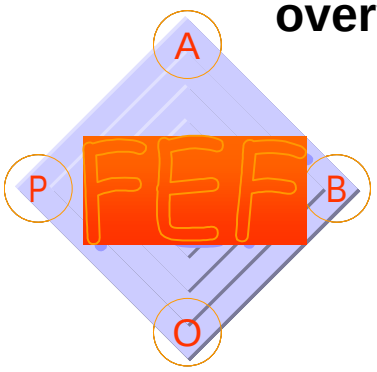
ITEA
INFORMATION TECHNOLOGY
FOR EUROPEAN ADVANCEMENT

Family Evaluation Framework overview & introduction



Frank van der Linden

Partner:	Philips Medical Systems Veenpluis 4-6 5684 PC Best, the Netherlands
Date:	29 August, 2005
Number:	PH-0503-01
Version:	1.0
Status:	Final
Level:	Consortium-wide
Contributors:	Philips
work product:	Management
Topic:	
work product Leader:	

Accepted: Y

Eureka S! 2023 Programme, ITEA project ip02009

PHILIPS

Families

Table of Contents

PHILIPS

Abstract

This document gives an overview and introduction of the Family Evaluation Framework. It is meant to assess organisations in their effectiveness of software product family engineering. The framework is built on the four BAPO concerns: Business, Architecture, Organisation and Process. Each of these concerns has a separate dimension in the FEF. This allows evaluation of the organisation for each of these software engineering concerns separately. It gives insight where the organisation best can improve itself. Especially for the process dimension, which is based on CMMI®, there is a separate report available giving more details of amplifications to the existing CMMI®, which are used within the FEF.

1 Introduction

Software product family engineering is a strategic approach that impacts on business, organisation and technology. Software product family engineering has proven to be the way to develop a diversity of software products and software-intensive systems at low costs, in short time, and at the same time with high quality. Numerous experience reports document the significant achievements gained by introducing product families in software industry.
Within 1995 and 2005, a series of cooperation projects were performed on the topic of software product family engineering. This document reports one of the consolidated results of this series of projects. The series started with the ESPRIT project ARES [8]. Between 1995 and 1998, three European companies and three Universities worked together on software family architecture. Between 1998 and 2000, the ESPRIT project PRAISE [22] investigated the process issues of software product family engineering. Based on these results, a group of European industries, together with a collection of research institutes and small and medium enterprises collaborate on the topic since 1999 in a series of projects within the ITEA [21] framework. These are the, ESAPS [9][19], CAFÉ [10][18] and FAMILIES [11][20] project. Companies within these projects are working on a large variety of mainly embedded systems including medical imaging, mobile phones, flight control software, utility control, supervision and management, financial services, and car electronics.
The two main differences of software product family engineering with single system software engineering is managed reuse facilitated through a separation of the development into domain and application engineering and the explicit management of variability. Domain engineering produces domain assets, with variability for reuse. Application engineering produces systems by reusing variants of the domain assets.
The ESAPS project investigated the development process, and variability management from an architecture and quality viewpoint. The CAFÉ project introduced more business concerns, requirements, asset management and testing. The FAMILIES project consolidated the results and exploits the ESAPS results, improve and automate them through model driven family engineering and reuse over family borders.

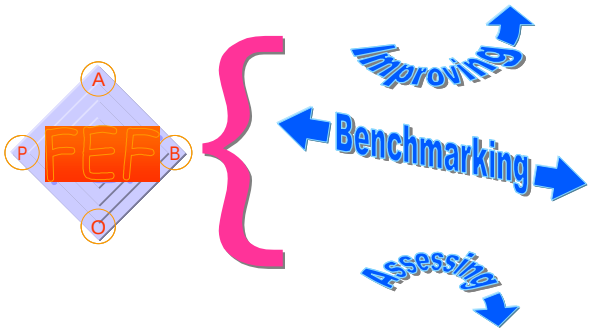


Figure 1: Use of the FEF

The Family Evaluation Framework (FEF) is one of the consolidation results of the FAMILIES project. It combines in a single framework the results of the experiences with software product families in the ESAPS, CAFÉ and FAMILIES projects. The purpose of the FEF is to evaluate the performance in software product family engineering of larger or smaller parts of companies including business units, divisions, and even complete companies. The evaluation is based on the best practices and other experiences of the companies cooperating in the projects. There are several reasons why such an evaluation is useful; see Figure 1. For instance, it can be used as an assessment to find out how a department is doing software product family engineering. Alternatively, it may be used as a benchmark tool, to compare the company's software product family engineering capabilities with others. It can be used as a decision tool,

to find out what have to be done to start, or improve, software product family engineering within a single department.
This document gives an overview of the FEF. There are four different dimensions that each can be evaluated independent of the others. Within each dimension, there is a set of evaluation aspects, and a set of levels. The document describes the levels in terms on the way the aspects are dealt with. In the remainder of this chapter, we give an overview of related approaches, and how the FEF relates to these. In addition, there is a section covering the differences of software product family engineering with single system software engineering. Finally, we introduce an introduction of a small example that is the basis of a running example in the document. The next chapters treat the main elements of the FEF, the BAPO model (chapter 2) the framework (chapter 3) and the different dimensions (chapters 4-7). Chapter 8 deals with the applicability of the FEF for different purposes in an organisation, Chapter 9 deals with special issues of complex organisations, where software product family engineering is distributed over several departments. Finally, in chapter 10 some conclusions are drawn.

1.1 Related approaches

The FEF is not the fist model to evaluate or assess software development. In particular, in the area of software development processes there are several capability evaluation models. The model gives an abstract view of the software development process. Actual activities in an organisation are mapped to the abstract ones, and gaps and differences are used to plan improvement. The most prominent process improvement framework is the Capability Maturity Model (CMM), which was developed by the Software Engineering Institute (SEI) and published in 1993 [12], later integrated to be applicable for systems engineering in CMMI® [16][17].
In the field of software product family engineering, the SEI published a Framework for Software Product Line Practice℠ [4] that distinguishes 29 practice areas, which are divided into three categories.

- 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 synchronisation of the entire software product family activities.

The SEI's Product Line Technical Probe (PLTP) allows to examine an organisation's compliance to adopt a software product family approach. The PLTP is based on the SEI's Framework for Software Product Line Practice℠ as a reference model in collection and in analysis of data about an organisation. The results of applying the PLTP include a set of findings, which characterize an organisation's strengths and challenges relative to its product family effort, and a set of recommendations.
In addition, there exist several initial economic models measuring the success of software product family engineering; see e.g. [7][14]. These methods try to evaluate the business value in several ways. Jan Bosch, both proposed an initial model on software product family architectures in [3], and an initial investigation on organisational structures in [2].
All these approaches are used in different ways in the FEF. The CMMI® is used as a basis for the process dimension within FEF. The Framework for Software Product Line Practice℠, the economic and architecture models are used to check to see if everything that is necessary for software product family development is included. The economic and architecture models were used as an inspiration for the FEF dimensions dealing with these matters. The FEF adds more structure to the whole, aiming to guarantee a more complete picture.
The main differences of software product family engineering with single system software engineering are the separation of the development into domain and application engineering and the explicit management of variability [11].
The management of variability introduces the notions of variation, through variation points and variants. Introducing these notions in software development has mainly, but not exclusively, impact in the requirements and architecture where they have to be represented in all kinds of development models. In addition, the process, organisation and business reflect the availability of the variability. As stated above, the process is split into two parts domain and application

engineering and the explicit management of variability; see Figure 2. Domain engineering produces domain assets, with variability for reuse. Application engineering produces systems by reusing the domain assets.
The FEF focuses on these differences with single system software engineering. Organisations that need to improve, or evaluate themselves, in the traditional way of software engineering, have to rely on other sources, since this is not covered by the FEF.

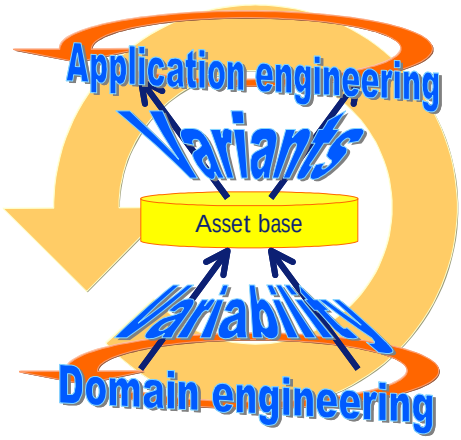


Figure 2: Software product family Engineering

1.2 Example

The example in this document is an imaginary company, ProtAct, providing security systems for office buildings and business plants. It delivers observation rooms, camera's, intrusion, fire and water sensors, and all kinds of alarms. It provides door locks operated by keypads and other kinds of authentication mechanisms. Since the systems are sold in many countries, many languages are supported. Each client has his own configuration of the system, and there are many possible configurations available. Therefore, ProtAct has organised its software development as a family development. ProtAct wants to assess itself, to find the best ways to improve this development in future in order to reduce software cost to a minimum level, and keep lead times of new developments as short as possible.

