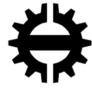

TAMPERE UNIVERSITY OF TECHNOLOGY

KRISTIAN HUHTANEN

APPLYING PRODUCT LINE APPROACH FOR A CONTROL SYSTEM FAMILY

Master of Science Thesis

Examiner: Professor Kai Koskimies
Supervisor: M.Sc.(Tech.) Antti Jaatinen
Examiner and topic approved in the Computing and Electrical Engineering Department Council meeting
on 7th December 2011

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalikäsittely ja tietoliikennetekniikan koulutusohjelma
HUHTANEN, KRISTIAN:
Tuotelinja-ajattelun soveltaminen ohjausjärjestelmätuoteperheeseen
Diplomityö, 87 sivua
Helmikuu 2012
Pääaine: Hajautetut ohjelmistot
Tarkastaja: professori Kai Koskimies
Avainsanat: ohjelmistotuotelinja, tuoterunkoarkkitehtuuri, variaatiot, tuoteperhe, ohjausjärjestelmät

Diplomityö on tehty Metso Automation Oy:lle RESPO(Reliable and Safe Processes) -projektin osana. RESPO on yksi kymmenestä EFFIMA(Energy and lifecycle efficient machines)-tutkimushankkeen projekteista. EFFIMA on FIMECC(Finnish Metal and Engineering Competence Cluster):n älykkäiden ratkaisujen tutkimusstrategian osa. RESPO-projektin yhtenä tavoitteena on kehittää sovellusarkkitehtuurin muodostamisen malleja ja suunnitteluperiaatteita. Työn tavoitteena on tutkia ohjelmistotuotelinja-ajattelun soveltamista kivenmurskauslaitteiden ohjausjärjestelmiin.

Ohjausjärjestelmätuoteperheessä on tunnistettu ohjelmistokehitykselle tyypillisiä ongelmakohtia sekä tuoteperheen heterogeenisyydessä että tuotteiden elinkaarten hallinnassa. Tuoteperheen lisääntynyt heterogeenisyys ja erilaiset variaatiot rajoittavat uudelleenkäyttöä sovelluksissa. Lisäksi ne kuluttavat ylimääräisiä resursseja tuotteiden koko elinkaaren ajan.

Työssä etsitään tuotelinja-ajattelusta ratkaisuja tuoteperheen heterogeenisyyden ja tuotteiden elinkaarten hallinnan ongelmiin. Työ keskittyy tuotelinjan kehityksen alkuvaiheeseen. Työn tavoitteena on kattaa muun muassa tuotelinjan rajaus sekä organisaatio-, prosessi- että liiketoimintanäkökulmia. Työssä mallinnetaan variaatioita nykyisessä tuoteperheessä tutkimalla eri ohjausjärjestelmien vaatimuksia ja ominaisuuksia. Näiden perusteella järjestelmien kehityssuuntaa ja tulevia tarpeita estimoidaan. Lisäksi työssä mallinnetaan variaatioita nykyisessä tuoteperheessä, jotta niitä voidaan hallita paremmin tulevaisuudessa. Tuoteperheen variaatioiden ja vaatimusten perusteella luodaan alustava modernisoitu tuoterunkoarkkitehtuuri uuden sukupolven tuoteperheelle.

Uuden arkkitehtuurin tavoitteina ovat: pienemmät kustannukset, lyhyempi kehitysaika, vähentyneet virheet, strateginen uudelleenkäyttö ja helpottunut tuotehallinta. Näiden tavoitteiden saavuttamiseksi työssä määritetään tuoterunkoarkkitehtuurin lisäksi myös tuotelinja-ajattelusta mukailtu ohjelmistokehitysprosessi ja organisaatiojako. Työ sisältää myös tuoterunkoarkkitehtuurin ja tuotelinjan arviointiosuudet, joiden tarkoituksena on arvioida vahvuuksia ja heikkouksia valitusta lähestymistavasta.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Signal Processing and Communications Engineering
HUHTANEN, KRISTIAN: Applying Product Line Approach for a Control System Family
Master of Science Thesis, 87 pages
February 2012
Major: Distributed software
Examiner: Professor Kai Koskimies
Keywords: product lines, product line architecture, variation management, product family, control systems

This thesis was done for Metso Corporation as a part of RESPO project. RESPO is one of the ten projects in EFFIMA (Energy and Life Cycle Efficient Machines) research program. EFFIMA belongs to FIMECC's (Finnish Metals and Engineering Competence Cluster) Intelligent Solutions (IS) strategic research theme. The purpose of task 2 in RESPO is to develop models and design principles into the development of software architecture. The goal of this thesis is to study the possibilities of applying software product line approach to rock crushing control system family.

