Case study questions for studying software product families

Mikko Raatikainen, Timo Soininen, Tomi Männistö
Software Business and Engineering Institute (SoberIT)
Helsinki University of Technology
P.O. Box 9210, FIN-02015 TKK, Finland
{Mikko.Raatikainen, Timo.Soininen, Tomi.Mannisto}@hut.fi

Abstract. Software product families (SPF) are becoming more commonplace in many industrial organizations due to the increasing variability requirements. One major challenge with SPFs is to understand the challenges companies are facing as well as solutions they have tried out. Feasible tools for gathering such knowledge are qualitative research methods that include the case study method. In this paper, we describe case study questions for a study framework for SPFs called CASFIS. CASFIS builds on top of case study methodology and existing work and understanding on SPFs with the aim of being a solid framework for investigating industrial SPFs. CASFIS has been tested and tried out in several industrial companies and adjusted accordingly

Introduction

A software product family (SPF) is typically understood to roughly consist of common assets that are specifically developed for the SPF and then shared and reused in the development of the product individuals in the SPF [1, 2, 3]. The concept of a SPF has existed for decades [4], but only recently research results have shown that a SPF may provide industrially relevant benefits. The benefits are, e.g., decreased development effort and time-to-market [5, 6] and that a SPF itself and some of the issues it addresses are important and predictive for success of reuse [7, 8, 9]. Examples of the success with reuse within SPFs concern various kinds of assets, such as software architecture, instead of only code [3]. Furthermore, reuse is planned and supported instead of opportunistically assumed reuse just happen [2], and reuse is not assumed to be solely a technical problem, but to also concern other issues, such as the business, process, organizational, and architecture (BAPO) concerns [10]. In addition to reuse, another important aspect of a SPF is systematic management of variability. These research results and issues of SPFs originate from industrial experience including case studies in companies such as Securitas and Axis [11] and CelsiusTech [12], and experience reports in companies such as Philips [13] and Nokia [14]. However, these report only little details on how the research has been carried out.

In addition to SPFs gaining popularity among researchers, recently, more attention is also focused on the research process itself. On the one hand, a need for rigorous research methods in software engineering is identified and, on the other hand, software engineering research is required to focus on practical solutions for real problems in the industry [15, 16, 17]. One class of such methods is qualitative methods, which include qualitative case study method. The qualitative methods are particularly applicable to acquire data of complex real life phenomena [18]. In particular, for case studies based on qualitative data exists also rigorous methodology such as in [19].

This report focuses on CASFIS (**CA**se **S**tudy **F**ramework for **I**ndustrial **S**PF). CASFIS is a tool for empirically studying SPFs in the industry using the qualitative research methods. On the one hand, CASFIS forms one piece in a methodological toolbox of scientific approaches that can be used to study SPFs. On the other hand, CASFIS is an attempt to form a solid basis for further discussion and development of methodologies for studying SPF. Consequently, CASFIS is defined to be used within the context of SPFs by combining our experience of scientific qualitative methods and case study methodology in particular with our experience with SPFs.

In this report, the case study questions of CASFIS are represented. These explicit case study questions are a central element of CASFIS. The rest of this report is organized as follows. In the next section, an overview of CASFIS methodology is presented. After the overview, an overview of the questions and the structured of the questions are given. The case study questions of CASFIS are in the last section.

CASFIS

CASFIS follows a qualitative case study strategy [19, 20]. We have used the steps introduced in [19] to describe the framework. Theses steps are designing case study, preparing data collection, collecting evidence, analyzing the evidence, and composing a report (Table 1). The case study questions are published in this report whereas all the other elements of CASFIS, as outlined in Table 1, are described in more depth in [21].

Designing a study	Preparing data collection	Collecting evidence	Analysis and reporting, and composing a report
-Problem formulation -Study proposition -Case selection -Unit of analysis	-Field procedures -Case study questions	-Interviews -Case study database	(CASFIS does not suggest or require any particular analysis or reporting approach)

The structure of CASFIS follows division to business, artifact, process and organization (BAPO) concerns as introduced in [10]. The business concern refers to how a company makes money with the SPF. The artifact concern refers to technology in a SPF and all kinds of pieces of software assets that the SPF consists of. The process concern refers to activities related to the SPF. The organization concern refers to people and their relations in SPF development. Division into BAPO concerns is similar with, for example, the evaluation framework [22]. However, the term 'architecture' is replaced with the term 'artifact' in order to emphasize that the term referred to several kinds of assets in addition to the software architecture alone.

The focus of CASFIS is on SPF practices rather than software engineering practices in general. The focus in the BAPO concerns is on the artifacts of a SPF and their variability in particular.

CASFIS is a revisited version of the framework that was used in a case study of the state of the practice of SPFs [23] which some results are published in [24, 25] although the name CASFIS was not used at that time. These studies proved that CASFIS is feasible to capture the state of the practice of SPF in a company. The studies also report the actual use of the earlier version of the framework.

Closely related to CASFIS is the software product line questionnaire [6] that is a survey instrument although it consists of mainly the questions. CASFIS differs from the existing frameworks such as the evaluation framework [22] and the SPF probe [3], and actual case studies, such as the above mentioned Securitas and Axis case studies, by describing in detail and making publicly available the guidelines for the research process.

Introduction to the case study questions

CASFIS includes the case study questions as a central element. These questions serve a dual role: On the first hand, the questions are interview questions to be asked from responders, on one hand, the questions serve as a guideline how an interview should proceed and to structure the interview. That is, the interviews follow principles of a semi-structured interview. An interviewer can omit some questions that seem unimportant for the particular SPF, change the wordings and the order of the questions, and ask additional questions for clarification and deepening of the topic under discussion. In addition, an interviewer can ask additional questions in addition the questions in CASFIS. Additional questions can be summarizing comments in a question format to generalize the responses and enable the respondents to correct misinterpretations and misunderstandings, or questions that go deeper to a particular topic of interest, for example. However, the questions also structure an interview. A deviation from the structure may lengthen the study and distort the focus.

The case study questions of CASFIS are mostly open-ended in order to enable true semi-structured interview and to achieve a rich qualitative description. The structure of the questions (Figure 1) is broken down into BAPO concerns. This division was done in order to take into account several points of views to a SPF. Each concern is in its own section. The order of the sections is business, process, organization, and artifacts. This order was selected in order to start from the overview and going later into the detail. That is, the artifacts are handled in most detailed manner. Further, organization follows process because these two are closely related and tied together. However, the order is not fixed, but can be changed if necessary.

For each concern, several aspects are identified that are properties of the concerns. For each aspect is a set of factors that characterize the aspects. The questions measure the factors. If a question includes a figure, the figure appears after the question. In addition, each section concludes with explicit question of what is important for success of a SPF and what are the problems with a SPF. Since each section represent different point of view following the BAPO concern, some questions are repeated in all sections. An outline of concerns and aspects are presented as follows and the questions in the next section.

The business concern covers profile of the company, SPF strategy, and SPF business aspects. Profile of the company includes business characteristics and the application domain factors. Business characteristics illustrate the internal characteristics of the company in terms of size and turnover, for example. The application domain illustrates the environment in for which the SPF is developed such as the kind of products in SPF and whether the software is embedded. SPF strategy includes present and history factors. The present factor characterizes how SPF is perceived from the point of view of business today while the past factor characterizes how the change to SPF has taken place. The SPF business aspect includes SPF quantified and product details factors. SPF quantifies concerns number of products individuals and prices. The product detail deals with detailed characteristics of product individuals in a SPF such as commonalities and variability.

The process concern covers general aspects, SPF development and SPF derivation. The general concern consists of the SPF concepts aspect that deals with understanding of SPF and nature of division of processes to development and derivation phases. The SPF development aspect consist of general, development phase, and maintenance factors. The general factor deals with phases and activities, and organizing them in SPF asset development whereas development phase factors deals with how these activities are performed. The maintenance factor outlines maintenance process of assets such as change requirements. The SPF derivation aspects is consist of the same factors as SPF derivation, except, naturally, these deal with product individuals instead assets.

The organization concern is divided into static and practices aspects. The static aspect deals with existing structure including the stakeholders and organizational model factors. The stakeholders deals with those individual involved in SPF development, whereas the organizational model deals with how these individuals are organized. The practices aspect concerns how organization operates consisting of responsibilities factor that deals with who is responsible of what in the SPF.

The artifact section covers concepts, and variability and evolution aspects. Concepts aspect deals with concepts of main artifacts in a SPF consisting of factors the concept of SPF, concept of products individuals, concept of SPF architecture and shared assets, and concept of SPF architecture and shared assets in a product individual. The concept of SPF deals with how a SPF is understood as a whole, while the concept of products individuals aspect deals with product individual that the SPF includes. The last two aspects deal with what kind of pieces the SPF consists of and how these pieces are reuses in product individual, respectively. Variability and evolution aspect consists of the variability and evolution in the SPF, resolving variability in the SPF, variability in the SPF architecture, and resolving variability in the SPF architecture and shared assets factors. These factors in the order introduced deal with how variability and evolution are perceived in the SPF, handled and mean in product individuals, embedded in the shared assets and SPF architecture, and handled and resolved when the shared assets and SPF architecture are used in product individuals.