2 BAPO Dimensions

The main concerns of software engineering are captured in BAPO [1][15]. It separates the concerns on **B**usiness, **A**rchitecture, **O**rganisation and **P**rocess.

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

The four BAPO concerns are all interlinked. Applying changes in one concern induces changes in the others. The BAPO acronym denotes a natural order to traverse the concerns. The Business is the most influential factor. This has to be set up right in the first place. The architecture reflects the business concerns in software structure and rules. The process enables the building of the software, based on architecture according to business rules. Finally, the organisation should host the process, assigning units and people that are responsible for business, architecture responsibilities. Note that the ease and speed of changing a concern decreases from Business over Architecture, Process to Organisation.
We use this separation of concerns to provide four dimensions of the family evaluation framework. An organisation will have a separate evaluation level for each of the BAPO con-

cerns. The interdependence between the BAPO 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. However, in many cases improvements the dimensions can be addressed separately. In the end, all BAPO aspects have to be treated to get improvements. A low score in one dimension may hamper the achievement of reaching a high score at another dimension. 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.
Since BAPO applies to software engineering, it also applies to software product family engineering. Consequently, the FEF has a dimension for each of these concerns, involving aspects and levels. In each dimension is evaluated how much software product family engineering is incorporated. For each dimension, there are many specific ways to perform software product engineering. Similar to the philosophy of CMMI®, the FEF evaluation does not deal with specific ways to perform a certain activity, modelling, structuring, responsibility or task. The ESAPS, CAFÉ and FAMILIES projects have delivered a large amount of such methods, tools & techniques. These are best practices, and can be applied in software system engineering. They provide a good insight of what is necessary in the different BAPO dimensions. However, none of them is obligatory. Often the specific methods, tools and techniques change over time.
Applying the four BAPO dimensions to software family engineering, we get the following aspect to consider:

- **B**, The business concern additionally has to deal with the business relationships between domain and application engineering and the costs, profits, market value and planning of variability.
- **A**, The architecture is split over domain and application architectures, related via variability. Important architecture concerns deal with the right mechanisms for variability and how the software product family architecture (Domain-engineering) influences the corresponding application architectures (Application-engineering), and vice versa.
- **P**, Processes can be distinguished between domain, application and collaboration and coordination processes, they all have can be evaluated against a maturity model such as CMMI®
- **O**, Organisation has structures and responsibilities for domain and/or application engineering and for collaboration and coordination roles. In particular, the organisation distributes responsibilities between platform (domain engineering), corresponding applications (application engineering) and collaboration and coordination, and determines the relative importance between them.

2.1 Example

Viewing ProtAct with respect to the different BAPO concerns, we see the following elements in software engineering, and software product family engineering.
Business: The ProtAct company has an internal tax system that deducts money from departments that get income, to fund serving departments. The departments that do domain engineering are examples of such departments that get funding. The business has a fixed amount of about 40% of the profit to be used for domain engineering. Product management uses a road map that plans the future variants of the product. It involves when new variants are to be built, and gives feedback to marketing and sales when these variants are available. The road map is prepared in agreement with development that gives feedback on the distribution of new requirements over generic and specific software. The management of ProtAct is forcing to make as much as possible generic, and tries to reduce the cost of specific developments, since they are aware that that costs too much. In particular, the management put limits on development budgets to reduce the total development cost.
Architecture: ProtAct has a reference architecture for the complete family of products. The reference architecture determines a layered structure. Each layer has a framework of components that have to be present in each product. The lower layer consists of the single operating system and database that is used in all applications. Variants are to be built by adding plug-in components at interfaces that are specially designed for them. Thus, a single application ar-

chitecture consist of the collection of fixed framework components and a configuration of plug-ins. A plug-in component may be specifically built for the application, but many plug-in components are shared by several applications, and are therefore part of the domain architecture.
Process: The development departments within ProtAct have separate processes for domain and application engineering. Domain engineering follows several iterative developments for separate parts of the architecture. A separate development is available for keeping the architecture in shape. Application-engineering departments usually follow a waterfall model to produce a single application. Collaboration and coordination processes deal with:

- Selection of reusable domain artefacts in the application
- Feedback of problem reports and application priorities from application engineering towards domain engineering
- Determination of the standard interfaces between domain-engineering frameworks and the plug-in components
- Promotion of specific (application) components and interfaces towards domain engineering

Organisation: Domain engineering is performed in a department separate from several application-engineering departments. Collaboration is supported through many cross-departmental groups that have the responsibility for one or more of the collaboration subjects, such as road maps, global architecture issues, interfaces, problem reports and maintenance, and promotion of assets from application to domain.

3 FEF framework

The FEF evaluates the software product family engineering performance of a company or a department in a company. It does not aim to evaluate single system software engineering practices. It only focuses on the aspects that are specific for software product family engineering. In particular, it has an emphasis on the separation in domain and application engineering and the management of variability.
The FEF framework is built on BAPO. Each BAPO concern is of importance for software family engineering, and therefore it needs its own attention. The FEF framework has a dimension of each of the four BAPO concerns. In each dimension, the performance of a company, or department, can be measured separately. The result of an FEF evaluation is a profile consisting of four values, one for each BAPO dimension. On purpose, the framework allows for diverse results in the different dimensions. This may be the case in companies where one or more BAPO concerns get more attention than the other concerns.
For each BAPO dimension a collection of evaluation aspects is identified, that comprise the main influential factors for the evaluation. The evaluation values per dimension are summarised in five levels, each determining a certain extent to which the aspects cover software product family engineering. The business dimension measures the business involvement in software family engineering, and the business consequences of managed variability. The Architecture dimension mainly deals with the relationship between domain and application architectures. This takes into account how variability is modelled in the architecture. The Process dimension measures which family process are there, and what is their maturity. Finally, the organisation dimension measures the effectiveness of the distribution of domain and application engineering over the organisation.

3.1 Connection to other approaches

The process maturity model, CMMI®, has been an inspiration for the complete framework. In each dimension, we have used some characteristics of the CMMI®. Each dimension has five levels, where the initial level represents the state that everybody achieves. Moreover, in each dimension it is measured what has to be done to reach a level, but specifics on how to do it are not given. This allows improvements, which are presently not envisioned. In contrast to CMMI®, we have not a single scale, but keep a separation of the different BAPO concerns, an assessment leads to four values. Differences in values may indicate that specific attention has to be paid for a specific BAPO concern.
In case of the process dimension, the levels are the CMMI® levels. The software product family aspects are reflected in amplifications of CMMI® practices, mainly to the levels 2, 3 and 4.

PHILIPS

Amplifications deal with the separation of domain- and application-engineering, and to coordination activities.
As the FEF does not describe specific details of how things are done, the Framework for Software Product Line Practice[SM], the economic and architecture models are mainly used as a check to see if everything that is necessary for software product family development is included. They can be used as examples of how to proceed with an improvement plan, but they are not used to prescribe ways to behave in each dimension. Similarly, the economic and architecture models were used as an inspiration for the FEF dimensions dealing with these matters. The FEF adds more structure to the whole, aiming to guarantee a more complete picture.

3.2 Example

ProtAct is interested in seeing how well they are doing in software product family engineering. They want to know what are the best improvement actions to be taken. Initially the expectation is that the architecture and organisation dimensions are satisfactory, but that the business and process may need improvement. The FEF will indicate whether they are right.
The result of execution of the FEF is the following profile: **B3, A4, P2, O3**. This means that their ideas about the business were not completely right, but some improvement in that dimension is still possible. However, the process indeed needs a lot of attention. In addition, the organisation has room for improvement.

4 Business Dimension

The business dimension deals in general with how to make profit from your products. In doing so, the business has several techniques to influence the development process, and improve marketing and sales of the products. Most of the business issues hold in any organisation making (embedded) software, and will not be regarded here. However, for software product family engineering the business concern additionally has to deal with the business relationships between domain- and application-engineering, involving investment decisions measuring the costs and profit of these activities and the means to use the sales of the product to pay the different activities. Finally, the business dimension deals with the managed variability. It measures and uses costs and profits of the amount of variability in the products. It also uses this to plan prices, and marketing strategies for family products.
The following aspects play a role in the business dimension of software product family engineering.