Several software-related problems have been recognized with the control system family. These include the long lifecycles and heterogeneity in the family. Another challenge is to manage variations in the family. The uncontrolled variations and heterogeneity prevent the effective reuse and increase the amount of extra work throughout the product lifecycle.

The product line approach is applied to find solutions to the problems presented before. The approach in this thesis concentrates in the early development phase of the product line that includes addressing business, organizational, process and technological aspects. The variations in the current product family are modelled by scoping the requirements and the properties of control systems. The scoping is used to provide an understanding of the development trend in the business segment and thus to estimate future requirements. It is also used to provide better means for variation management in the product family. The scoping process and the variation modelling are used to create preliminary modernized product line architecture for next generation control systems.

Less development and maintenance costs, shorter time-to-market, less errors, increased expandability, strategic reuse and easier product management are key incentives for the new architecture approach. To achieve these, the organization and its processes must be adapted and committed to the product line concept. In order to gain full benefits from the approach, the strengths and the weaknesses of both architecture and the product line itself need to be evaluated.

PREFACE

This thesis was done as a part of RESPO project in EFFIMA research program for Metso Corporation. I give my regards to my inspector professor Kai Koskimies and to my instructor M.Sc.(Tech.) Antti Jaatinen. Mr Jaatinen is responsible of Metso Corporation participation in RESPO project. I would like to thank Mikko Mäkinen for making this thesis possible. I would also like to extend my gratitude to other personnel in Metso Corporation for giving guidance and alternative viewpoints to the topic. In addition I would like to give special thanks to my parents and brothers for given support and encouragement.

On 9th of January 2012, In Tampere, Finland

Kristian Huhtanen

TABLE OF CONTENTS

1 Introduction 10
2 Background 12
- 2.1 Software architecture 12
  - 2.1.1 Overview 12
  - 2.1.2 Architecture views and viewpoints 14
  - 2.1.3 Design patterns 14
  - 2.1.4 Features 15
  - 2.1.5 Functionality and architecture 15
  - 2.1.6 Architecture and quality attributes 16
  - 2.1.7 Importance of software architecture 16
- 2.2 Variation in software systems 17
  - 2.2.1 Variation management 17
  - 2.2.2 Modelling variability 18
  - 2.2.3 Variation visualization 20
  - 2.2.4 Implementing variation 21
- 2.3 Software product lines 22
  - 2.3.1 Overview 23
  - 2.3.2 Variation point 24
  - 2.3.3 Product line management 26
  - 2.3.4 Domain engineering 27
  - 2.3.5 Application engineering 27
  - 2.3.6 From requirements to a product 27
  - 2.3.7 Modelling commonality and variability in product lines 28
- 2.4 Initiating a product line 29
  - 2.4.1 Approaching product line architecture 29
  - 2.4.2 Business case analysis 30
  - 2.4.3 Scoping 31
  - 2.4.4 Product and feature planning 31
  - 2.4.5 Product line architecture design process 31
  - 2.4.6 Validation 32

3 Starting point 34
- 3.1 Rock crushing automation 34
- 3.2 Business 36
- 3.3 Organization 37
- 3.4 Process 37
- 3.5 Technology 39
  - 3.5.1 Metso DNA 39
  - 3.5.2 Developing tools 42
- 3.6 IC product family 43
  - 3.6.1 Overview 44

