reacts* to the situation.

2. **Awareness -** The business is *aware* of the software family engineering. However, it does not know the right instruments to influence and use it for the business's sake.
3. **Extrapolate -** The business is *extrapolating* on the results of the software family engineering. It influences software family engineering for obtaining business goals.
4. **Proactive -** The business is *proactively* planning and managing the software family engineering and the business goals to obtain the best business results out of it.
5. **Strategic -** The software family engineering is *a strategic* asset for reaching the business goals.

Below, we discuss each level in more detail

3.1 Level 1: Reactive

The business does not actively influence the software family engineering. Instead, it *reacts* to the situation.

1. **Identity**: implicit
2. **Vision**: short term (just cash flow)
3. **Objectives**: missing
4. **Strategic planning**: missing

This is the basic level for the business dimension. The identity of the organisation is derived from the activities it happens to perform. If there is software family engineering, it is not visual at the business level. There is no software family engineering vision. The objectives and business-planning process do not support software family engineering. The reuse of assets in product development is mainly for opportunistic reasons. Decisions about whether to make or buy assets, or obtain them differently, are taken only for opportunistic reasons. There is no strategy to align marketing to software family engineering. Products from the family are marketed just as any other product.

3.2 Level 2: Awareness

The business is *aware* of the software family engineering. However, it does not know the right instruments to influence and use it for the business's sake.

1. **Identity**: available
2. **Vision**: short to medium term
3. **Objectives**: partially, and qualitative
4. **Strategic planning**: ad hoc process

At this level, there is an awareness of the software product family engineering at the business level. The business sees the benefits of software family engineering for the short or medium term, but is unable to connect it to all the relevant objectives of the organisation. There is no strategic plan in relationship to software family engineering.

3.3 Level 3: Extrapolate

The business is *extrapolating* on the results of the software family engineering. It influences software family engineering for obtaining business goals.

1. **Identity**: identified
2. **Vision**: medium term
3. **Objectives**: qualitative
4. **Strategic planning**: ad hoc process

At this level, planning for the software product family engineering is available. Scoping is performed to determine the borders of the product range, and roadmaps are used to plan the software product family engineering and to decide on making assets or obtaining them differently. Often, the roadmaps are based on business and technological scenarios. On a regular basis, the roadmaps and scopes are updated.

There is an ad hoc strategy to align marketing to software family engineering. Products produced are predominantly marketed as any other product. However, the long-term vision of the marketing department is taken into account in scoping and family planning.

3.4 Level 4: Proactive

The business is *proactively* planning and managing the software family engineering and the business goals to obtain the best business results out of it.

1. **Identity**: communicated
2. **Vision**: medium/long term
3. **Objectives**: partially quantitative
4. **Strategic planning**: defined process

At this level, decisions are based on partial cost models. Scoping is based on expectations of the parameters of the software family engineering effort. Roadmaps are based on intra-company agreements, time-to-market estimations, and profit expectations. Scope definitions and roadmaps are maintained locally and communicated with other departments of the company. Expectations on resources, as well as key and core technologies for the company, influence decisions about whether to develop family assets, or to buy, mine, or commission assets.

There is a process available to align marketing to software family engineering. The marketing long-term vision is adapted according to the family planning.

3.5 Level 5: Strategic

The software family engineering is *a strategic* asset for reaching the business goals.

1. **Identity**: managed
2. **Vision**: long term
3. **Objectives**: quantitative
4. **Strategic planning**: institutionalised process

Once the business dimension has the highest maturity, decisions are based on quantitative cost models. Scoping is based on quantitative predictions of the return on investment of the software family engineering effort. Roadmaps are defined based on intra-company agreements, time-to-market estimations, and profit expectations. Scope definitions and roadmaps are aligned between the different units within a single company. Resource availability, key and core technology for the company, time-to-market expectations, and profit and cost models influence decisions about whether to develop family assets, or to buy, mine, or commission them.

Marketing aligns with software family engineering. The family is marketed as a whole, and the position of the product in the family is part of the marketing strategy. New products requiring little development effort are actively promoted. Products that involve expensive developments (e.g., due to architectural mismatches) are avoided.