- Sales, marketing, product management involvement
  How much is marketing, sales and product management involved in and influenced by the software product family.
- Budgeting and investment
  How much are budgets influenced by software product family engineering
- Vision and business objectives
  How well does the organisation aim for a future involving software product family engineering
- Strategic planning
  How well does the organisation plan the software product family business and development?

4.1 Levels

The following levels are recognised for the business dimension of software product family engineering.

1. Project based
2. Aware
3. Managed
4. Measured
5. Optimised

In the following sections, each of these levels is discussed in more details.

4.1.1 Level 1: Project based

This is the basic level. The business is arranged for project based single system engineering. Domain-engineering results and variability are not visible at the business level. None of the aspects covers software product family engineering. With regard on the business concerns, we see the following typical situation:

- Sales, marketing, product management involvement
  There is no, or little, involvement in of software product family engineering, nor is there much understanding of it. Systems are planned, sold and marketed on a single system basis.
- Budgeting and investment
  There are no specific budgets for domain engineering. Instead, budgeting is done on a per system basis.
- Vision and business objectives
  The vision and the business objectives do not mention the existence of software product family engineering
- Strategic planning
  The business planning does not consider relations among systems

4.1.2 Level 2: Aware

At this level, the business is aware of the benefits of software product family engineering for the company. It provides some context in which software product family engineering can be done. However, a clear management of software product family engineering is not available. This level shows following typical situation

- Sales, marketing, product management involvement
  By the selling force, marketing and product management there is an awareness of the opportunities of software product family engineering in marketing, sales and product planning. It is expected that managed variability will lead to more variability in the sold systems, and that production cost will be lower. The mere fact of supporting more variants is seen as an additional benefit for the customer. However, there is no clear strategy available for using the software product family in marketing, sales and product planning.
- Budgeting and investment
  The business invests in domain-engineering activities to support a repository for reusable assets. There are budgeting consequences to encourage the use of the domain-engineering results.
- Vision and business objectives
  There is a commitment from top management to do software product family engineering. However, there is no clear vision on its use for the company.
- Strategic planning
  The planning is still committed to single system development. However, the results of domain engineering are taken into account in an opportunistic way in product roadmaps.

4.1.3 Level 3: Managed

At this level, software product family engineering is part of business strategy. It takes control on the execution of the corresponding activities, the benefits and the drawbacks.

- Sales, marketing, product management involvement
  The expected return on investment drives the marketing, sales and development of software product family products. Marketing addresses the user values of having a large amount of variability for low costs.
- Budgeting and investment
  Software product family engineering influences the investment decisions. There is a well-defined budget for domain and for application-engineering activities. There is an institutionalised mechanism to generate budget for domain engineering by the sales of systems produced by application engineering. There is an awareness of the costs and profits of variability, and how that generates a return on investment.
- Vision and business objectives

The top management strongly supports software product family engineering. The organisation's vision and business objectives incorporate in a qualitative way the software product family, its value for the organisation, and its evolution. The software product family engineering strategy is visible to the organisation.

- Strategic planning
  There are separate plans and roadmaps for domain and application engineering. The plans are related, and commonalties in applications provide the basis of the domain-engineering plan.

4.1.4 Level 4: Measured

At this level, the business measures software product family engineering to improve the strategy.

- Sales, marketing, product management involvement
  The costs, profits and return on investment of software product family products and managed variability are measured to determine the marketing and sales strategy. In addition, the product management strategy is guided by measured return on investment.
- Budgeting and investment
  The costs and savings of reuse and variability and software product family engineering is measured, and reflected in the budgets.
- Vision and business objectives
  The top management measure the effects of software product family development. The business objectives incorporate in a quantitative way the software product family, its value for the organisation, and its evolution. The advantages of software product family appear in the vision and business objectives. The drawbacks are recognised, and measures are planned to diminish their effects. The software product family engineering strategy is visible outside the organisation (clients, investors, …).
- Strategic planning
  The plans and roadmaps are coordinated to get the best business value out of software product family engineering.

4.1.5 Level 5: Optimised

At this level, the business strategy involves optimisation of software product family engineering.

- Sales, marketing, product management involvement
  Marketing, sales and development has an understanding of software product family engineering and use its value in the strategy. Marketing and sales use the costs, profits and return on investment of software product family engineering. These costs, profits are also applied to improve the business strategy.
- Budgeting and investment
  There is an accurate integration of budgeting and investment with the forecast of sales, costs and savings of software product family products.
- Vision and business objectives
  The vision and business objectives are influenced by software product family development upon a well-understood basis.
- Strategic planning
  The plans and roadmaps are used strategically to get the best business value out of software product family engineering.

4.2 Summary

The evaluation levels for the business dimension are: Project based, Aware, Managed, Measured and Optimised. At the initial level, there is no real business involvement in software product family engineering. The business deals with a project based organisation, and all projects are treated in a similar way by the management. The business dimension of the FEF deals with the following aspects

- Sales, marketing, product management involvement
  From starting in an unaware state, product management, marketing and sales force gets aware of the possibilities of dealing with managed variability. In the higher level,

the marketing and sales grow to use variability management in such a way that the product planning supports the expected sales, and marketing supports the variants that are available in the family.

- Budgeting and investment
  Initially there are no specific budgeting or investments available for software product family engineering. At the next level, the management is aware of possible benefits of software product family engineering, but the best way to deal with this is still to be decided. Initial investments are made to fund domain engineering, is supported by the top management. In the higher level, budgets, and funding of domain engineering gets more sophisticated, and supports the position of domain engineering within the organisation. Domain engineering earns its own budget through a good internal business model.
- Vision and business objectives
  Initially, the vision is only by the people doing software product family engineering. Next, the management gets aware and incorporates the managed variability in their vision and objectives for the future.
- Strategic planning
  From an initial stage, where software product family engineering is not visible in the plans, at the higher levels, it becomes an important driving force in the planning.

4.3 Example

The ProtAct company has the following business evaluation:

- Sales, marketing, product management involvement
  Marketing and sales is aware of the software product family. Product management keeps an eye on features that are needed in future and it is in contact with development to determine when such features may be available. The marketing department uses this information to determine which features can easily be supported and which are difficult to build, or take some time to be finished. The former ones are priced lower than the latter ones. As there is no specific measurement of the costs and profits of variability, this aspect is satisfied at level 3: Managed
- Budgeting and investment
  An internal tax system deducts money from departments that sell products, to fund domain engineering. The business has a fixed amount of about 40% of the profit to be used for domain engineering. The costs and savings of product family engineering are not yet measured, therefore also for this aspect the evaluation is satisfied at level 3: Managed.
- Vision and business objectives
  The management of ProtAct is forcing to make as much as possible generic, and tries to reduce the cost of specific developments, since they are aware that costs are too high. There are limits on development budgets to reduce the total development costs and the management advocates this to the organisation. Again, product family engineering is not yet measured, therefore also for this aspect the evaluation is satisfied at level 3: Managed
- Strategic planning
  Product management uses a road map to plan the future variants of the product. It involves new variants that are to be built, and it gives feedback to marketing and sales when these variants are available. The road map is prepared in agreement with development that gives feedback on the distribution of new requirements over generic and specific software. Coordination among different roadmaps is still to be done. Thus, also for this aspect the evaluation is satisfied at level 3: Managed

Combining the results, the ProtAct company is evaluation for the business dimension at **B3**: Managed.

5 Architecture Dimension

The architecture dimension deals with the technical means to build the software. It involves more than what is traditionally known as architecture, it also involves requirements, domain

modelling, detailed design and testing. However, the architecture is the main topic of this dimension. It determines the technical realisation of the product in the family. The architecture is split over domain and application architectures, related via variability. The evaluation in the architecture dimension mainly deals with the relationship between domain and application architectures. It takes into account how variability is modelled in the architecture. Important architecture concerns deal with the right mechanisms for variability and the relationship between software product family architecture (Domain-engineering) and corresponding application architectures (Application-engineering).
The following aspects play a role in the architecture dimension of software product family engineering.

- Asset reuse level
  The extent of the use of domain assets in products
- Software product family architecture
  The extent of the software product family architecture determining the application architectures
- Variability management
  The explicit use of variation points and supporting mechanisms

5.1 Levels

The following levels are recognised for the architecture dimension of software product family engineering.