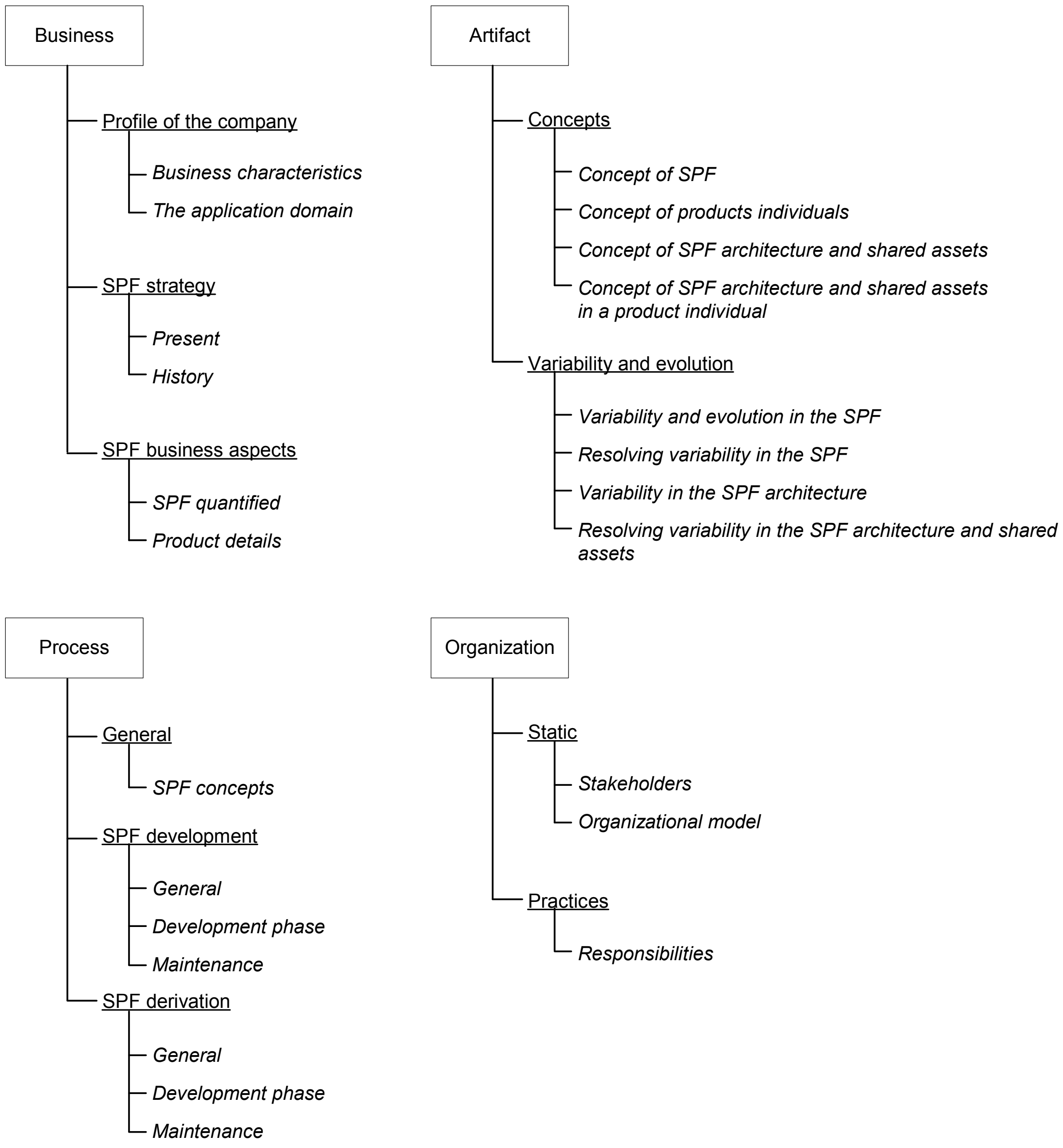


Figure 1 Organization of the questions

The case study questions

1 Software product family business

1.1 Profile of the company

1.1.1 Business characteristics

Q:1.1.1.1 How much was the last annual turnover of the company?

Q:1.1.1.2 The company can focus on areas such as product development, product installation, education, consulting etc. In which areas does the company operate? In which areas is the focus of the company in the sense from where comes the money?

Q:1.1.1.3 How many employees does the company employ? How many employees work with the software product family?

Q:1.1.1.4 Does the company use formal or external verifications or validations, such as CMM or ISO9000?

1.1.2 Application domain

Q:1.1.2.1 What software products and software product families does the company have?

Q:1.1.2.2 The company can operate in market areas such as telecommunications, enterprise information systems, embedded software. In which market areas does the company operate?

Q:1.1.2.3 Is the product a physical one, which includes software, e.g., as embedded? If so, how important role does the software part have if compared to, e.g., mechanical and electronic parts in cost and profit causing sense?

Q:1.1.2.4 Maturity can be defined as a property of an industry, a market, or a product that is no longer subject to great expansion of users or development of the core technology. How mature are the application domains, measured in amount of growth in the markets and changes in the core technology from the viewpoints of the products the company develops and markets in general.

1.2 The software product family strategy

1.2.1 Present

Q:1.2.1.1 How does the company understand and define a software product family?

Q:1.2.1.2 Does the company have data of efficiency of the software product family approach based on quantitative measures such as costs, profits, lead times, or other factors? If this has been measured, how is the measurement done and what are the results of the measurement?

Q:1.2.1.3 Does the company have data of efficiency of the software product family approach based on, qualitative measures such as customer satisfaction? If this has been measured, how they are measured and what are the results?

Q:1.2.1.4 How do sales and marketing influence on the development of the software product family?

Q:1.2.1.5 What are the strengths, weaknesses, opportunities, and threats (SWOT) of the software product family approach?

Strengths	Weaknesses
Opportunities	Threats

1.2.2 History

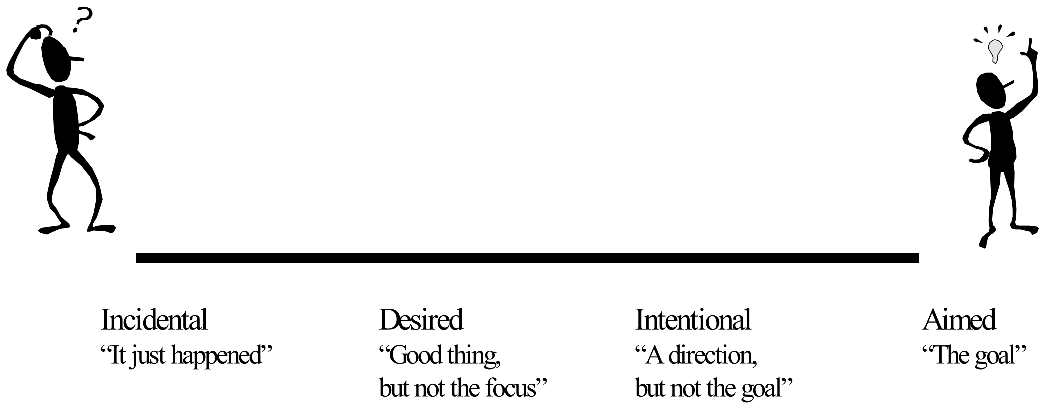
Q:1.2.2.1 Has there been a clear strategic change towards the software product family approach or has the change taken place slowly?

Q:1.2.2.2 When did the change to the software product family approach start?

Q:1.2.2.3 How did the company operate before the software product family?

Q:1.2.2.4 Did the software product family replace existing products?

Q:1.2.2.5 The software product family approach can be more or less goal oriented meaning that the approach is, e.g., incidental, desired, intentional or aimed. How goal oriented has the adaptation of the software product family approach been on this scale?


Incidental
“It just happened”
Desired
“Good thing, but not the focus”
Intentional
“A direction, but not the goal”
Aimed
“The goal”

Q:1.2.2.6 What led the company to adapt the software product family approach?

1.3 The software product family oriented business.

1.3.1 The software product family quantified

Q:1.3.1.1 What are the sales volumes of product individuals in terms of annually sold different variants and number of each sold variant?

Q:1.3.1.2 How much does a product individual cost at the minimum, typically and maximum?

1.3.2 Product details

Q:1.3.2.1 A feature is a logical unit of behavior that is specified by a set of functional and quality requirements [2]. From the customer or sales viewpoint, what are important features in the software product family?