4 Architecture Dimension (BAPO-A)

The architecture of software product families differs significantly from architecture in single-product development. It is essential to detect, design, and model the variable and common parts of the software family engineering (i.e., software variability is a key concern). Variability management needs to start during requirements engineering. Common parts will useably be implemented as a reusable platform, where the variable parts fit in at explicit variation points.

The development of software product families is aimed at a dynamic set of resulting products. Platform technologies typically evolve very rapidly. Because of this, systems in production cannot be re-implemented for each change in technology. In parallel, the quality of products within the family needs to be improved, and product variants have to be delivered in less time. This results in a need for software architectures that will enable us to move to new emerging platforms against a minimal investment.

The technological approach taken to the development of software products varies substantially between different organisations. From our experience, these approaches can be categorised in five levels 3, where the first level exhibits no sharing of software artefacts and the highest level requires no product-specific software development. The preferred approach for an organisation depends on the business goals that the organisation aims to achieve with the software family engineering, as well as the application domain of the software products and the maturity of the organisation in the process and organisation dimensions.

In our experience, one can identify four aspects that define the five approaches to the technology of the software product family engineering. As with the business dimension, the aspects are partially dependent on each other. A higher level for one aspect often goes together with higher levels for the others as well. Below, these aspects are discussed in more detail:

1. **Software product family architecture**: The software product family architecture can exist at several levels. The level determines what is and is not shared among the produced products. At higher levels, the influence of the architecture on the development increases. Higher levels of software product family architecture do not mean larger amounts of shared software between the products. At low levels, the software product family architecture may only make a distinction between infrastructure and product-specific components. At higher levels, the software architecture is enforced.

2. **Product quality**: The quality of the set of products as a whole is, at lower levels, typically accidental as all the attention is towards providing the right functionality. With increasing levels, quality is managed more and more explicitly within the architecture.
3. **Reuse levels**: The reuse level indicates the amount of relative effort spent on producing shared, reusable assets, when compared to application or product engineering.
4. **Software variability management**: At lower levels, the management of the software variability is focusing mainly on supporting compile-time and link-time binding. At higher levels, the complete software life cycle is taken into account, determining when to introduce and bind which variability, and which mechanism should be used.

We have defined the following level structure. Below, we discuss the levels in more detail:

1. **Independent product development –** There is no software family engineering. Instead, *products are developed independently.*
2. **Standardised infrastructure –** The family architecture focuses itself on the *standardisation of the infrastructure*.
3. **Software platform –** The family architecture defines a *software platform* to be used as the basis of product development.
4. **Variant products –** The family architecture determines the construction of *variant products*.
5. **Self-configurable products –** The family architecture defines pervasive rules that enable the automatic selection of assets to *configure products*.

4.1 Level 1: Independent Product Development

There is no software family engineering. Instead, *products are developed independently.*

1. **Software product family architecture**: not established
2. **Product quality**: ignored or managed in an ad hoc fashion
3. **Reuse level**: Although ad hoc reuse may occur, there is no institutionalised reuse.
4. **Software variability management**: absent

An organisation developing products independently has no sharing of external or internal software artefacts. The commonality between products is not exploited.

4.2 Level 2: Standardised Infrastructure

The family architecture focuses itself on the *standardisation of the infrastructure*.

1. **Software product family architecture**: specified external components
2. **Product quality**: The infrastructure supports certain qualities; for the remaining qualities, an overengineering approach is used.
3. **Reuse level**: only external components
4. **Software variability management**: limited variation points from the infrastructure components

The first step that an organisation typically takes when evolving towards exploiting commonality in its products is to standardise the infrastructure on which product development is based. This infrastructure typically consists of the operating system and the usual commodity components on top of it, such as a database management system and a graphical user interface. In addition, the organisation may acquire some domain-specific components from external sources. Typically, these components are integrated through some proprietary glue code.

4.3 Level 3: Software Platform

The family architecture defines a *software platform* to be used as basis for the development of the products.