1. Independent product development
2. Standardised domain independent infrastructure
3. Software platform
4. Derivable variant products
5. Automated product derivation

In the following sections, each of these levels is discussed in more details.

5.1.1 Level 1: Independent product development

This is the basic level. There are only architectures for single systems. Reuse is not visible in the architecture. None of the aspects covers software product family engineering. With regard on the architecture concerns, we see the following typical situation:

- Asset reuse level
  There is no or only unsystematic reuse.
- Software product family architecture
  There is no software product family architecture.
- Variability management
  There is no variability to manage.

5.1.2 Level 2: Standardised domain independent infrastructure

At this level, reuse is focussed on third party infrastructure. Common software infrastructure (such as middleware or COTS) is defined. Nevertheless, there is no formal reuse of domain specific assets.

- Asset reuse level
  There is a common third party infrastructure defined and in use. There is only ad hoc reuse, mainly based on the repository of the third party products.
- Software product family architecture
  The software product family architecture is derived from the third party infrastructure. It only enforces the use of this infrastructure.
- Variability management
  Only variability offered by the third party infrastructure is somewhat limited. The remainder of the variation is open to be determined by the application architect.

5.1.3 Level 3: Software platform

At this level, domain commonality is captured and implemented in a software platform. There is a common reference architecture available for all applications, mainly determining the use

of the common platform. This platform is used for the different products. The platform could be configured. Nevertheless, there is no variability support for product derivation

- Asset reuse level
  There is a common platform defined as a collection of common assets in a domain repository. Reuse is restricted to this platform, and it is restricted by architectural constraints.
- Software product family architecture
  The common reference architecture contains common rules, and determines the use of the common platform. This incorporates the common use of certain quality solutions offered by the reference architecture. The common reference architecture is in use for the applications.
- Variability management
  The reference architecture determines which configurations of domain assets are allowed within applications. It determines explicit variation points, where application specific variants may be bound.

5.1.4 Level 4: Derivable variant products

At this level, the domain commonality and variability is captured and a reference architecture is specified for the complete software product family. Domain assets include support for deriving products. Variability management is explicitly addressed in the software product family architecture.

- Asset reuse level
  There is systematic and managed reuse based on an asset repository, with explicit variability in the assets
- Software product family architecture
  There is an explicit reference architecture determining explicitly where application architectures may vary. Many quality solutions are incorporated in the software product family architecture.
- Variability management
  The software product family architecture determines which configurations are allowed for application architectures. The reference architecture determines variation points and restricts the allowed variants for most of these variation points. It determines rules that application specific variants have to obey.

5.1.5 Level 5: Automated product derivation

At this level, the domain engineering is dominant and application engineering is only marginal. Products can be derived automatically from the domain without product specific development. The reference architecture supports the automated configuration of products.

- Asset reuse level
  There is systematic reuse based on an asset repository, with explicit variability in the assets plus automated product derivation mechanisms.
- Software product family architecture
  The reference architecture determines the application architectures completely, with automated configuration support, to derive specific applications. Quality is supported through the managed use of specific variation points.
- Variability management
  Variability management is fully integrated in the architecture. Variability is described in models and/or semantic and syntactic locally standardised languages. Variants are derived automatically.

5.2 Summary

The evaluation levels for the architecture dimension are: Independent product development, Standardised domain independent infrastructure, Software platform, Derivable variant products and Automated product derivation. At initial level, there is no domain architecture available, each product gets its own architecture, and reuse is unsystematic and ad hoc.

- Asset reuse level

From an initial level of unsystematic reuse, this aspect grows via a common infrastructure to larger parts of the architecture that is reused, ending in a level where reuse is managed through variation points.

- Software product family architecture
  From an initial level of no available domain architecture, this aspect grows to a reference architecture that governs all applications, leaving not much freedom for the application architectures.
- Variability management
  From an initial level with no variability management, the aspect grows to a level where variability is fully integrated in the architecture and variants are derived automatically.

5.3 Example

The ProtAct company has the following architecture evaluation

- Asset reuse level
  Important reusable assets are the architecture, the framework components and the frameworks themselves. A single application consists of a collection of fixed framework components and a configuration of plug-ins. A plug-in component may be specifically built for the application, but many plug-in components are shared by several applications, and are therefore part of the domain architecture. The requirements and generic (regression and integration) test cases are reused over the whole family. As not all plug-ins in the family are available, and some of them have a complicated structure, level 5 is not reached, but level 4: Derivable variant products is applicable for the asset reuse level.
- Software product family architecture
  There is a reference architecture for the complete family of products. The reference architecture determines a layered structure. Each layer has a framework of components that have to be present in each product. The lower layer consists of the single operating system and database that is used in all applications. Variants are to be built by adding plug-in components at interfaces that are specially designed for them. Thus, a single application architecture consists of the collection of fixed framework components and a configuration of plug-ins. A plug-in component can be added only using specific interfaces. Plug-in components are variants of the variation point embodied in these interfaces. Components have a configuration interface that is used to select the right variant in the applications. There are explicit variation points that govern the values offered to these interfaces. As still some parts governing specific plug-ins are not completely determined by the architecture, the software product family architecture is at level 4: Derivable variant products.
- Variability management
  Variant systems are to be built by adding plug-in components at interfaces that are specially designed for them. Variation is governed by variation in requirements, that determine which configurations have to be built, and which plug-ins have to used, or built. Reusable components have standard interfaces to select a variant. This is all determined in the reference architecture that determines where and how variation is possible. Again, variability management is satisfied at level 4: Derivable variant products.

Combining the results, the ProtAct company is evaluation for the business dimension at **A4**: Derivable variant products.

6 Process Dimension

The process deals with the roles, responsibilities and relationships within software development. It deals with the ways to perform activities to do the development. For software product family engineering, the processes can be distinguished between domain, application and collaborating processes. The domain and application engineering processes are development processes, and CMMI® can be applied to each of them separately. Because application-

engineering processes have to be coordinated with domain engineering, and with other application-engineering processes, additional collaboration processes have to be available.
The following aspects play a role in the process dimension of software product family engineering:

- Domain engineering
  Those processes that perform the domain engineering work
- Application engineering
  Those processes that perform the application engineering work
- Collaboration
  Those processes that perform the collaboration activities between domain- and application engineering

6.1 Levels

The levels for the process dimension of software product family engineering are based on the CMMI® levels. The software product family engineering levels contain amplifications for CMMI® practices at the same level. See the deliverable of task 2.2 [5] for more details for the process dimension. Processes are separated for domain- and application engineering. Further processes are needed for collaboration between domain- and application engineering. Based on CMMI®, the following levels are recognised for the process dimension of software product family engineering.

1. Initial
2. Managed
3. Defined
4. Quantitatively managed
5. Optimising

In the following sections, each of these levels is discussed in more details.

6.1.1 Level 1: Initial

This is the basic level, domain and application engineering and collaboration processes are performed at CMMI® level 1.

- Domain engineering
  If present at all, domain-engineering is performed at CMMI® level 1
- Application engineering
  Application engineering is performed at CMMI® level 1
- Collaboration
  If present at all, collaboration is performed at CMMI® level 1

6.1.2 Level 2: Managed

At this level, basic project-management is in place. For software product family engineering, domain and application engineering projects are synchronised.

- Domain engineering
  Domain engineering is performed at CMMI® level 2. Amplifications are necessary for the following process areas:
  - ***Requirements Management (RM):*** Manage software product family requirements. Maintain traceability between variation points and variants.
  - ***Project Planning (PP):*** Define variability. Involve application engineering as stakeholder for reusing the domain assets. Define a policy of communication and cooperation with application engineering.
  - ***Project Monitoring and Control (PMC)***
  - ***Measurement and Analysis (MA):*** Take global product family view into account.
  - ***Configuration Management (CM):*** Attention should be paid to baseline created and released for reusable assets.
- Application engineering
  Application engineering is performed at CMMI® level 2. Amplifications are necessary for the following process areas:

- ***Requirements Management (RM):*** management of application requirements, both as reused domain requirements and as application specific requirements.
- ***Project Planning (PP):*** Reuse domain assets and bind variability. Analyse the risk of dependency on domain engineering. Involve domain engineering as stakeholder for developing reusable domain assets. Consider influence of domain engineering to the scope of an application project.
- ***Project Monitoring and Control (PMC):*** Monitor the usage of reusable assets.
- ***Measurement and Analysis (MA):*** Measure use of common assets by application-engineering activities
- ***Configuration Management (CM):*** Reusable assets provide a basis for the identification of configuration items.