3.6.2 History 45
3.6.3 Hardware 45
3.6.4 Communication 46
3.6.5 Software 48
3.6.6 Example IC control systems 49
4 From individual products to a product line 54
4.1 Scoping 54
4.1.1 Common requirements 54
4.1.2 Product feature matrix and graph 57
4.2 Business case analysis 61
4.3 Product and feature planning 62
4.4 Design of the product line architecture 63
4.5 Organization 66
4.6 Process 68
4.7 Best practices 70
5 Evaluation 71
5.1 Architecture assessment 71
5.1.1 Architectural design decisions 71
5.1.2 Modifiability 72
5.1.3 Scenario analysis 73
5.1.4 Results 77
5.2 Product line assessment 79
5.2.1 Strengths 79
5.2.2 Weaknesses 79
5.2.3 Opportunities 80
5.2.4 Threats 80
6 Conclusion 81
References 83

LIST OF FIGURES

Figure 1. Conceptual class diagram of a system. Adapted from [5]. 13
Figure 2. Variation in time. Adapted from [16]. 20
Figure 3. Methods to visualize variation in a system. Adapted from [17]. 21
Figure 4. Basic concepts of Software product line engineering. Adapted from [3]. 24
Figure 5. Key activities in software product line engineering. Adapted from [3]. 24
Figure 6. Variability planes. Adapted from [23]. 25
Figure 7. Early and late variability funnel with variability levels. Adapted from [15].. 26
Figure 8. Software development with product lines. Adapted from [24]. 28
Figure 9. Alternative approaches into product line architecture development. Adapted from [9, p. 167]. 30
Figure 10. Small open pit mine. [30]. 35
Figure 11. Controller tasks in rock crushing automation. Adapted from [31]. 36
Figure 12. General view of MIPA. Adapted from [30]. 37
Figure 13. Software development phases and deliverables. Adapted from [30]. 38
Figure 14. Three activities in Metso Dynamic Network for Applications. Adapted from [33]. 39
Figure 15. Metso DNA architecture. Adapted from [33]. 40
Figure 16. Metso DNA concepts. Adapted from [30]. 42
Figure 17. The automation level of IC control systems. 44
Figure 18. IC product family. 44
Figure 19. Hardware abstraction of a machine control system. Adapted from [31]. 46
Figure 20. Communication levels with IC control systems. 47
Figure 21. An abstract software structure of IC control system. 48
Figure 22. The hardware of stationary cone crusher. Adapted from [30]. 49
Figure 23. The automation hardware of IC7000. Adapted from [30]. 50
Figure 24. The hardware of a portable jaw crusher. Adapted from [30]. 51
Figure 25. The automation hardware of IC10. Adapted from [30]. 51
Figure 26. The hardware of a stationary jaw crusher. Adapted from [30]. 52
Figure 27. The automation hardware of IC1000. Adapted from [30]. 53
Figure 28. Partial feature graph of current IC family. 59
Figure 29. IC product line overview. 63
Figure 30. IC product line architecture as a class diagram. 65
Figure 31. A structure design for an abstract unit controlled by IC. 66
Figure 32. The engineering unit hierarchy in IC product line. 67
Figure 33. Product line development applied to Metso needs. 69

TERMS AND DEFINITIONS

Term	**Definition**
ALP	Alarm processing application server in Metso DNA.
API	Application Programming Interface.
Automation	Use of control systems and information technologies to reduce the need of human participation in production.
BU	Maintenance Server. Backup Server is used to save current configuration and packages of a Metso DNA application.
Control system	Device or a set of devices to manage, command, direct or regulate the behaviour of other devices and systems.
DCS	Distributed Control System. See control system.
DIA	Maintenance Server. Diagnostics Server is used for debugging Metso DNA applications.
DNA Operate	Application Server. Operator Interface Server is used for all user interaction.
EAS/EAC	Engineering Server. EA Repository Server / Client are used to configure the control system.
GUI	Graphical User Interface to enable human machine interaction.
HAL	Hardware Abstraction Layer.
HCI	Human Computer Interaction or Human Computer Interface.
HMI	Human Machine Interaction or Human Machine Interface.
HSE	Health, Safety and Environment.
IC	Intelligent Controller, Control system family at machine automation level.
IP	Ingress Protection. Used to classify rugged hardware.
Metso DNA	Metso Dynamic Network for Applications. DCS produced by Metso Inc.
PCS	Application Server. Process Control Server includes most of the business logic.