1. **Software product family architecture**: Only the features common to all products are captured.
2. **Product quality**: inherited from the platform
3. **Reuse level**: reuse of internal platform components
4. **Software variability management**: managed at the platform level

As a first step in achieving the intra-organisational reuse of software artefacts, the organisation may develop, maintain, and evolve a platform on which the creation of the products or applications is based. A platform typically includes a standardised infrastructure as a basis (as discussed in the previous section) that typically contains generic functionality. On top of that, the platform captures domain functionality that is common to all products or applications. The common functionality that is not provided by the infrastructure is implemented by the organisation itself, but typically, the application development treats the platform as if it were an externally bought infrastructure.

4.4 Level 4: Variant Products

The family architecture determines the construction of *variant products*.

1. **Software product family architecture**: fully specified
2. **Product quality**: a key priority for development
3. **Reuse level**: managed
4. **Software variability management**: many variation points and dependencies between them

Once the benefits of exploiting the commonalities between the products become more accepted within the organisation, a consequent development may be to increase the amount of functionality in the platform to the level where functionality common to several but not all of the products becomes part of the shared artefacts. Now we have reached the stage of variant products. Variation is managed strictly, and functionality specific to one or a few products is still developed as part of the product derivation. Functionality shared by a sufficient number of products is part of the shared artefacts, with the consequence that individual products may sacrifice resource efficiency for development effort offered by the software family engineering. In addition, all products are developed based on the defined family architecture. In particular, it specifies how and when to configure variants.

4.5 Level 5: Self-Configurable Products

The family architecture defines pervasive rules that enable the automatic selection of assets to *configure products*.

1. **Software product family architecture**: enforced
2. **Product quality**: Quality attributes are implemented as variation points in the architecture and components.
3. **Reuse level**: automatic generation of software family engineering members
4. **Software variability management**: The automated selection and verification of variants at variation points have been optimised.

Especially if the organisation develops products in relatively stable domains and derives many product instances, there is a tendency to further develop the support for systematic product derivation. The consequence is that the architecture is enforced for all the products, and derivation is performed through the application of the defined rules. Large parts of the product derivation can be automated and/or performed at the customer's site.

5 Process Dimension (BAPO-P)

The process dimension emphasizes the process, roles, work products, and corresponding responsibilities and relationships within software development. The Capability Maturity Model for Software (SW-CMM) and its current successor Capability Maturity Model Integration (CMMI) 5, 6 (both developed by the Software Engineering Institute [SEI]) are the de facto standards in assessing and evaluating software processes. For that reason, CMMI is the most natural choice to be the basis of the evaluation approach for software product family engineering in the BAPO-P dimension. Of course, the practices for software family engineering are more extensive and more special, which is a basis for further refinement. Many of the ITEA project's best practices 8 and the work of the SEI on software product line engineering 4 have to be considered as input. The levels of the CMMI (Staged Representation) are introduced briefly using the original text from the SEI's technical report 6 in the following by using the following aspects to depict their differences:

1. **Predictability**: How predictable is software development at each level?
2. **Repeatability**: How repeatable is the development process at each level?
3. **Quantifiability**: How quantifiable is software development?

The staged representation of the CMMI 6 offers its maturity levels to be used also in the software product family engineering context. We do not go into details here, because we just follow the CMMI terminology. For software family engineering, the different stages will have different focus areas and/or basic practices.

1. **Initial** – There is no managed and stable process available. Development proceeds in an ad hoc way.
2. **Managed** – There are planned processes available.
3. **Defined** – Processes adhere to standards.
4. **Quantitatively managed** – Processes are quantitatively tracked, and improved.
5. **Optimising** – Processes are continuously improved based on quantitative data.

5.1 Level 1: Initial

There is no managed and stable process available. Development proceeds in an ad hoc way.

1. **Predictability**: unpredictable
2. **Repeatability**: not repeatable at all (i.e., there is no related learning in the organisation)
3. **Quantifiability**: No data is available about past projects.

5.2 Level 2: Managed

There are planned processes available.

1. **Predictability**: tolerably
2. **Repeatability**: Good practices can be applied, and bad practices can be avoided.
3. **Quantifiability**: Data is available on past projects.