- Collaboration
  Collaboration is performed at CMMI® level 2. Amplifications are necessary for the following process areas:
  - ***Requirements Management (RM):*** Maintain bi-directional traceability between software product family and application requirements. Use it for identification of inconsistencies.
  - ***Project Planning (PP):*** Asset life cycles live longer than projects. Synchronise between application- and domain-engineering. Monitor the involvement between domain- and application engineering.
  - ***Project Monitoring and Control (PMC):*** Monitor and control the synchronisation points between domain- and application engineering.
  - ***Configuration Management (CM):*** Change requests regarding reused assets can be relevant for corresponding reusable assets. Synchronise application and domain configuration management.

6.1.3 Level 3: Defined

At this level, processes are aligned over the organisation and engineering is performed in a disciplined way over the organisation. For software product family engineering this means control over variability and reusable assets, both in creation and in use.

- Domain engineering
  Domain engineering is performed at CMMI® level 3. Amplifications are necessary for the following process areas:
  - ***Requirements Development (RD):*** Develop for multiple products in a whole market segment. Define the scope of the family. Identify the products to be built. Identify commonality and variability.
  - ***Technical Solution (TS):*** Variability must be included in operational concepts and scenarios for the domain. Develop platform architecture and technical data package. Consider multiple origins and destinations for interfaces.
  - ***Verification (VE):*** Ensure that application engineering makes the proper intended use of domain deliverables.
  - ***Validation (VA):*** Application-engineering is stakeholder of domain validation process.
  - ***Organisational Process Focus (OPF):*** Include platform for a given domain, procedures of use of this platform, methodologies, reusable components and guidelines. Consider multiple products in a whole market segment. Use the scope of the family.
  - ***Organisational Process Definition (OPD):*** Include platform for a given domain, procedures of use of this platform, methodologies, reusable components and guidelines. Consider multiple products in a whole market segment. Use the scope of the family.
  - ***Organisational Training (OT):*** Add training about products, application processes, and application project groups.
- Application engineering
  Application engineering is performed at CMMI® level 3. Amplifications are necessary for the following process areas:

- ***Requirements Development (RD):*** A single customer is considered. The software product family's variability and capabilities are communicated to the customer. Reuse system-family requirements, bind variability and develop application specific requirements.
- ***Technical Solution (TS):*** Reuse domain assets, bind variability and develop application specific assets. Specialise platform architecture for the application, and use domain technical data package.
- ***Validation (VA):*** Validate both domain and application work products. Staff must be especially trained to know what use they may make of the domain assets. Domain engineering is stakeholder of application validation process.
- ***Organisational Training (OT):*** Add training about platform, asset usage, domain processes, and domain project groups.

- Collaboration
  Collaboration is performed at CMMI® level 3. Amplifications are necessary for the following process areas:
  - ***Requirements Development (RD):*** Identify application requirements as potential software product family requirements.
  - ***Technical Solution (TS):*** Determine selection criteria for and coordinate the inclusion within domain from application. Communicate existing and planned application and domain assets. Identify application assets as potential domain assets. Coordination about make, buy and reuse decisions.
  - ***Product Integration (PI):*** Maintain a roadmap of future products and product enhancements. Determine the actual transfer protocol of deliverables and the timing of the product transfers. Support integration between domain and application engineering.
  - ***Verification (VE):*** Develop the domain verification environment, procedures and criteria concurrently and iteratively with the application verification environment. Communicate verification results and corrective actions between domain- and application engineering. Share policy of planning between domain- and application engineering.
  - ***Validation (VA):*** Develop the domain validation environment, procedures and criteria concurrently and iteratively with the application validation environment. Share policy of planning between domain and application engineering.
  - ***Organisational Process Focus (OPF):*** Determine the organisation's process performance objectives over the whole family. Synchronise action plans between domain and application engineering.
  - ***Organisational Process Definition (OPD):*** Assign responsibilities that cover several projects and products.
  - ***Integrated Project Management (IPM):*** Communicate existing and planned application and domain assets. Identify application assets as potential domain assets.
  - ***Risk Management (RSKM):*** Risk management strategy and risk mitigation plans cover both application- and domain-engineering.
  - ***Decision Analysis and Resolution (DAR):*** Ensure that alternative solutions evaluation covers aspects from both the applications and the domain.

6.1.4 Level 4: Quantitatively managed

At this level, processes are managed and engineering is measured within the organisation. For software product family engineering this means quantitative control over variability and reusable assets, both in creation and in use.

- Domain engineering
  Domain engineering is performed at CMMI® level 4, and software product family processes of level 3 are performed. Amplifications are necessary for the following process areas:
  - ***Quantitative Project Management (QPM):*** Consider in the statistics the related application engineering sub-processes.

- Application engineering
  Application engineering is performed at CMMI® level 4, and software product family processes of level 3 are performed. Amplifications are necessary for the following process areas:
    - ***Quantitative Project Management (QPM):*** Consider in the statistics the related domain engineering sub-processes.
- Collaboration
  Collaboration is performed at CMMI® level 4, and software product family processes of level 3 are performed. Amplifications are necessary for the following process areas:
    - ***Quantitative Project Management (QPM):*** Measure the dependencies and the behaviour of the synchronisation activities between application- and domain engineering. Communicate influences between application- and domain engineering. Negotiate improvement actions on project performance dependent on other projects. Coordinate stakeholder identification over application and domain projects.

6.1.5 Level 5: Optimising

At this level, processes are continuously optimised for their effectiveness for the organisation. For software product family engineering, this means a collaborated improvement of domain- and application-engineering.

- Domain engineering
  Domain engineering is performed at CMMI® level 5, and software product family processes of level 4 are performed
- Application engineering
  Application engineering is performed at CMMI® level 5, and software product family processes of level 4 are performed
- Collaboration
  Collaboration is performed at CMMI® level 5, and software product family processes of level 4 are performed

6.2 Summary

The evaluation levels for the process dimension are based on CMMI®: Initial, Managed, Defined, Quantitatively managed and Optimising. At the initial level, there are no software family processes available. Domain engineering and collaboration are almost absent. The process dimension of the FEF deals with the following aspects:

- Domain engineering
  From being absent, this grows to be the dominating process. It starts with the determination of commonality, variability the reusable platform and ends with the planning and definition of policies for all application-engineering processes.
- Application engineering
  From being the only development process, be it at lower maturity levels this aspect grows to processes of minor importance, but at a high maturity level, reusing not only technical assets, but all kinds of policies as well.
- Collaboration
  From being absent, this becomes an important set of mature processes supporting the coordination between the domain and application engineering processes. They involve activities that align other processes, and that communicate available assets between the different projects.

6.3 Example

The ProtAct company has the following process evaluation:

- Domain engineering
  Domain engineering follows several iterative developments for separate parts of the architecture. A separate development is available for keeping the architecture in shape. The domain-engineering department is at CMMI® level 3. In addition, the do-

main engineering amplifications are performed. Therefore, this leads to an evaluation of level 3: Defined.

- Application engineering
  Application engineering departments usually follow a waterfall model to produce a single application. As such most departments have CMMI® level 3, although some of them are still at level 2. Certain level 3 application engineering activities are performed, such as the technical solution and validation activities. In total the evaluation leads to a level 2: Managed
- Collaboration
  The collaboration processes are mainly organised by domain engineering, and they are performed at CMMI® level 3. They involve the communication of reusable artefacts between application and domain. The feedback from application engineering towards domain-engineering involves problem reports and application priorities. Not all level 3 collaboration activities are performed, for instance within the requirements development process area. Therefore, the evaluation leads to a level 2: Managed.

Combining the results to the lowest of the levels measured, the ProtAct company is evaluated in the process dimension at **P2** Managed.

7 Organisation Dimension

The organisation deals with the actual mapping of roles and responsibilities to organisational structures. Within software product family engineering, the organisation dimension measures the effectiveness of the distribution of domain and application engineering over the organisation. The organisation involves structures and responsibilities for domain and/or application engineering and for supporting and coordinating roles. In particular, the organisation distributes responsibilities between platform (Domain-engineering), corresponding applications (Application-engineering) and internal coordination.
The following aspects play a role in the organisation dimension of software product family engineering.

- Roles & responsibilities
  How well does the organisation manage the distinct responsibilities and relationships occurring in the software product family engineering: undifferentiated or software product family specific roles for engineering.
- Structure
  This deals with the organisation structure that puts the roles and responsibilities into practice. It involves both the primary structure, as shown in the organisation chart and the secondary, often virtual, structure not visible in the organisation chart.
- Collaboration schemes
  This deals with the extent of shared values. It involves the cooperation in primary and secondary organisation structures.