Q:1.3.2.2 From the customer or sales viewpoint, are some of performance, security, modifiability, portability, and operational reliability important quality characteristics, and are there other quality characteristics of the products?

Q:1.3.2.3 Commonality means that something is shared or duplicated between the product individuals. From the customer or sales viewpoint, what are the commonalities in product individuals?

Q:1.3.2.4 Differences are opposite to commonalities, i.e. characteristics that make product individuals different. From the customer or sales viewpoint, what are the differences between product individuals?

Q:1.3.2.5 After a product individual is delivered to a customer, does the company, customer, or some other company change characteristics of the product individual, for example, by adding features? If characteristics are changed, what are changed characteristics and who does it?

Q:1.3.2.6 What is most important to success in software product family development?

Q:1.3.2.7 What are the biggest problems and needs when applying the software product family approach? What would the company need?

2 Processes

2.1 General

2.1.1 Software product family concepts

Q:2.1.1.1 How does the company define and understand a software product family?

Q:2.1.1.2 A software product family is defined many ways. Jan Bosch [2] defines: "[A software product family] consists of a product line architecture, a set of reusable shared assets and a set of products derived from shared assets." Does the company understand the software product family concept similarly as Bosch? If concept is understood differently, what are the differences?

Q:2.1.1.3 Did some kind of changes in software development processes take place at the time the company adopted the software product family approach? If changes took place, what were changed?

Q:2.1.1.4 Processes in reuse oriented software development within organization are typically divided into two processes. In one process, assets are developed for reuse whereas in other process the assets are reused to develop product individuals for customers. However, these processes may be organized in different ways. For example, both processes might be constantly ongoing in the company, processes can alternate such that when assets are being developed for reuse products are not development whereas when products are being developed assets are not developed. Are there these two processes? If there are the two processes, how do these two processes take place?

Q:2.1.1.5 If there are the two processes, are these processes explicitly identified somewhere, e.g., in a process model?

Q:2.1.1.6 What kind of meetings does the company hold during the software product family development?

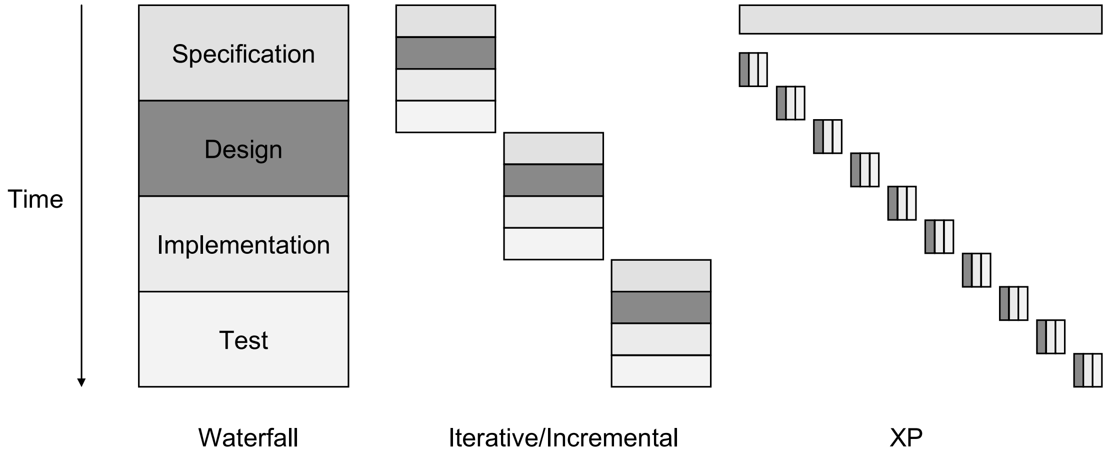
Q:2.1.1.7 Are there meetings in which are present employees that develop the software product family as well as people who derive product individuals? If such meetings are hold, how often do they take place and who are present in them?

2.2 Software product family development

2.2.1 General

Q:2.2.1.1 What kinds of processes and activities are included in production of reusable assets in the software product family, i.e. development?

Q:2.2.1.2 There are several types of process models for organizing specification, design, implementation, and test phases[26]. In the waterfall model, the phases follow each other in a sequence. In the incremental model, the phases are like in the waterfall model but there are several iterations. In the agile models, such as in the extreme programming, the phases are not separated and are rather short. What kind of process model does the company have for software product family development? Which one of the above-mentioned models is closest to the model that the company uses?



Q:2.2.1.3 Does the software product family development contain other essential phases in addition to the above mentioned specification, design, implementation, and test phases?

Q:2.2.1.4 In the development process might be constraints such as time to be completed, implementation quality of wanted features, scarce resources and in addition several others. What are the dominating constraints?

Q:2.2.1.5 Is the development process documented, e.g., in a process guide or model? If the process is documented, what kind of documentation is used?

Q:2.2.1.6 If the process is documented, how well the documented process is followed?

Q:2.2.1.7 Is the development process periodical or continuous? If the process is periodical, how often do the periods take place and how long do they take?

2.2.2 Development phase

Q:2.2.2.1 What does the company do to capture the requirements of the software product family?

Q:2.2.2.2 How does the company capture the requirements of the software product family?

Q:2.2.2.3 What sources are used to capture the requirements of the software product family?

Q:2.2.2.4 Characteristic, such as features, have to be selected either to be included in the software product family or to be excluded from the software product family. How does the company do this selection?

Q:2.2.2.5 Is the selection process documented? If the selection process is documented, how is it documented?

Q:2.2.2.6 Is the selection done in certain intervals? If it is done in certain interval, what kind of intervals is it done?

Q:2.2.2.7 When does the selection process take place?

Q:2.2.2.8 Who is responsible for the selection process?

Q:2.2.2.9 What kind of documentation is produced in the selection?

Q:2.2.2.10 What kind of process did the company follow to design, construct, and test the software product family architecture?

Q:2.2.2.11 Is the software product family architecture design, construction, and test process documented? If the process is documented, how is the process documented?

Q:2.2.2.12 Who is responsible for designing, constructing, and testing the software product family architecture?

Q:2.2.2.13 What kind of process does the company follow to design, construct, and test the shared assets?

Q:2.2.2.14 Are the shared assets design, construction, and test process documented? If the process is documented, how is the process documented?

Q:2.2.2.15 Who is responsible for designing, constructing, and testing the shared assets?

Q:2.2.2.16 If the company produces or has earlier experience in producing one of kind products, what is different in the architecture and shared assets design for the software product family when compared with one of kind products?

Q:2.2.2.17 How are software product family architecture and shared assets tested?

2.2.3 Maintenance

Q:2.2.3.1 Required characteristics of the software product family change and on the other hand all of them may not be captured once or they are not correctly handled, e.g., wrongly prioritized. How are required changes handled?

Q:2.2.3.2 If during product individual derivation is constructed software that could become a part of shared asset, how is this handled?

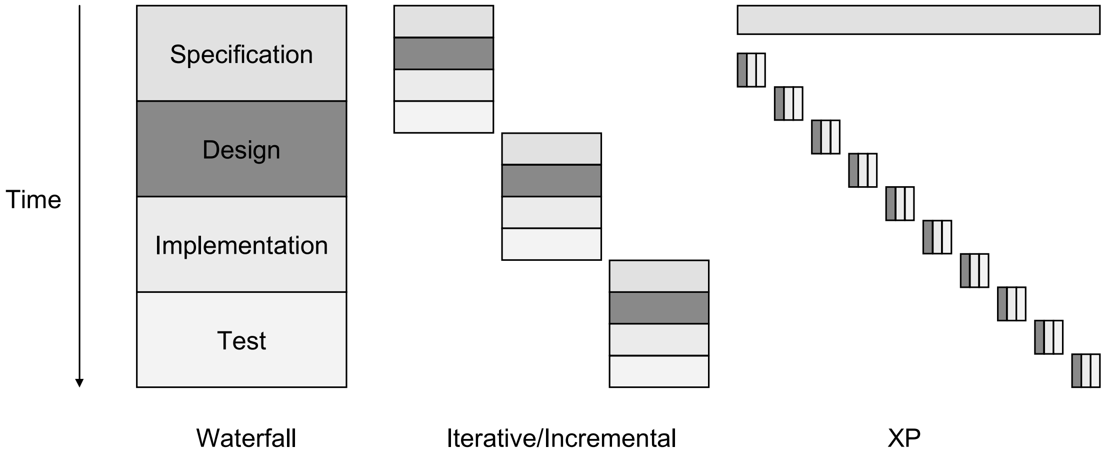
Q:2.2.3.3 What are the biggest problems and need when developing the software product family and what kind of solutions would the company need?

2.3 Product derivation

2.3.1 General

Q:2.3.1.1 What kinds of processes and activities are included in production of a product individual in the software product family, i.e. derivation?