Planned processes are available guiding the work of the development. Their execution is measured to determine problems in time. No explicit tracking and learning are available. For most CMMI-identified practices at this level, there will be specific consequences for software product family engineering. Many of them stem from the interconnection of different processes and the distinction between common and variable software.

5.3 Level 3: Defined

Processes adhere to standards.

1. **Predictability**: satisfactorily
2. **Repeatability**: The process is tailored from the organisational software process.
3. **Quantifiability**: Data on past projects is available and analysed to be more effectively used by future projects.

Processes adhere to standards that are improved over time. Process measurements are used for process improvement. For most CMMI-identified practices at this level, there will be specific consequences for software product family engineering. Many of them stem from the interconnection and the continuity of different processes and the use of common software over project boundaries.

5.4 Level 4: Quantitatively Managed

Processes are quantitatively tracked and improved.

1. **Predictability**: very predictable
2. **Repeatability**: The process is tailored from the organisational software process with corresponding quantifiable data.
3. **Quantifiability**: Software development data from past projects have been packaged into quantified models to be used to estimate and predict future projects.

Processes are quantitatively tracked and improved. The organisation adheres to these processes. It is not clear whether software product family engineering has much impact at this level. It may well be that just doing software product family engineering eases the execution of this level.

5.5 Level 5: Optimising

Processes are continuously improved based on quantitative data.

1. **Predictability**: extremely predictable
2. **Repeatability**: fully repeatable given the commonality and variability between past and new projects
3. **Quantifiability**: The process of new projects can be optimised based on the data and corresponding analysis of past projects.

Processes are improved continuously based on a quantitative understanding of the common causes of variation inherent in processes. Just as for level 4, it is not clear whether software product family engineering has much impact at this level. It may well be that just doing software product family engineering eases the execution of this level.

6 Organisational Dimension (BAPO-O)

The organisational dimension deals with the way the organisation is able to deal with complex relationships and many responsibilities. This dimension is refined with respect to the work of van der Linden and associates 9 and combined with the taxonomy presented by Bosch 2.

Software family engineering results in a separation of activities, often over organisational borders. In addition to software assets, other artefacts such as development plans and roadmaps are shared over these borders. This means that the organisation should have individuals or teams that are responsible for the interaction of the shared artefacts between the different parties involved in the development.

Despite the emergence of a variety of technological solutions aimed at reducing the effects of geographical location, the physical location of the staff involved in the software product family engineering still plays a role. It is simply more difficult to maintain effective and efficient communication channels between teams in disparate locations and, perhaps even, time zones, than it is between collocated teams. Therefore, units that need to exchange much information should preferably be located closer to each other than units that can cooperate with less information. The following aspects can be identified that influence the evaluation in the organisational dimension:

1. **Geographic distribution**: How complex is the geographic distribution of the software family engineering organisation: local projects, and departments, company-wide or even over company borders.
2. **Culture**: What are the shared values related to the software family engineering: internally or cooperatively focused, individual or central valued, conservative vs. innovative, product vs. process focused
3. **Roles & Responsibilities**: How well does the organisation manage the distinct responsibilities and relationships occurring in the software family engineering: undifferentiated vs. specialised roles for software family engineering

4. **Product life cycle**: An important factor influencing the optimal organisational model is the type of systems that are produced, in relationship to the characteristics of the product life cycle. Factors that play a role are the length of the life cycle, the pace of the generations, and the kind of maintenance that is provided.

We have defined the following level structure. Below, we discuss the levels in more detail:

1. **Unit oriented** – The software family engineering takes place within single, small development *units*.
2. **Business lines oriented** – The software family engineering takes place within several units of a *business line* responsible for a single range of products.
3. **Business group/division** – The software family engineering takes place over business line borders within a single *business group* or product *division,* responsible for many business related products.
4. **Inter-division/companies** – The software family engineering takes place between several *divisions* and *companies*, with mutual trust, each with their own commercial responsibility, which may be conflicting.
5. **Open business** – The software family engineering is not restricted to a collection of companies that trust each other. The open business involves everybody who sees the advantage.

6.1 Level 1: Unit Oriented

The software family engineering takes place within single, small development *units*.