7.1 Levels

The following levels are recognised for the organisation dimension of software product family engineering.

1. Project
2. Re-use
3. Weakly connected
4. Synchronised
5. Domain engineering

In the following sections, each of these levels is discussed in more details.

7.1.1 Level 1: Project

This is the basic level. The organisation is arranged for project based single system engineering. With regard to the organisation concerns, we see the following typical situation:

- Roles & responsibilities

Only the application-engineering roles, which are the traditional software engineering roles, are defined.

- Structure
  The structure is organised around project based single system development.
- Collaboration schemes
  The organisation is internally focused, human resources may be shared among projects, but software assets are usually not shared.

7.1.2 Level 2: Re-use

At this level, the projects drive the reuse in an opportunistic way. Reuse is driven by application projects. First, certain common assets are identified and then re-factored and/or developed as re-usable components between projects.

- Roles & responsibilities
  There are no explicitly defined domain-engineering roles. The application-engineering experts collaborate over project borders to identify and share common assets.
- Structure
  The structure is focussed on doing projects. Certain senior resources are allocated to reusable component identification and development.
- Collaboration schemes
  Collaboration is based on negotiations and information sharing among projects.

7.1.3 Level 3: Weakly connected

At this level, there are one or more separate domain-engineering organisations and multiple application-engineering organisations. There are simple interactions between them at early and late phases of domain and application engineering life cycles.

- Roles & responsibilities
  There are both domain and application engineering roles and responsibilities defined. There are defined responsibilities between separate domain-engineering organisation (for platform assets) and application-engineering organisations (for application assets).
- Structure
  The domain and application roles are distributed over the organisation. There are separate domain-engineering projects. Both domain and application engineering have mostly project-oriented structure.
- Collaboration schemes
  Collaboration is document-based, mostly in exchanging requirements and shared management of change requests / problem reports between domain-engineering projects and several application-engineering projects.

7.1.4 Level 4: Synchronised

At this level, multiple interactions between domain-engineering and application-engineering, institutionalised secondary structure (for early problem prevention, domain-engineering and application-engineering coordinated planning)

- Roles & responsibilities
  There are coordination roles between domain and application engineering, and across domain-engineering organisations. Domain engineering has a major role in software development.
- Structure
  There exists a secondary (virtual) structure needed, incorporating cross-functional teams. The reference architecture identifies the most important development aspects in the organisation. Especially functional domains, embodies in certain variation points determine structure in the organisation.
- Collaboration schemes
  There is a strong cooperation across domain and application engineering projects, cross-functional teams, task force groups, etc. There are regular (virtual) meetings of people carrying collaboration roles

7.1.5 Level 5: Domain engineering

At this level, domain-engineering and application-engineering separation as secondary structure and the functional domains determine the secondary structure.

- Roles & responsibilities
  Domain and application engineering are integrated. The most important roles are the domain-engineering roles. Most people are involved in both domain and application engineering roles
- Structure
  The structure is driven by disciplines in domain engineering. They are not fixedly determined by domain and application engineering.
- Collaboration schemes
  Persons can assume domain and application engineering roles as needed.

7.2 Summary

The evaluation levels for the organisation level are: Project, Re-use, Weakly connected, Synchronised and Domain engineering. At the initial level, there are no organisation structures available for doing software product family engineering. If it is done, it is within a single department, and not visible to the remainder of the company. The organisation dimension of the FEF deals with the following aspects.

- Roles & responsibilities
  From a state where no domain-engineering roles are available, the organisation gets more domain-engineering roles, up to the point these become the most dominant roles in the organisation.
- Structure
  From a state where the structure is defined by a project based application organisation, domain engineering defines more of the structure, first mainly in the secondary organisation, but eventually in the primary organisation.
- Collaboration schemes
  From a situation where there is no organised collaboration, the cooperation moves from an internal focus to a cooperative one.

7.3 Example

The ProtAct company has the following organisation evaluation:

- Roles & responsibilities
  The domain and application engineering roles and responsibilities defined, and people are assigned to these roles and responsibilities. Although there are coordinating roles defined between domain and application engineering, they do not play an important role. In particular, domain engineering has no major role in software development. As a result, the evaluation for this aspect results in a level 3: Weakly connected.
- Structure
  Domain engineering is performed in a department separate from several application-engineering departments. A secondary structure is defined through many cross-departmental groups that have the responsibility for one or more of the collaboration subjects, such as road maps, global architecture issues, interfaces, problem reports and maintenance, and promotion of assets from application to domain. As the primary structure is not driven by domain engineering, the evaluation for this aspect results in a level 4: Synchronised.
- Collaboration schemes
  Collaboration is supported through many cross-departmental groups that have the responsibility for one or more of the collaboration subjects. However, collaboration mainly results in reports between domain-engineering projects and several application-engineering projects. As a result, the evaluation for this aspect results in a level 3: Weakly connected.

Combining the results to the lowest of the levels measured, the ProtAct company is evaluation for the organisation dimension at level **O3**: Weakly connected.

8 Applicability

The FEF evaluation result is a profile consisting of a value between 1 and 5 for each of the BAPO dimensions; see Figure 3. The optimal profile for a given unit is dependent on the situation. In general, having the maximum on all axes may not be optimal. There are several reasons to prefer a less than maximal profile.

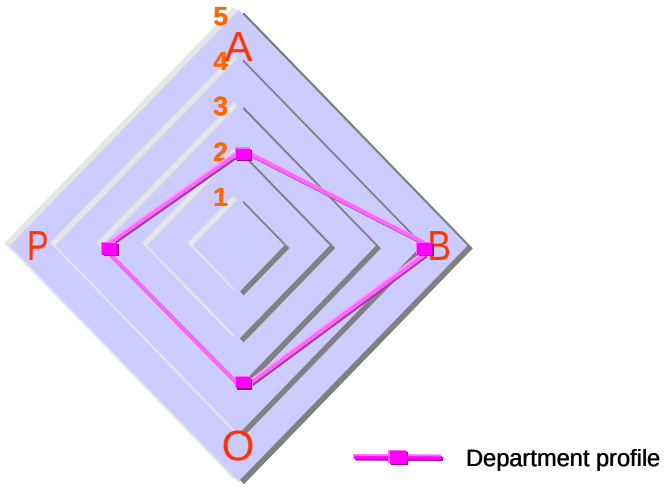


Figure 3: Example company profile

In cases that the domain is not mature enough or the system lifecycle is small, it may be wrong to be maximal, in at least the B and O dimensions, since flexibility may be more important. In case that a company is a business follower, it may afford to have a lower level in the business dimension, than a company that is a business shaper.
There are three main reasons to use the FEF. It can be used to *assess* the organisation, to get information how well the organisation is doing. It can be used as a *benchmark* tool, in comparing organisations, and it can be used as an improvement tool, to plan the improvement of the organisation. Each of the three uses has its own characteristics, which are discussed in the following sections.

8.1 Assessment

In case of an assessment, the organisation, or the assessors, may not know beforehand what is the score, although some indication of the outcome may be available. This indication may be used as a target to which the assessment is done. In each of the 4 dimensions the aspects from level 2 up to the target level the properties assessed. A questionnaire has to be available to perform the actual assessment for each level. A level is reached in a certain dimension if all corresponding elements of the questionnaire are satisfied.
The FEF only covers the software product family evaluation of the software development. This means that single system software engineering practices are not covered. Other means are to be used for assessing them. For the process dimension, an additional CMMI® assessment will cover the normal software process maturity. In the other dimensions, there is no generally established evaluation available. This means that the contents of these dimensions may need improvement, even if a high FEF level is reached. Examples of such practices, not covered by the FEF, are:

- In the Business dimension: The availability of a good business plan.

- In the Architecture dimension: The presence of a good architecture. This may be evaluated through a normal architecture assessment, such as ATAM [6].
- In the Organisation dimension: the presence of training and appraisal schemes for the personnel.

After the assessment is finished, the organisation is responsible for follow-ups. It may decide to improve itself in certain, or all, dimensions, but it may also decide that the present situation is acceptable and that work shall be organised and performed as before.