Q:2.3.1.2 There are several types of process models for organizing specification, design, implementation, and test phases[26]. In the waterfall model, the phases follow each other in a sequence. In the incremental model, the phases are like in the waterfall model but there are several iterations. In the agile models, such as in the extreme programming, the phases are not separated and are rather short. What kind of process model does the company have for product individual derivation? Which one of the above-mentioned models is closest to the model that the company uses?



Q:2.3.1.3 In the derivation process might be constraints such as time to be completed, implementation quality, scarce resources, and in addition several others. What are the dominating constraints?

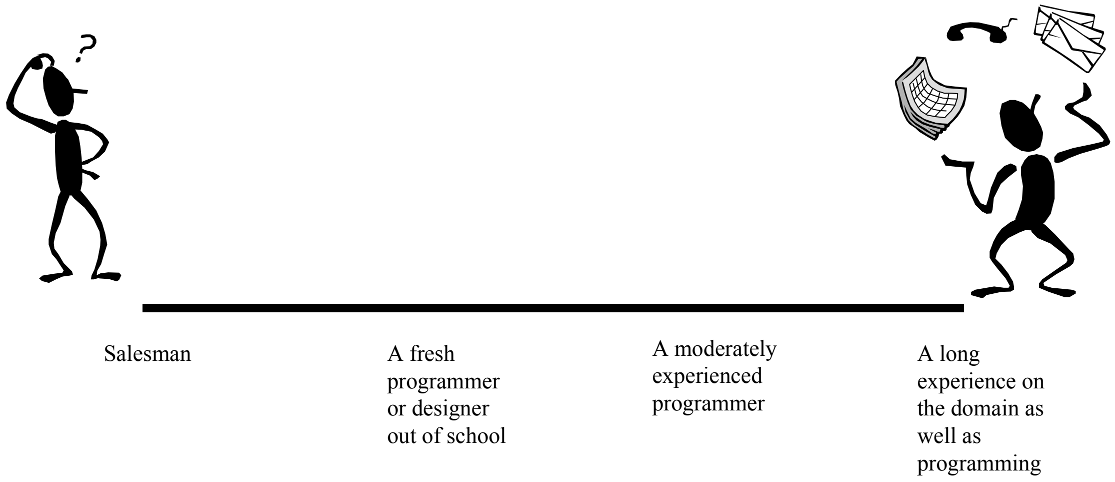
Q:2.3.1.4 Is the derivation process documented, e.g., in a process guide or model? If the process is documented, what kind of documentation is used?

Q:2.3.1.5 If the process is documented, how well the documented process is followed?

Q:2.3.1.6 How long does the derivation process take, from the first thing done, e.g. from requirement specification, to delivery?

Q:2.3.1.7 Does the company establish a specific project for each derivation?

Q:2.3.1.8 How complex is the derivation in each phase: requirement specification, design, and instantiation? Are some special skills needed? Can a sales representative, a fresh programmer out of school do it or does it need a moderately experienced programmer, experienced software designer, or even a long experience on the domain as well as software design?


Salesman
A fresh programmer or designer out of school
A moderately experienced programmer
A long experience on the domain as well as programming

2.3.2 Development phase

Q:2.3.2.1 What does the company do to capture customer requirements for a product individual?

Q:2.3.2.2 How does the company capture other than customer requirements such as local regulations?

Q:2.3.2.3 Who captures the requirements?

Q:2.3.2.4 How are the captured requirements documented?

Q:2.3.2.5 How is validated that requirements are fulfilled in a product individual?

Q:2.3.2.6 What kind of design and implementation process is needed for a product individual?

Q:2.3.2.7 In the derivation may be activities in construction such as parameter setting, programming, copy-pasting code. What does the company do in the product derivation phase and how is this done?

Q:2.3.2.8 How does the company test a product individual?

Q:2.3.2.9 How does the company document testing?

Q:2.3.2.10 Testing of a product individual can be included in the software product family, e.g., in form of test plans or cases in assets or already tested software components. How does the company take advantages of the software product family in testing?

2.3.3 Maintenance

Q:2.3.3.1 Is a product individual possible to change after delivery to customer, e.g., by adding features, components or configuration? If such a change is possible, what is changed and how is the change done?

Q:2.3.3.2 How does the company assure that a change after delivery work as intended?

Q:2.3.3.3 How bugs in a product individual are taken care of?

Q:2.3.3.4 A product individual may need deficiency or bug fixes. If a product individual is complemented with such new things, what does the company do to assure that the new things work and how is this done?

Q:2.3.3.5 What are the biggest problems and needs when deriving a product individual? What kind of solutions would the company need?

3 Organization

3.1 Structure

3.1.1 Stakeholders

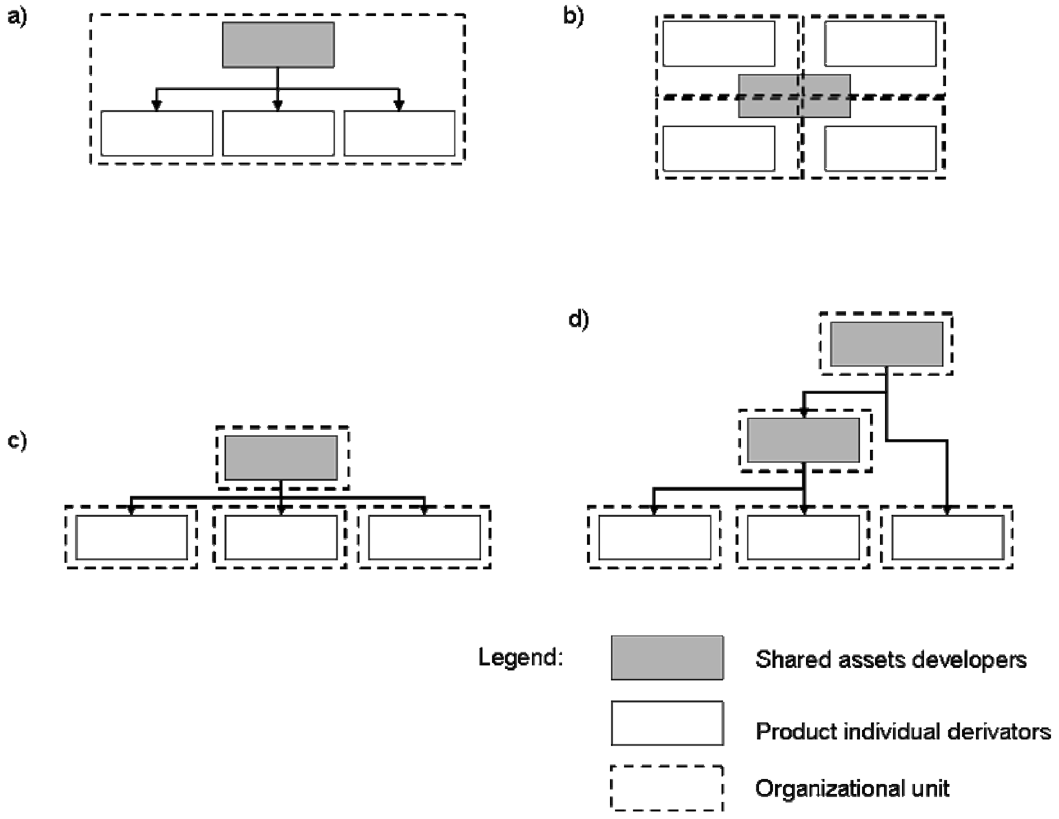
Q:3.1.1.1 Who are the stakeholders that influence on the software product family? What kind of influence does each of them have?

Q:3.1.1.2 How strict are the roles in the organizational structure in software product family development? Does everybody do everything or is there a clear separation, e.g., between developers of software product family and derivators of a product individual?

Q:3.1.1.3 Who dominates the software product family centric decision-making in the company? Are they software product family producers, architect, derivators of product individuals of the software product family, marketing staff, or some others?

3.1.2 Organizational model

Q:3.1.2.1 Bosch [2] identifies four different structures for organizing the software product family development. The first is the development department model that has no clear separation of software product family assets developers and product individual derivators. The second is the business unit model, in which each business unit is specialized around one subset of product individuals. These business units develop and evolve the shared assets without separate organizational unit for taking care of them. The third is the domain engineering unit model, in which one unit, called domain engineering unit, is responsible for developing and evolving the shared assets and separate business units produce product individuals. The fourth is hierarchical domain engineering unit that is similar with domain engineering unit model, but several domain engineering units are organized hierarchically to take care of the assets. Does the company apply some of these models or does it apply some other model?



Q:3.1.2.2 Did the company reorganize the organization at the time the software product family approach was adopted? If there was reorganization, what was reorganized?

Q:3.1.2.3 Has the organizational model changed during software product family development?