1. **Geographic distribution**: local projects
2. **Culture**: internally focused
3. **Roles & Responsibilities**: software family engineering undifferentiated
4. **Product life cycle**: medium term

This level is referred to as the "development department" by Bosch 2. In this model, software development is concentrated in a single development department. No organisational specialisation exists with either the software product family engineering assets or the systems in the family. The model is especially suitable for smaller organisations. The primary advantages are that it is simple and communication between staff members is easy, whereas a disadvantage is that it does not scale to larger organisations.

The internally focused culture supports the trust and respect that people have for each other, leading to ease in the distribution of work and taking over work from each other. The products have a medium or long life span, with no large maintenance commitments, resulting in only a low-maintenance burden on the developers.

6.2 Level 2: Business Lines Oriented

The software family engineering takes place within several units of a *business line* responsible for a single range of products.

1. **Geographic distribution**: multiple application engineering units
2. **Culture**: cooperative within the business line
3. **Roles & Responsibilities**: software family engineering roles and asset roles
4. **Product life cycle**: medium to long term

In Bosch's work 2, this level is called "Business units." The software family engineering takes place in a single business line involving several development units. An advantage of the model is that it allows for the effective sharing of assets between a set of organisational units. A disadvantage is that business units easily focus on the concrete systems rather than on the reusable assets.

Often, each development unit is assigned the additional responsibility of evolving a subset of the domain assets that are to be reused by the other application units. People are relatively close to each other, but the number of developers within the family is large. Therefore, specialisation is necessary. In particular, the roles for software family engineering and application development are recognised and distributed. People are assigned to be responsible for the maintenance of certain family assets, such as components, or the architecture. In this situation, specific departments in the units, allowing longer life cycles of the products, can conduct product maintenance.

6.3 Level 3: Business Group/Division

The software family engineering takes place over business line borders within a single *business group* or product *division* responsible for many business-related products.

1. **Geographic distribution**: multiple units within one company
2. **Culture**: cooperative across business lines
3. **Roles & Responsibilities**: coordinated roles across business lines
4. **Product life cycle**: short to long term

This level is referred to as the "Domain engineering unit" by Bosch 2. At this level, the software family engineering takes place over different business lines within a single business group or division. A domain engineering unit is responsible for the design, development, and evolution of the reusable assets. Product engineering units are responsible for developing and evolving the products built based on the software family engineering assets. The model is widely scalable, from the boundaries where the business unit model reduces effectiveness up to hundreds of software engineers. Another advantage of this model is that it reduces communication from n-to-n in the business unit model to one-to-n between the domain engineering unit and the system engineering units. Finally, the domain engineering unit focuses on developing general, reusable assets, which addresses one of the problems with the aforementioned model—too little focus on the reusable assets.

6.4 Level 4: Inter-division/companies

The software family engineering takes place between several *divisions* and *companies*, with mutual trust, each with their own commercial responsibility, which may be conflicting.

1. **Geographic distribution**: consortium-based cooperation over company borders
2. **Culture**: externally focused
3. **Roles & Responsibilities**: liaison roles (between companies)
4. **Product life cycle**: short to long term

This level is called "Hierarchical domain engineering units" by Bosch 2. However, it generalises too many kinds of engineering structures. At this level, the development is spread over several cooperating companies. Some of them will act as the subcontractor of others. In particular, this applies when different companies serve different parts of the domain. The structure is necessary to distribute the workload of the domain engineering. This model is applicable especially in large or very large organisations with a large variety of long-lived systems. The advantage of this model is that it provides an organisational model for effectively organizing large numbers of software engineers.

6.5 Level 5: Open Business

The software family engineering is not restricted to a collection of companies that trust each other. The open business involves everybody who sees the advantage.

1. **Geographic distribution**: industry-wide cooperation
2. **Culture**: extremely cooperative, competitive
3. **Roles & Responsibilities**: lobbying, marketing, standardization bodies
4. **Product life cycle**: very short to very long term

At this level, there is an open standard for the structured software family engineering. Several companies improve parts of the family through additions adhering to the standard. Often, but not necessarily (see open source), a single company is strong enough to set the standard.