8.2 Benchmark

In case of a benchmark, an organisation, *A*, wants to compare itself with other organisations *B, C, ….* The FEF can be used as a benchmark tool in cases the evaluation of the organisations *B, C, …* is available. For instance, the evaluation result of *B* and *C* is made available in press. However, it may also be the case that *A, B* and *C* are parts of the same company. In that case, the information may only be available within the company itself.
For benchmarking, the FEF assessment can be used with the FEF results for *B* as target. It will show the differences, where *A* may show to be better, equal or worse than *B* in certain dimensions. In the dimensions where *A* has a higher evaluation value, the assessment may stop at the level of *B*'s evaluation, since only a benchmark is necessary.

8.3 Improvement

In case of an improvement plan an organisation sets a target where it wants to be at some future time. It assesses itself against this target, and it gets results where improvements may be necessary. It may even get feedback to matters where the evaluation is higher than the target was. This may mean that for the moment less attention can be spent in that dimension, and effort may be shifted to improve another dimension.
Improvement may be evolutionary. A target may be to reach a certain profile first, and next a more ambitious profile will be reached. For instance, first go to reach level 3 in all dimensions, and next to reach level 4 in the architecture and organisation dimension.

8.4 Connection to other approaches

An FEF evaluation is similar to a CMMI® assessment. For each of the dimensions, an incremental questionnaire leads to an evaluation result. However, an FEF evaluation does not overlap with a CMMI® assessment. Instead, CMMI® is used to evaluate the single system software engineering process. The process dimension of the FEF only measures the amplifications that are necessary to do software product family engineering. In addition to the CMMI® assessments, the FEF also covers the business, architecture and organisation concerns.
The Framework for Software Product Line Practice℠ (PLP) [4] is a list of necessary practices for software product family development. It does not give a structured framework that can be used to assess, benchmark or incrementally improve organisations. Most of the practices in the PLP are crucial in certain parts of the FEF framework. For instance,

- The software engineering practice areas belong to the architecture dimension
- The technical management practice areas mainly relate to the process dimension
- The Organisational management practice areas belong to the business or organisation dimension.

The Product Line Technical probe (PLTP) and the economic models are mainly useful for organisations that have an FEF level of **B1, A1, P1, O1**. These organisation are considering whether they should move into software product family engineering. The PLTP and the economic models gives them insight on how to start and to move in the direction of **B2, A2, P2, O2**.

8.5 Example

ProtAct is interested in seeing how well they are doing in software product family engineering. They want to use the result in an improvement action, and therefore they need to know what are the best improvement actions to be taken. Initially the expectation is a level of **B2, A3, P2, O3.** The result of execution of the FEF is the following profile: **B3, A4, P2, O3**. This means

that the business and the architecture are better than expected. Because of this, the company decides on an improvement plan to reach level **B3, A4, P3, O3** first, and next to go to level **B4, A4, P4, O4**. This means that first the process has to be addressed, in order to reach level **P3**. This means that each application department has to move to CMMI® level 3, and level 3 collaboration amplifications have to be put in place. For instance, the identification of application assets as potential domain assets has to improve. For the next goal, the business, process and organisation have to be improved to reach level 4. For business this means that several measurements have to be introduced, e.g. for costs and profits of variability has to be measured. For the process dimension this means that CMMI® level 4 has to be reached in the organisation. In addition, the quantitative project management amplifications have to be introduced. For the organisation, this means an improvement of the secondary structure that deals with collaboration over the organisation.
In all these improvement actions, care has to be taken that the architecture stays at level 4. This means that the architecture needs attention to stay healthy and keeps its present qualities. It is expected that this less effort than what is needed in the other dimensions.

9 Complex organisations

Many organisations, which are doing software product family engineering, are complex. The development is distributed over different departments that are located at different sites and even in different time zones. Other companies are involved that are specialised in doing a part of the work, e.g. through outsourcing, or other kinds of agreements between companies. This situation complicates the evaluation of software product family engineering, but in principle, FEF evaluation is possible.
To be practical, the FEF cannot be applied to a complex organisation. It takes too much time to finish the evaluation, and the different parts of the organisation may perform differently, which makes it difficult to end with an evaluation higher than **B1, A1, P1, O1**.
Therefore, the evaluation has to be applied upon units of manageable size. Such units may be departments, divisions, or subcontractors, or even virtual parts of a group of organisations. Restricting the evaluation to such units mean that only parts of the software product family engineering can be evaluated, since the unit that is evaluated may cover only a part of the software product family engineering. Several aspects do not apply completely to certain departments. For instance, it will happen that a single unit is mainly involved in domain, or mainly in application engineering.
Applying the FEF to parts (units) of complex organisations give rise to the following observations, with respect to the BAPO dimensions:

- **B**: Business concerns involve business relations internal to the complex organisation as well. In particular the business relationships of the department that is assessed with the remainder of the software product family engineering parts of the organisation. External business concerns mainly apply if the given part of the organisation is dealing with that aspect.
- **A**: Architecture concerns apply to that part of the architecture that the specific unit is responsible for, either in creating it, or in using it.
- **P**: Process concerns only apply for those that the unit are performing.
- **O**: Organisation concerns only apply for internal organisation of the unit, and to its role in relationships with other parts of the organisation.

9.1 Example

The ProtAct company has development units in several countries. It is decided to do the FEF assessment to a single department only. It is chosen to first assess a department that is mainly responsible for the domain engineering.

- In the Business dimension: The department is not directly involved in marketing and sales. However, instead it has to do marketing and sales internal to the company, for spreading their solution as the reusable platform that the others should use. Product management restricts itself to product management of the platform that is developed, with as clients the other departments of the organisation. In the evaluation of the department, the following elements will be considered: marketing, sales and product

management determine the features in coordination with the other departments. Budgeting and investments mainly deal with the results of marketing and sales, in getting money for the own organisation to do software family engineering. Of course, the higher management of the company that is responsible for this department is responsible that enough money is available to do domain engineering. The tax system takes into account how many people are working in the domain. It is only a rough measure that is not able to discriminate in good and bad elements of the platform. The vision and business objectives of the department mainly deal with a vision of spreading the platform over the organisation, and how this is communicated to the higher management and the remainder of the organisation. Strategic planning deals with the own roadmaps and how the alignment with the road maps of the other departments in coordinated. As a result, the department is evaluated at level **B3**: Managed.

- In the Architecture dimension: the department is defining the architecture for the complete company. It is measured how well it succeeds in getting their own results being reused, how much the architecture is determining the architecture of the other departments, and whether variability is managed well within the platform. As a total this may lead to an evaluation of **A4**: Derivable variant products.
- In the Process dimension: The department is involved in domain engineering, and in collaboration. These processes can be assessed. As this department is working at CMMI® level 3, and it performs all level 3 amplifications for domain and collaboration, it is evaluated to be at level **P3**: Defined.
- In the Organisation dimension, the department is clearly a domain-engineering department. This means that part of the structure is already defined. However, the relationships with other departments determine how the department should be evaluated. As most of the collaboration is governed by the domain engineering department, in a weakly connected way, the evaluation of the department leads to level **O3**.

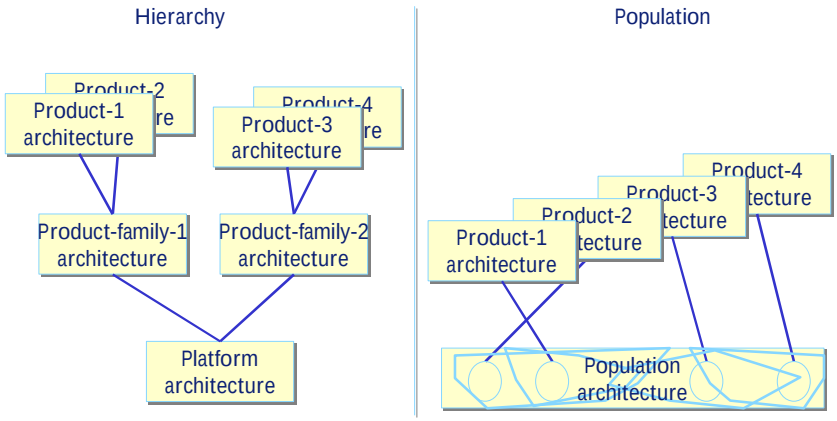


Figure 4: hierarchy and population

9.2 Structured architecture

In many cases, in complex organisations, the software product family is structured (hierarchy), or the family is part of a population; see Figure 4. A hierarchy exists if there are several product families that use a single infrastructure that is modelled as a product family. These families do not reuse software between themselves, not all use this platform. In a population, several product families reuse different parts of a very broad platform. Reuse is within the platform [13]. Usually there are no systems, or not many that use the complete platform.