3.2 Practices

3.2.1 Responsibilities

Q:3.2.1.1 Who is responsible for the management of the software product family and where in the organization structure does he reside?

Q:3.2.1.2 Who is responsible for the product individual derivations and where in the organization structure does he reside?

4 Product

4.1 Concepts

4.1.1 The concept of software product family

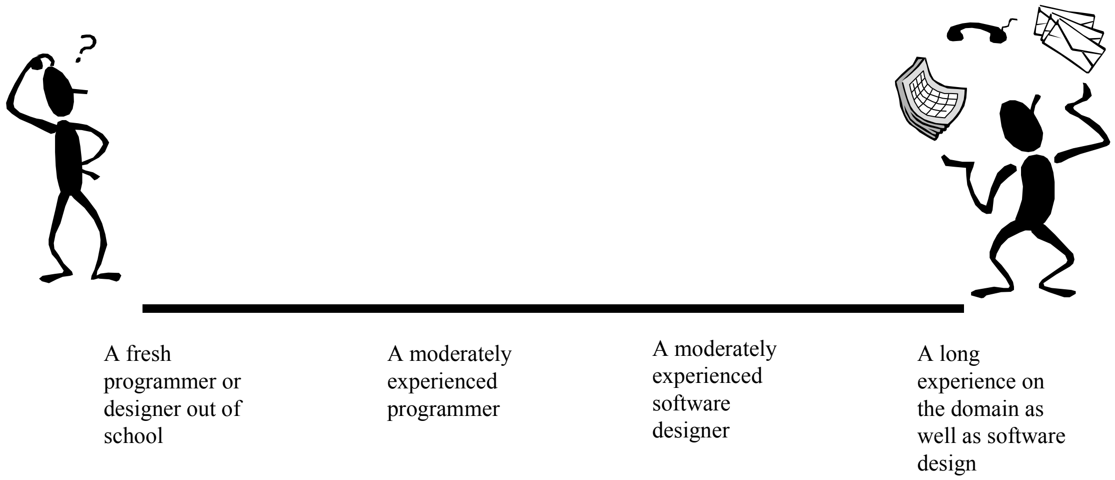
Q:4.1.1.1 How does the company understand or define a software product family?

Q:4.1.1.2 What software product families does the company develop?

Q:4.1.1.3 Was the software product family based on existing products, or was it developed for new products?

Q:4.1.1.4 Why does the company have the software product family?

Q:4.1.1.5 How complex is it to develop a software product family? Are some special skills needed to develop a software product family? Can a fresh programmer or designer out of school do it or does it need a moderately experienced programmer, experienced software designer, or even a long experience on the domain as well as software design?


?
A fresh programmer or designer out of school
A moderately experienced programmer
A moderately experienced software designer
A long experience on the domain as well as software design

Q:4.1.1.6 What are the biggest problems and challenges in developing a software product family?

4.1.2 Concept of a product individual in the software product family

Q:4.1.2.1 What are product individuals in the software product family?

Q:4.1.2.2 What information does the company have for deriving a product individual?

Q:4.1.2.3 How does the company document and store the information for deriving a product individual?

Q:4.1.2.4 What would the company need in order to be able to take advantage of the software product family better?

Q:4.1.2.5 What are the biggest problems and challenges in deriving product individuals?

4.1.3 The concept of a software product family architecture and shared assets

Q:4.1.3.1 How does the company define and understand the software product family architecture?

Q:4.1.3.2 One definition of software architecture is “a structure or structures of a system, which comprise software components, externally visible properties of those components, and relationships among them” [27]. Is this definition applicable for the company, does the definition lack something, or is there something that should not be included in the architecture definition?

Q:4.1.3.3 Does the software product family contain a common architecture?

Q:4.1.3.4 Does the software product family architecture and single system architecture differ from each other? If these architectures differ from each other, what are the differences?

Q:4.1.3.5 The software product family architecture may consist of entities such as interfaces, connectors, and components. What entities does the software product family architecture consist of? How many of each entity are there in the software product family architecture? How large are these entities?

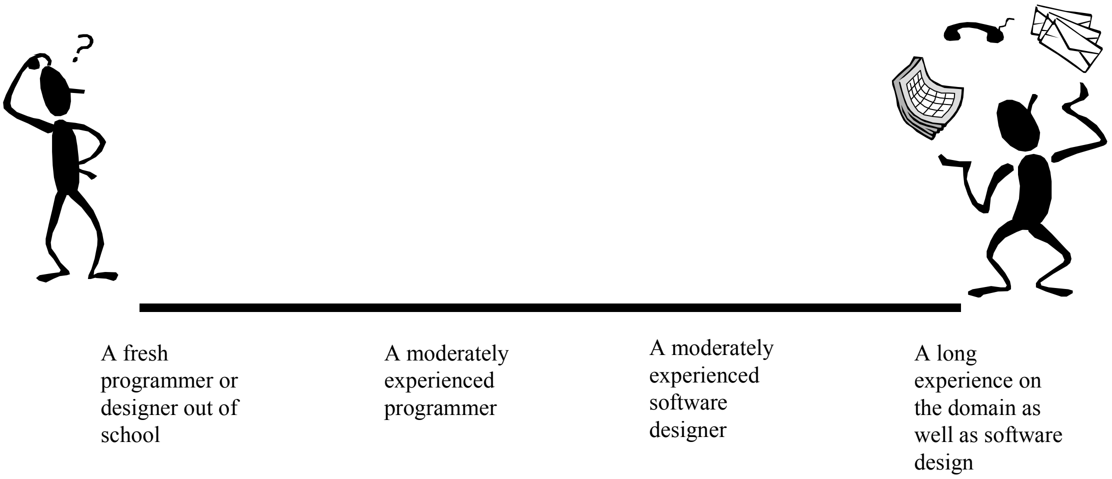
Q:4.1.3.6 What is documented about the software product family architecture?

Q:4.1.3.7 How is the software product family architecture documented?

Q:4.1.3.8 Does the software product family architecture documentation contain different views? If documentation contains different views, what are the views and how are they documented?

Q:4.1.3.9 What kind of knowledge does designing the software product family architecture require?

Q:4.1.3.10 How complex is to design the software product family architecture? Can, e.g., a fresh programmer or designer out of school do it, or does it need a moderately experienced programmer, experienced software designer, or even a long experience on the domain as well as software design?


A fresh programmer or designer out of school
A moderately experienced programmer
A moderately experienced software designer
A long experience on the domain as well as software design

Q:4.1.3.11 One class of shared assets are reusable software entities that can be, for example, in form of modules, components, or libraries. In what form are the reusable software entities in the company?

Q:4.1.3.12 What are examples of reusable software entities in the software product family?

Q:4.1.3.13 Different kinds of things of reusable software entities may be documented such as features, behavior, functional requirements, and non-functional or quality requirements. What is documented about a reusable software entity and how is it documented?

Q:4.1.3.14 In addition to reusable software entities, a software product family can contain other assets. Examples of such assets are models, requirement specifications, functional specifications, feature specifications, interface specifications, and test cases. What kinds of assets are there in the software product family and in what form are they?

Q:4.1.3.15 What is documented of the shared assets and how is it documented?

Q:4.1.3.16 What are examples of shared assets in addition to reusable software entities?

Q:4.1.3.17 If measured, e.g., in quantities of shared assets, lines of source code or amount of documentation, how much does the company have each kind of shared assets in the software product family?

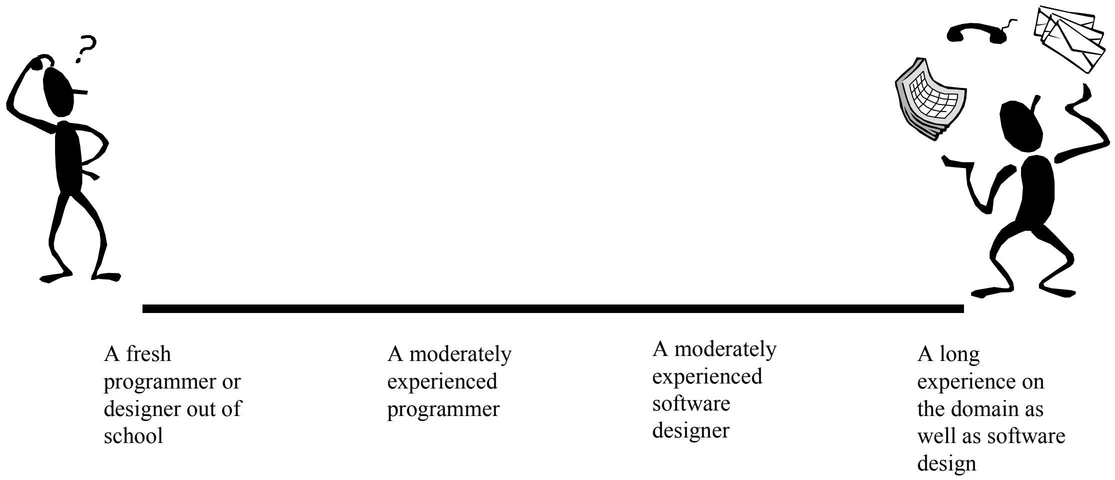
Q:4.1.3.18 What is the size of each type of shared asset in minimum, maximum and typically, when measured e.g. in KLOC, lines of documentation, or work needed to produce it?