7 Illustrative Example

In this section, we take the development described by Jaring, Krikhaar, and Bosch 7 to provide an example for evaluation and validation purposes. Although our evaluation framework is clearly not finished, it may already be useful to get initial ideas about its merits. The example development is about a family of magnetic resonance imaging (MRI) scanners. MRI is based on the principles that the nuclear magnetic relaxation times of different tissues differ, which opted magnetic resonance for scanning the inside of the human body.

A warning is in place here. We did not do a formal assessment of the development described here. The evaluation of the example below is done in a very global way, since, for most of the dimensions, a set of precise criteria is not yet available. These still have to be determined. We are aiming for this result for next year. However, the result gives an *indication* of what the evaluated level of the development will be, and where the focus on improvements will be.

7.1 Business Dimension

1. **Identity** – This is very clear: the business is to provide high-quality MRI scanners to the hospitals. The scanners are organised in families at the business level.
2. **Vision** – It is recognised that a software family engineering approach will improve the products' quality and time to market, resulting in a medium-term vision.

3. **Objectives** – The objectives for doing software family engineering within the business are mainly qualitative.
4. **Strategic planning** – The business has a process in place for planning the family.

Based on these findings, we come to a level 3 at the business dimension.

7.2 Architecture Dimension

1. **Software product family architecture** – This is enforced within the organisation.
2. **Product quality** – Quality is tracked from stakeholders to assets and tests.
3. **Reuse levels** – The reuse is managed through many architectural guidelines, rules, and frameworks.
4. **Software variability management** – There are many variation points that are managed through the architecture.

Based on these findings, we come to a level 4 at the architecture dimension.

7.3 Process Dimension

The organisation is assessed to be at CMM level 3. Of course, this does not mean that the process level would be 3 for the family-specific point of view, but presumably, it is close to this level.

7.4 Organisation Dimension

1. **Geographic distribution** – The development as discussed here is part of a development within several units within one company.
2. **Culture** – cooperation over business lines
3. **Roles & Responsibilities** – coordination across business lines
4. **Product life cycle** – very long life cycle. Separate departments for maintenance, distributed development within company borders, and the development are parts of a structured software family engineering.

Based on these findings, we come to a level 3 at the organisation dimension.

Concluding, we arrive with total assessment results in a profile of B3 A4 P3 O3 (i.e., intermediate in all dimensions). This may be very satisfactory for this business, as all dimensions are taken into account. This result implies that none of the dimensions is stressing too far with respect to the others. If the products are serving a mature product domain where the group itself can be seen as a shaper of the business itself, more effort may be put into the business dimension, moving towards level 4. If this grows, the process and organisation can grow to level 4 as well. The precise choice for improvement actions is, however, also dependent on the wishes and goals of the development organisation itself. Please note that *this assessment result does not reflect the actual maturity of the business group. It is intended for illustrative purposes only*. However, the maturity model for software product family engineering offers a highly useful technique for evaluating software-developing organisations that employ product families.

8 Looking Back: Related Work

Since the late 80s, a number of capability evaluation models have been developed for various fields of assessment including systems engineering, software engineering, software acquisition, workforce management and development, and integrated product and process development. The overall idea is to compare a process carried out by an organisation to an ideal that is presented by a capability evaluation model. Process improvement is the elimination of differences between the current and the ideal process. This approach can be characterised as normative and top down. The assumption behind this approach is that improved processes lead themselves to improved products.

The most prominent process improvement framework is the CMM, which was developed by the Software Engineering Institute (SEI) and published in 1993 5, 11. The "non-U.S." counterpart to the CMM is the International Organisation for Standardization/International Electrotechnical Commission (ISO/IEC) 15504 Standard for Software Process Assessment (commonly called SPICE). For systems engineering, the CMM for Software Engineering (SE-CMM) and the Electronic Industries Alliance Interim Standard EIA/IS 731 were developed. CMMI 6 has been created recently by the SEI to integrate the SEI's SW-CMM, EIA/IS 731, and the Integrated Product Development CMM (IPD-CMM) into a single coherent process improvement framework 5 6. Some of the developments resulting in the CMMI are shown in Figure 2.