In a hierarchical situation, application engineering for the generic platform is domain engineering for a single software product family. In the case of populations, it may happen that a department is both responsible for a specific software product family, and for parts of the domain. Such a situation complicates the FEF evaluation, but evaluation is in principle the same. Each unit can be evaluated for each of the different families in which it is involved and in the interplay between them in the complete family structure.The evaluation has separately to be applied upon each of the roles of the units in each development separately. These roles involve parts of domain and/or application engineering. This means that a single organisation may get a separate evaluation for each of its roles. These evaluations may result in different profiles. Similar to the situation in complex organisations this may mean that only parts of the software product family engineering per role can be evaluated. Several aspects do not apply completely to the role in the evaluated department

9.3 Example

The ProtAct company has a hierarchical architecture for its products. It has a department that produces a configurable company platform, PlatPA, for several product families. One such product families, ObserCheap, produces observation rooms and wired camera systems, to be used in small areas only. It uses the PlatPA platform as a basis of its own development. It has a department that produces the product-family architecture for ObserCheap products. The department will be evaluated for it different roles: as application engineering department of the company platform, and as domain engineering unit for the ObserCheap product family.

- In the Business dimension: in the application-engineering role, it is evaluated in its relationship to the PlatPA platform. It is evaluated whether the internal taxation to the PlatPA department is in line with the business interest of the ObserCheap software product family. It is evaluated in its relationship to the PlatPA platform, whether product management of ObserCheap is in alignment with the PlatPA product management. It measures whether the vision and business objectives incorporate the use of PlatPA. In this role, the department is doing reasonably well, and it is measured at a level of **B3**: Managed.
  In the domain-engineering role, it is evaluated in regard of the business relationship to the departments producing the ObserCheap products. Product management has to align with the request from these product development departments. In addition, it has to deal with reimbursing the taxation costs of the use of PlatPA to the development departments. The vision and business objectives are measured in respect to the use of ObserChear platform in the software product family. In this role the department is doing less well, it is evaluated to be at level **B2**: Aware.
- In the Architecture dimension: in the application-engineering role, it is measured how well it reuses the PlatPA platform in its own architecture, and how ObserCheap PlatPA specific elements are in this platform. As the PlatPA platform fits well, the department is evaluated to be at level **A3**: Software platform.
  In the domain-engineering role, the department is defining the architecture for the complete ObserCheap software product family. Since it determines the architecture of all these products largely, the department is evaluated in this role to be at level **A4**: Derivable variant products.
- In the process dimension: The department is assessed to be at CMMI® level 3. In the application-engineering role, it is measured how well the application engineering and collaboration amplifications are done. Collaboration is in this case with respect to the PlatPA department. As such, it performs well, and it is involved in all the amplifications. Therefore, for this role, the department is evaluated to be at level **P3**: Defined.
  In the domain-engineering role, the domain engineering and collaboration amplifications are measured. In this case, collaboration with respect to the ObserCheap product groups is evaluated. As not all these collaboration activities are performed well, e.g., the verification and validation are not performed completely, the department is evaluated to be at level **P2**: Managed.
- In the organisation dimension: in the application-engineering role, it is evaluated how the roles and responsibilities with respect to the relationship with the PlatPA department are assigned, whether its structure fits well to application use of PlatPA, and

whether the collaboration schemes are adequate for the relationship. The evaluation results in level **O3**: weakly connected.

- In the domain-engineering role, the roles and responsibilities, the structure and the collaboration schemes with respect to the ObserCheap development are evaluated. Just as for the other role, the department is evaluated to be at level **O3**: weakly connected.

This means that the department is evaluated at level **B3, A3, P3, O3**, for its application-engineering role. For its domain-engineering role, it is evaluated to be at level **B2, A4, P2, O3**. This may lead to different improvement actions for the two roles. In the application-engineering role, the improvement actions will firstly focus on the architecture dimension. For the domain-engineering role, the improvement actions will initially consider the business and process dimensions.

10 Conclusions

This document gives an overview of the Family Evaluation Framework for the evaluation of software product family development units. The framework is based on a long experience in software product family development, and a major result of a series of European collaboration projects in the ITEA framework: ESAPS [9][19], CAFÉ [10][18] and FAMILIES [11][20]. This framework is improving upon existing approaches in that it systematically distinguishes the four BAPO concerns: Business, Architecture, Process and Organisation. Each of these concerns is evaluated separately, and each leads to its own evaluation value. In the evaluation, only software product family development issues are covered. Other software development issues are treated elsewhere, and are not part of the framework. In particular, the Framework relies on process maturity models, like CMMI®, for normal software development process issues.

Each dimension of the framework has a collection of aspects that are to be considered in the evaluation. Dependent on the evaluation in these aspects a level from 1 to 5 can be obtained. A level 1 means that the aspects are not dealing with software product family engineering. A level 5 means that all the aspects deal largely with software product family engineering. Note that the framework is built in such a way that the level 5 is reachable. This level 5 is not an ideal situation that does not allow improvements any more.

The result of an evaluation is a profile that can be used for several reasons, such as assessing a department, benchmarking with other departments and/or companies or as a starting point for improvement actions.

As organisations doing software product family engineering are usually distributed over several sites, and time zones, the evaluation can be performed for single departments that only perform a part of the software product family engineering. In addition, when the product family is structured, the evaluation may be performed for several roles separately.

This document provides a first public version of the FEF. A steering committee will be established to guarantee the continuous improvement of the FEF, based on company experiences.

11 Literature

[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 SPLC1, Denver, August 2000, 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] Paul Clements, Linda Northrop: “Software Product Lines – Practices and Patterns”, Addison Wesley, 2001.

[5] Piergiorgio DiGiacomo, “Families Deliverable 2.2: CMMI-software product family engineering”, version 0.5, December 2004

[6] Stefan Ferber, Peter Heidl,Pater Lutz, “Reviewing Product Line Architectures: Experience report of ATAM in an Automotive Context”, proceedings Product family Engineering, PFE-4, Springer LNCS 2290, 2001, pp. 364-382

[7] Claudia Fritsch, Ralf Hahn, “Product Line Potential analysis in Software Product Lines”, Proceedings SPLC 2004, Springer LNCS 3154, 2004, pp. 228--237

[8] Mehdi Jazayeri, Alexander Ran, and Frank van der Linden: “Software Architecture for Product Families”, Addison-Wesley, Reading, Massachusetts, 2000.

[9] Frank van der Linden, “Engineering Software Architectures, Processes and Platforms for Software product families”, Proceedings SPLC2, Springer LNCS 2379, San Diego, August 2002, pp. 383-397

[10] Frank van der Linden, “Software Product Families in Europe: The ESAPS and CAFÉ projects”, IEEE Software, July/August 2002, pp. 41-49

[11] Klaus Pohl, Günter Böckle, Frank van der Linden, “Software Product Line Engineering”, Springer Verlag, 2005

[12] M. Paulk et al. “Capability Maturity Model of Software, Version 1.1”. Tech Report CMU/SEI-93-TR24, Carnegie Mellon University, Pittsburgh, 1993

[13] Rob van Ommering, “Building Product Population with Software Components”, PhD. thesis University Groningen, 2004.

[14] Klaus Schmid, “A Quantitative Model of the Value of Architecture in Product Line Adoption”, in Software Product-Family Engineering, Proceedings PFE 2003, Springer LNCS 3014, 2004, pp. 32-43

[15] Jan Gerben Wijnstra, “Critical factors for a successful Platform-Based Product software product family Approach”, Proceedings SPLC2, San Diego, August 2002, Springer LNCS 2379 pp. 68-89

[16] CMMI[SM] for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Continuous Representation (CMMI-software engineering/SW/IPPD/SS, V1.1, Continuous), Technical Report CMU/SEI-2002-TR-011, Carnegie Mellon University, Pittsburgh, 2002

[17] CMMI[SM] for Systems Engineering/Software Engineering/Integrated Product and Process Development/Supplier Sourcing, Version 1.1, Staged Representation (CMMI-software engineering/SW/IPPD/SS, V1.1, Staged), Technical Report CMU/SEI-2002-TR-012, Carnegie Mellon University, Pittsburgh, 2002

[18] CAFÉ Web-site at ESI: http://www.esi.es/en/Projects/Cafe/cafe.html

[19] ESAPS Web-site at ESI: http://www.esi.es/en/Projects/esaps/esaps.html

[20] FAMILIES web-site at ESI: http://www.esi.es/en/Projects/Families/

[21] ITEA web-site: http://www.itea-office.org/

[22] Praise Web-site at ESI: http://www.esi.es/en/Projects/Praise/praiseProject.html