Q:4.1.3.19 Does the company have shared assets that are used in all product individuals? If there are such assets, what are they?

Q:4.1.3.20 Are the shared assets developed to be used only in one specific software product family or are the shared assets designed such that they are or will probably be used in several different software product families?

Q:4.1.3.21 How does the company manage shared asset and their documentation, i.e. where are they stored and how are they retrieved for usage?

Q:4.1.3.22 How complex is it to design and implement shared assets? Are some special skills needed? Can a fresh programmer or designer out of school do it or does it need a moderately experienced programmer, experienced software designer, or even a long experience on the domain as well as software design?


A fresh programmer or designer out of school
A moderately experienced programmer
A moderately experienced software designer
A long experience on the domain as well as software design

Q:4.1.3.23 Interface is a point of outside interaction or communication of a reusable software entity. What kind of interface is in a reusable software entity?

Q:4.1.3.24 Does a reusable software entity have several interfaces? If an entity has several interfaces, why does the entity have several interfaces?

Q:4.1.3.25 A provided interface means that a component provides certain functionality whereas required interface means functionality that a component needs. An example is an HTTP component, which provides interface to retrieve web pages whereas it requires TCP/IP interface. Does the company make difference between required and provided interfaces of a reusable software entity?

Q:4.1.3.26 What does the company documented of an interface and how is it documented?

Q:4.1.3.27 How does the company document the syntax or the signature of an interface?

Q:4.1.3.28 How does the company document the behavior or semantics of an interface?

Q:4.1.3.29 If the company makes difference between required and provided interfaces documented, what is different in documenting?

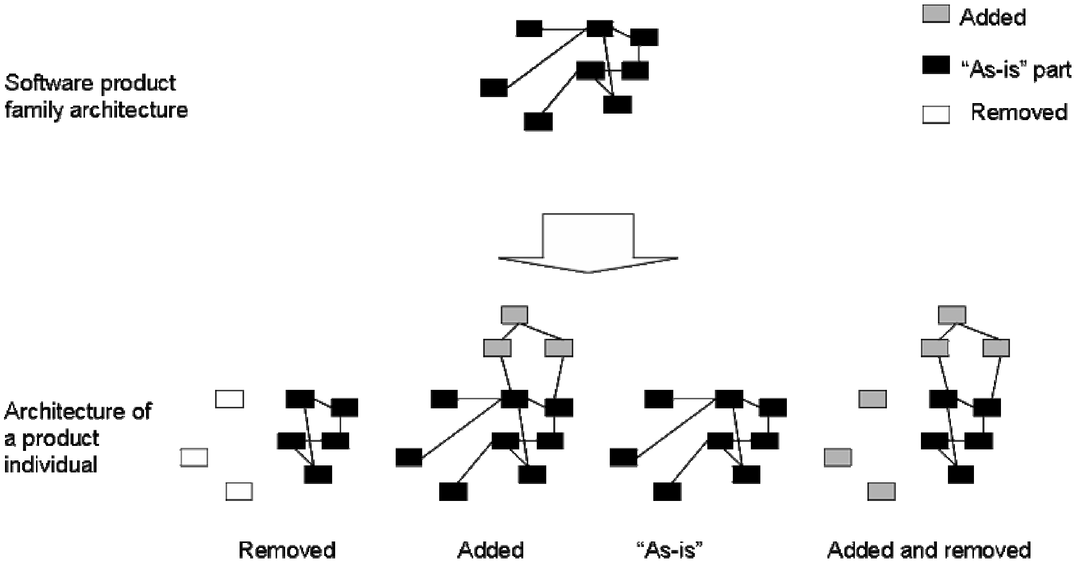
Q:4.1.3.30 What is most challenging in developing a software product family architecture and shared assets?

Q:4.1.3.31 What are the problems and needs when developing a software product family architecture and shared assets?

4.1.4 Concept of architecture of a product individual

Q:4.1.4.1 What are the similarities and differences between architecture of the software product family and architecture of a product individual?

Q:4.1.4.2 Is the software product family architecture used 'as-is' for product individuals, is it only a part of the product individuals in the software product family which is extended, are entities removed, or are entities both removed and added?


Added
"As-is" part
Removed
Software product family architecture
Architecture of a product individual
Removed
Added
"As-is"
Added and removed

Q:4.1.4.3 How many entities are in the architecture typically, minimally and maximally? How many of these entities are based on the software product family architecture?

Q:4.1.4.4 Is the architecture of a product individual documented? If the architecture of a product individual is documented, how is it documented?

Q:4.1.4.5 How much typically, minimally and maximally are there shared assets in a product individual, broken down as the company does, e.g., to percents of assets of a product individual, lines of source code, or number of shared assets?

Q:4.1.4.6 Does the company document what assets are used in product individuals? If use of assets is documented, how is it documented?

Q:4.1.4.7 How does a developer find right shared assets and information about shared assets?

Q:4.1.4.8 Different kinds of tools and techniques can be used for deriving a product individual such as generators, copy-paste and build tools. What tools and techniques does the company use in derivation?

Q:4.1.4.9 If the company cannot fulfill customer's requirements totally with the assets in the software product family, the company may end up doing some special customization by writing product specific code. How often does this take place?

Q:4.1.4.10 What is the reason for the product specific code?

Q:4.1.4.11 How much is in a product individual product specific code, when measured e.g. in extra work, needed time, or KLOC?

Q:4.1.4.12 How much does the software product family, for example, in form of reuse save efforts in the producing of a product individual, approximated e.g. by costs, person-hours or amount of code?

Q:4.1.4.13 What are the problems and needs when deriving a product individual, and the shared assets and architecture for the product individual?

4.2 Variability and Evolution

4.2.1 Variability and evolution in the software product family

Q:4.2.1.1 The term variability is used for differences between product individuals in certain point of time. The term evolution is used for changes in time in the software product family. Variability can concern e.g. functionality, user interface, quality, performance, platform, or underlying hardware. What varies in the software product family?

Q:4.2.1.2 What are the reasons for variability in the software product family?

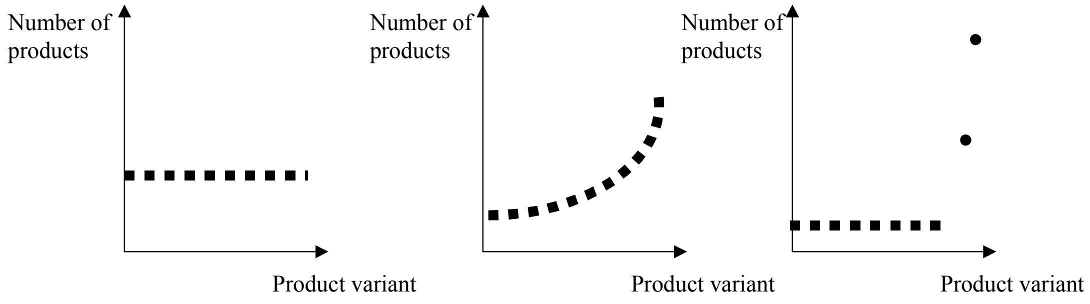
Q:4.2.1.3 Is there variability in the software product family that the company has not yet taken advantage of systematically, but knows it exists or could be taken advantage of? If such variability exists, why has the variability or commonality not been exploited?

Q:4.2.1.4 The term evolutionary software product family means that the company starts to develop software product family such that when commonality is noticed between any products, it is added to a common base. The term revolutionary software product family means that large effort is put to produce a common base that serves as a base from which products are derived. Which approach did the company use?

4.2.2 Resolving variability in the software product family

Q:4.2.2.1 How many different product individuals in the software product family are possible in theory? How many different product individuals have been actually derived?

Q:4.2.2.2 How do the product individuals in the software product family distribute in quantities of different individuals and quantities of copies of an individual?



Q:4.2.2.3 Commonality means that something is shared or duplicated between the product individuals in the software product family. What are commonalities in all product individuals?

Q:4.2.2.4 Are there such commonalities that are common for some, but not all product individuals? If there are, what are such commonalities?

Q:4.2.2.5 What are differences between product individuals?

Q:4.2.2.6 What is unique in each product individual?

Q:4.2.2.7 How many decisions have to be made for resolving variability of a product individual typically, minimally and maximally?

Q:4.2.2.8 What are the biggest problems and challenges in variability in the software product family?