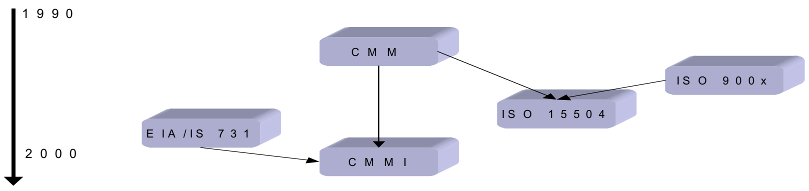

Fig. 2. History of process improvement frameworks

The SEI's Product Line Technical Probe (PLTP) allows the examination of an organisation's readiness to adopt or its ability to succeed with a software product family engineering approach. The PLTP is based on the SEI's Framework for Software Product Line Practice^{SM} (version 4.0 was published in Clements and Northrop's book 4) as a reference model in the collection and in analysis of data about an organisation. The results of applying the PLTP include a set of findings that characterize an organisation's strengths and challenges relative to its product line effort and a set of recommendations.

The Framework for Software Product Line Practice distinguishes 29 practice areas that are divided loosely into three categories 10. Software engineering practice areas are necessary to apply the appropriate technology to create and evolve both core assets and products. Technical management practice areas are those management practices necessary to engineer the creation and evolution of the core assets and the products. Organisational management practice areas are necessary for the synchronization of all the activities for software product family engineering.

Our model is an improvement with respect to the Framework for Software Product Line Practice in the fact that it clearly separates the distinct BAPO development concerns. The focus of the SEI framework is mainly on what we call process concerns because the practice areas of that framework roughly cover the different issues addressed by CMMI, engineering, support, process, and project management. We have identified architecture, business, and organisation aspects as important as the process, which will be pursued to get a better understanding of them. Another important difference between the SEI framework and our model is their structure. The SEI framework does not comprise any levels, implying a direction of improvement, while our model offers levels and is thus also suitable for benchmarking and roadmapping.

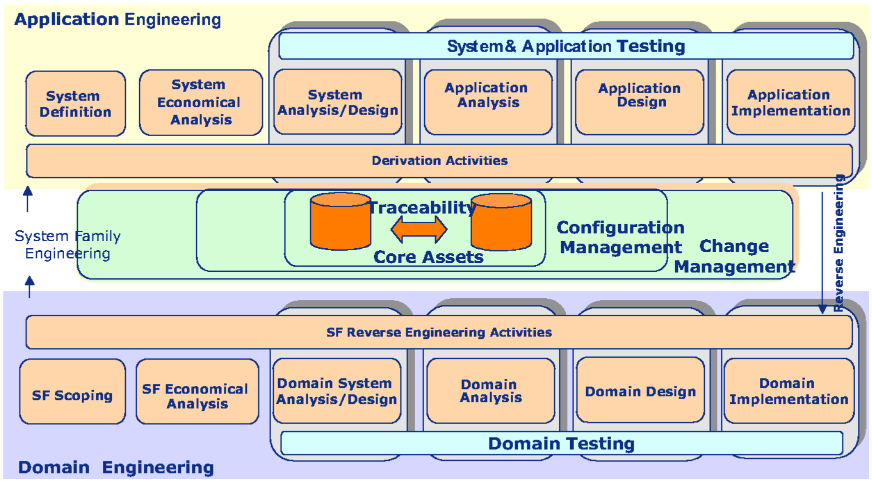

Fig. 3. CAFE Process Reference Model

The ITEA's CAFÉ project has developed a reference framework for software family engineering practices (CAFÉ-RF) 7. The CAFÉ-RF takes into account the different BAPO aspects and consists of several models: a reference information technology (IT) life cycle for system families (which identifies the major phases in an IT organisation when carrying out a system family effort), a process reference model (CAFÉ-PRM, which represents major engineering activities operating on the core assets), and an asset reference model (CAFÉ-ARM, which represents the major assets of interest). The reference process model shown in Figure 3 shows only the most important activities on software family engineering, grouped according to domain engineering (development for reuse) at the bottom, application engineering (development with reuse, for customers) at the top, and supporting processes for asset management in the middle. No explicit flows between activities are depicted since they can change over and within organisations. Application engineering processes are put at the top, since they are the most important business concerns. Domain engineering and asset management are supporting and only necessary for an efficient software family engineering process.