4.2.3 Variability in the software product family architecture and shared assets

Q:4.2.3.1 Does the software product family architecture contain variability? If the software product family architecture contains variability, what varies in the software product family architecture?

Q:4.2.3.2 What is the reason for variability in the software product family architecture?

Q:4.2.3.3 Are there examples of variability in the software product family architecture?

Q:4.2.3.4 A variability point is a predefined point where variability takes place. How many variability points does the software product family architecture contain?

Q:4.2.3.5 How are variability points documented in the software product family architecture?

Q:4.2.3.6 Variability within a variability point can be optional, which means something is added or removed, or alternative, which means choosing from a set of possibilities. Does the company have optional, alternative or both types of variability in the software product family architecture?

Q:4.2.3.7 How many percent is optional variability and how many percent is alternative variability in the software product family architecture?

Q:4.2.3.8 How many different alternatives exist for a variability point in the software product family architecture typically, minimally and maximally?

Q:4.2.3.9 How are optional and alternative variability documented in the software product family architecture?

Q:4.2.3.10 Relations between variability points may be mutually exclusive or inclusive. Mutually exclusive variability means that one choice in a variability point causes that some choice cannot be made in another variability point. Inclusive variability means that if one alternative has selected then also some other alternative needs to be selected. Does the software product family architecture contain such inclusive and exclusive relations? If there are such relations, how are they documented?

Q:4.2.3.11 How many inclusive and exclusive relations does the software product family architecture contain?

Q:4.2.3.12 How many variability points in the software product family architecture have inclusive or exclusive relations, measured, e.g., in percents?

Q:4.2.3.13 How many other variability points does an inclusive or exclusive relation affect typically, minimally and maximally in the software product family architecture?

Q:4.2.3.14 Software product family architecture can evolve differently: It can be further developed in the domain to realize more functionality, it can be developed to cover better the current domain, or it can be expanded to new domains, for example. How does the software product family architecture evolve?

Q:4.2.3.15 What are the reasons for evolution of the software product family architecture?

Q:4.2.3.16 Do shared assets of the software product family vary? If they do, what kinds of shared assets vary?

Q:4.2.3.17 Are there invariable shared asset? If invariable assets exist, how many assets are invariable measured, e.g., in percents?

Q:4.2.3.18 What varies in the shared assets?

Q:4.2.3.19 What are the reasons for variability in the shared assets?

Q:4.2.3.20 Does every shared asset vary or do only some shared assets vary? If only some of the shared assets vary, how many of them vary, measured, e.g., in percents?

Q:4.2.3.21 If there are shared assets that do not vary, what kinds of shared assets do not vary?

Q:4.2.3.22 How many variability points are in total in the all shared assets in minimum, typically, and maximally?

Q:4.2.3.23 How many variability points are in total in one shared asset typically, minimally and maximally?

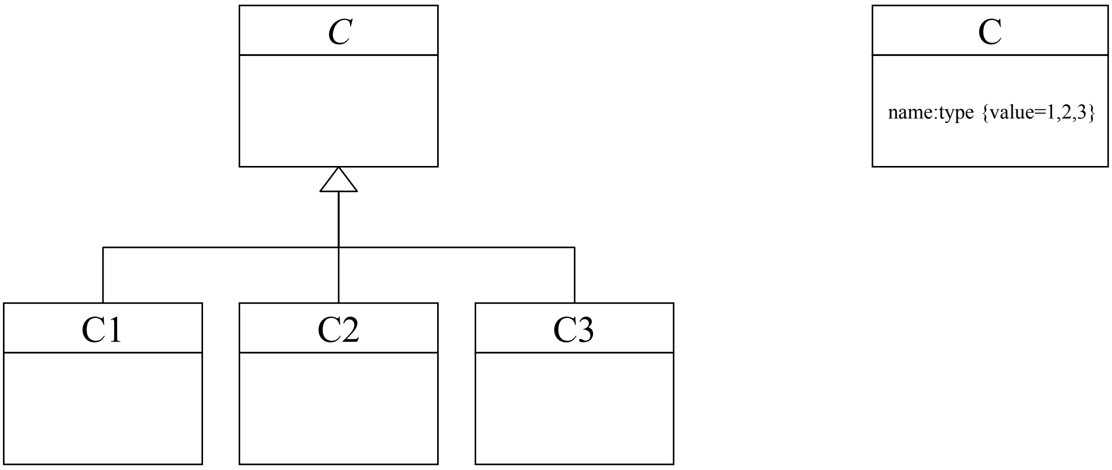
Q:4.2.3.24 How many different alternatives exist for a variability point in a shared asset typically, minimally and maximally?

Q:4.2.3.25 Are there examples of variability in the shared assets?

Q:4.2.3.26 Does the company have optional, alternative, or both types of variability in the shared assets?

Q:4.2.3.27 How many percent is optional and how many percent is alternative variability in the shared assets?

Q:4.2.3.28 Variability in a shared asset may be implemented at least in two different manners. A shared asset has several implementations or only one implementation exists in which variability is implemented, e.g., by parameters. How does the company resolve variability?



Q:4.2.3.29 How often do several implementations exist and how often does only one implementation with parameters exist?

Q:4.2.3.30 Do variability points have default variants? If such default values exist, how many variability points have a default value?

Q:4.2.3.31 How is variability documented?

Q:4.2.3.32 Do the shared assets contain inclusive and exclusive relations? If there are such relations, how are they documented?

Q:4.2.3.33 How many variability points in a shared asset has inclusive or exclusive relations typically, minimally, and maximally, measured, e.g., in percents?

Q:4.2.3.34 How many other variability points does an inclusive or exclusive relation affect typically, minimally and maximally in a shared asset?

Q:4.2.3.35 If, e.g., behavior, an interface, or quality attributes of a shared asset change, is it still the same asset? When is there variability and when are there two different shared assets?

Q:4.2.3.36 Can an interface of reusable software entity vary? If an interface can vary, how does it vary and how variability is documented?

Q:4.2.3.37 Do shared asset evolve? If they do, what evolves in the shared asset?

Q:4.2.3.38 What are the reasons for the evolution?

Q:4.2.3.39 Is it possible that an interface of a reusable software entity evolves? If evolution is possible, how does an interface evolve?

Q:4.2.3.40 How often do shared assets evolve?

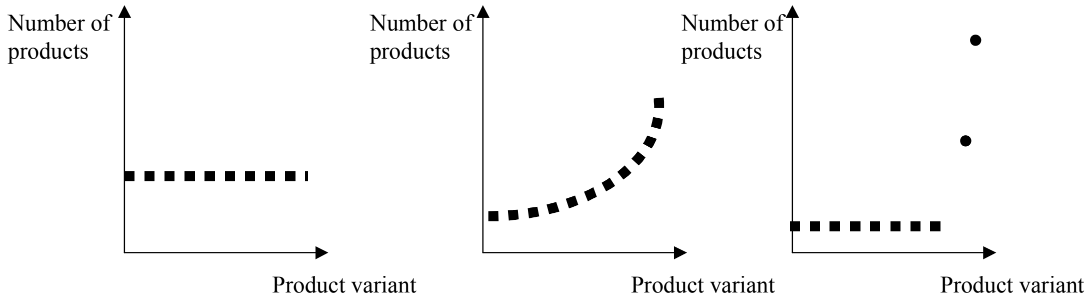
4.2.4 Resolving variability in the software product family architecture and shared assets

Q:4.2.4.1 How many variability points are resolved in total for the architecture of a product individual typically, minimally and maximally?

Q:4.2.4.2 Variability can be resolved, e.g., by selecting one from the alternatives defined for the variability point or designing a new variant for the variability point. Is it possible to design such a new alternative for a variability point in the software product family architecture during derivation?

Q:4.2.4.3 How often is a new variant designed and how often is an existing alternative or option chosen in the software product family architecture?

Q:4.2.4.4 How is the usage of alternatives in a variability point distributed in the software product family architecture? Is, e.g., some alternative used most of time or are all alternatives equally often used?



Q:4.2.4.5 Is in the software product family architecture something invariable such that it is in every product individual, and does not vary? If there is such invariables, what are they?

Q:4.2.4.6 Different kind of techniques, such as selecting a component, can be used to derive a variant of the software product family architecture. What techniques are used in a variability point for resolving a variant in the software product family architecture?

Q:4.2.4.7 Why is the company using those specific techniques for achieving and resolving variability? What are strengths and weaknesses of the techniques?

Q:4.2.4.8 What other techniques have been considered and why are they not in use?

Q:4.2.4.9 How does a derivator know about variability in the software product family architecture?

Q:4.2.4.10 Variability points can be resolved, e.g., during designing time, coding time, compile time, linking time, or execution time. What are the possible points of time for variability resolving in the software product family architecture?