9 Conclusion and Future Work

This paper improves the evaluation frameworks for software family engineering presented in the work of van der Linden and associates 9 and Bosch 3. The framework serves for benchmarking the organisation against others, enables assessments for single organisations, and provides directions for improvement. As there are four groups of development concerns (BAPO), we have defined four distinct evaluation scales. Evaluation of an organisation will lead to a profile with separate values for each of the four scales. Depending on the organisation context, there will be an optimum profile for that organisation (i.e., top marks for each dimension may not be optimal from a business and economic perspective). The organisation may use the framework to determine whether it has the optimum profile. In case it does not have the right profile, it helps to select those actions that should be done to reach the optimum. Care should be taken that improvement actions for one of the profile scales may lead to reduced values for the other scales.

We plan to improve the present evaluation framework by providing more details and clear descriptions, taking into account the practices in present industrial software product family engineering. This improvement will be performed mainly in the ITEA FAMILIES project.

Acknowledgments. We thank Lech Krzanik for the discussions during the Dagstuhl Seminar 03151 on Product Family Development, April 6 – 11, 2003, Schloss Dagstuhl, Wadern, Germany. These discussions were the basis of the first paper 9 on the framework to which all have contributed. Moreover, we thank Klaus Pohl, Peter Knauber, Linda Northrop, and Günter Böckle who were co-organizers (with Frank van der Linden) of this Dagstuhl seminar, facilitating the discussions. Finally, we thank all those involved in the ITEA projects ESAPS, CAFÉ and FAMILIES presenting and discussing best practices in family engineering in many workshops and other meetings. There are too many people involved in these projects to name them all.

References

1. Pierre America, Henk Obbink, Rob van Ommering, Frank van der Linden, CoPAM: A Component-Oriented Platform Architecting Method Family for Product family Engineering, Proceedings SPLC-1 pp. 167-180
2. Jan Bosch, Software Product Lines: Organisational Alternatives, Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001), pp. 91-100, May 2001.
3. Jan Bosch, Maturity and Evolution in Software Product lines: Approaches, Artefacts and Organisation, Proceedings of the Second International Conference on Software Product Lines (SPLC 2), Springer LNCS 2379 pp. 257-271, August 2002.
4. Clements, Paul; Northrop, Linda; Software Product Lines – Practices and Patterns, Addison Wesley, 2001.
5. CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Continuous Representation (CMMI-SE/SW/IPPD/SS, V1.1, Continuous), Technical Report CMU/SEI-2002-TR-011, Carnegie Mellon University, Pittsburgh, 2002

6. CMMI for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Staged Representation (CMMI-SE/SW/IPPD/SS, V1.1, Staged), Technical Report CMU/SEI-2002-TR-012, Carnegie Mellon University, Pittsburgh, 2002
7. Michel Jaring, René L. Krikhaar and Jan Bosch, Visualizing and Classifying Software Variability in a Family of Magnetic Resonance Imaging Scanners, Software Practice and Experience, June 2003.
8. Frank van der Linden, Software Product families in Europe: The Esaps and Café Projects, IEEE Software, July/August 2002, pp. 41-49
9. Frank van der Linden, Jan Bosch, Erik Kamsties, Kari Känsälä, Lech Krzanik, Henk Obbink, Software Product Family Evaluation, Proceedings PFE 2003, 5th workshop on product-family engineering, November 2003, Springer LNCS 3014, pp. 376-394.
10. Northrop, et al. A Framework for Software Product Line Practice Version 4.1, http://www.sei.cmu.edu/plp/framework.html
11. M. Paulk, et. al., Capability Maturity Model of Software, Version 1.1. Tech Report CMU/SEI-93-TR24, Carnegie Mellon University, Pittsburgh, 1993
12. Klaus Schmid, Scoping Product lines, Software Product lines - Proceedings of the First Software Product Line Conference (SPLC1), Kluwer, August 2000, pp. 513-532