Q:4.2.4.11 When is resolving in the software product family architecture usually done?

Q:4.2.4.12 How is resolved variability in the software product family architecture documented?

Q:4.2.4.13 Does the company have shared assets that are used only in some product individuals? If there are such assets, what are they?

Q:4.2.4.14 How does variability resolving in the shared assets differ from variability resolving in the software product family architecture

Q:4.2.4.15 Is it possible to design a new variant for a variability point in the shared assets during derivation?

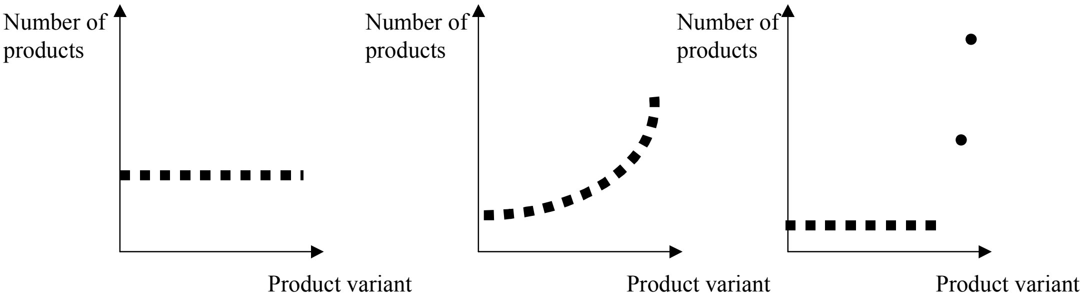
Q:4.2.4.16 How often is a new variant designed and how often is an existing alternative or option chosen in a shared asset typically, minimally, and maximally?

Q:4.2.4.17 How many different variants are, both in practice and in theory, of a shared asset typically, minimally and maximally?

Q:4.2.4.18 Do all variable shared assets vary also in practice or are they used similarly in all product individuals?

Q:4.2.4.19 How many percents of variable shared asset do not vary in practice and why do they not vary?

Q:4.2.4.20 How is the usage of alternatives in a variability point distributed in the shared assets? Is, e.g., some alternative used most of time or are all alternatives equally often used?



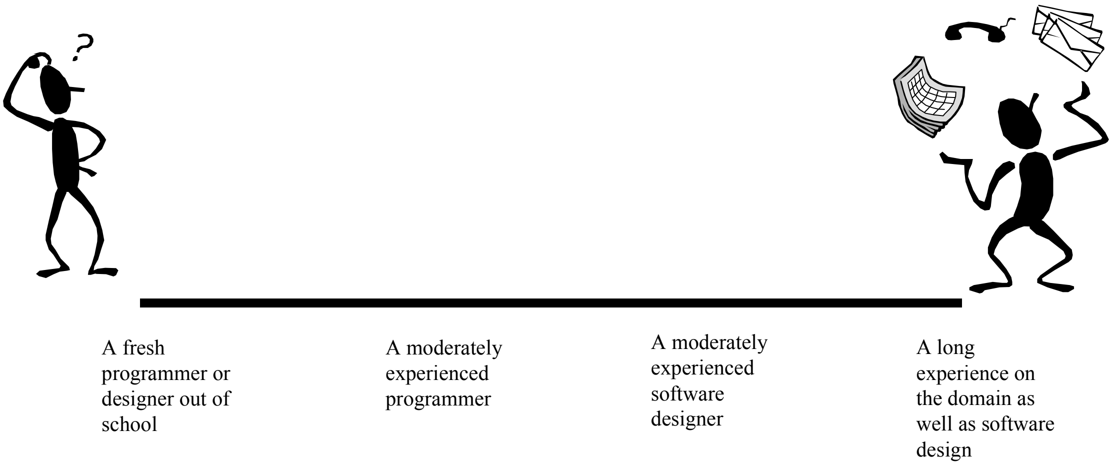
Q:4.2.4.21 Several techniques exist to achieve variability in the shared assets, such as selecting implementation, setting a parameter, conditional compiling, and inheritance. What techniques does the company use to achieve variability?

Q:4.2.4.22 Why is the company using those specific techniques for achieving variability in the shared assets, i.e. what are their strengths and weaknesses and what other techniques have been considered?

Q:4.2.4.23 How resolved variability in the shared assets is documented?

Q:4.2.4.24 The specification of a variant can be done in designing time, coding time, compile time, linking time, or execution time. When the variant can be specified and when is it usually done?

Q:4.2.4.25 How complex is it to resolve variability within a shared asset? Are some special skills needed? Can a fresh programmer or designer out of school do it or does it need a moderately experienced programmer, experienced programmer, or even a long experience on the domain as well as programming?


A fresh programmer or designer out of school
A moderately experienced programmer
A moderately experienced software designer
A long experience on the domain as well as software design

Q:4.2.4.26 What are the problems and needs in resolving variability in the software product family?

References

1. Weiss, D. and Lai, C. T. R.: Software product-line engineering: a family based software development process. Addison Wesley (1999)

2. Bosch, J.: Design and use of software architectures - adopting and evolving a product-line approach. Addison-Wesley (2000)

3. Clements, P. and Northrop, L. M.: Software Product Lines: Practices and Patterns. Addison-Wesley (2001)

4. Parnas, D. L.: On the Design and Development of Program Families. IEEE Transactions on software engineering 17(4) (1976) 40-52

5. Knauber, P., Bermejo, J., Böckle, G., Julio Cesar Sampaio do Prado Leite, Linden, F. v. d., Northrop, L. M., Stark, M., Weiss, D.: Quantifying Product Line Benefits. Lecture Notes in Computer Science 2290(2002) 155-163

6. Cohen, S.: Product Line State of the Practice Report. CMU/SEI-2002-TN-017 (2002)

7. Tracz, W.: Software Reuse Myths. ACM SIGSOFT Software Engineering Notes 13(1) (1988) 17-21

8. Frakes, W. B., Fox, C. J.: Sixteen Questions About Software Reuse. Communications of the ACM 38(7) (1995) 75-87

9. Morisio, M., Ezran, M., Tully, C.: Success and Failure Factors in Software Reuse. IEEE Transactions on software engineering 28(4) (2002) 340-357

10. Linden, F. v. d.: Software Product Families in Europe: The Esaps and Café projects. IEEE Software 19(4) (2002) 41-49

11. Bosch, J.: Product-Line Architectures in Industry: A Case Study. In: International conference on software engineering, ICSE'99ACM Los Angeles (1999) 544-554

12. .Brownsword, L., Clements, P.: A case study in successful product line development. Technical report CMU/SEI-96-TR-016 (1996)

13. Ommering, R. v.: Building Product Populations with Software Components. In: Proceeding of the International Conference on Software Engineering (ICSE'02) (2002)

14. Jaaksi, A.: Developing Mobile Browsers in a Product Line. IEEE Software 19(4) (2002) 73-80

15. Shaw, M.: What Makes Good Research in Software Engineering? International Journal on Software Tools for Technology Transfer 4(1) (2002) 1-7

16. Glass, R. L.: The Software Research Crisis. IEEE Software 11(6) (1994) 42-47

17. Tichy, W. F.: Should Computer Scientists Experiment More? Computer 31(5) (1998) 3-40

18. Kitchenham, B. A., Pfleeger, S. L., Pickard, L. M., Jones, P. W., Hoaglin, D. C., Emam, K. E., Rosenberg, J.: Preliminary Guideline for Empirical Research in Software Engineering. IEEE Transactions on software engineering 28(8) (2002) 721-734

19. Yin, R. K.: Case study Research. Sage (1994)

20. Eisenhardt, K. M.: Building Theories from Case Study Research. Academy of Management Review 14(4) (1989) 532-550

21. Raatikainen, M., Männistö, T., Soininen, T.: CASFIS-Approach for studying software product families in industry. In: 2nd Groningen Workshop on Software Variability Management (2004)

22. Linden, F. v. d., Bosch, J., Kamsties, E., Känsälä, K., Obbink, H. J.: Software Product family evaluation. Lecture Notes in Computer Science 3154 (Proc. of SPLC 2004)(2004)

23. Raatikainen, M. *A Research Instrument for an Empirical Study of Software Product Families*, Master's Thesis. Helsinki University of Technology. 2003.

24. Raatikainen, M., Soininen, T., Männistö, T., Anttila, M.: Characterizing Configurable Software Product Families and their Derivation. Software Process: Improvement and Practice To appear(2005)

25. Raatikainen, M., Soininen, T., Männistö, T., Anttila, M.: A Case Study of Two Configurable Software Product Families. Lecture Notes in Computer Science 3014(2004)

26. Beck, K.: Embracing Change with Extreme Programming. Computer 32(10) (1999) 70-77

27. Bass, L., Clements, P. and Klein, D. V.: Software architecture in practice. Addison-Wesley (1998)