SAAM	Software Architecture Analysis Method.
SCADA	Supervisory Control And Data Acquisition.
SEI	Software Engineering Institute, Carnegie Mellon University.
SWOT	Also known as SLOT-analysis. SWOT is a strategic planning method used to evaluate strengths, weaknesses, opportunities and threats of an approach.
VP	Variation point. Used to identify locations in product line at which variation may occur.
Variation space	Variability existence of an artefact in different shapes at the same time.

1 INTRODUCTION

Organizations developing automation software systems face a great deal of challenges today in both choosing a market segment and system domain. The systems are complex, because of the integration of mechanical, electrical and software components. Typically these systems are developed in small series ranging from a few to a few hundred units. The lifecycle of machine hardware can be up to 30 or 40 years when the lifecycles of automation hardware and software are up to 10 and 20 years at most. The fact of automation hardware has shorter lifecycle than the software, generates challenges especially in maintenance. Another characteristic is the high commonality between the systems. This is due to the similar software requirements from different customers. [1.]

However, even though the requirements are similar the end products vary. There are at least four reasons for the heterogeneity of end products. Firstly, all systems are created by developers each having their own preferences and unique background. Secondly, alternative solutions are enabled as technology advances. Thirdly, development tools and process can never be introduced throughout a large organization instantly. Finally, even though the requirements are similar, there is always need to product tailoring. This is because no single solution serves the needs of every customer. [2.]

Another key issue in software development is the reuse of software. Especially when developing similar software systems in series, the amount of overlapping work can be reduced by reusing different assets. The reuse can be applied for example to specifications, designs, implementation and testing. With a proper approach reusability of documentation, architecture, components and tests can significantly reduce the costs, time-to-market and quality of the end product.

History of reuse in software development begins in 1960s by reuse of subroutines. Only a decade later the reuse of modules was first introduced. In 1980s objects and in 1990s components were used to create applications. [3.] Recently an approach commonly known as software product lines was introduced. The product lines combine business strategy to the technical one striving for strategic reuse. In practice this means a planned approach using reusable assets in software development.

The product line approach copes with the problems introduced before by having common core assets for all products within a family. The core assets are developed separately from product development. In practice the skeleton of all applications is created from same reusable assets. Thus the benefits of the reuse are gained and only a small amount of customization is required to meet the demands of a customer.

The challenges of software development have also been recognized in Metso Corporation. Metso is a global supplier of process industry machinery with automation and after sales support. Rock crushing automation is one of many business sections in Metso. The rock crushing automation aims to provide software solutions to the needs of Mining and Construction (MAC), which is a business segment within Metso organization. MAC provides whole and partial crushing plant solutions to both underground and surface mines.

The goal of this thesis is to study the possibility of using a product line approach in a control system family for the harsh environment of portable and stationary rock crushing machines. This thesis addresses business, organizational, process and technology aspects at the early phase of a product line designing. One of the key issues in product lines is to manage and model the variance in systems in order to create a basis for common product line architecture for future applications. This is done by scoping the control system family and its requirements. Other aspects discussed in the thesis, are the organization and the processes needed to provide means for better communication between different stakeholders involved in software development.

The content of the thesis is divided into six chapters including introduction. Chapter 2 provides the background from software architecture, variations in software systems and product line approach. Chapter 3 illustrates the domain of rock crushing automation. This includes describing the different control systems, technology, processes and the organization needed in development. Chapter 4 provides adaptation of product line approach to the situation described in previous chapter. The adaptation includes among others feature matrix of the family, a prototype of product line architecture and adjusted software development process to support the product line approach. The chapter also describes the situation from business and organizational viewpoints. Chapter 5 reviews and evaluates the thesis and the rationality of provided results. Chapter 6 provides a concluding summary for the thesis.

2 BACKGROUND

Architecture is the core of a software system. It is used to provide an abstraction of the structure and functionality of the system. Same architecture can be used to create several systems. However, all systems created from the same architecture are not necessarily alike. These variations need to be modelled and managed especially in system families. Unmanaged variations reduce the effectiveness of software development and maintenance. This chapter provides the basic principles of software architecture, variation in software systems and concludes in the description of software product line approach, which is used later on this thesis to address the common problems in software development.

2.1 Software architecture

Software architecture is a documented description of the most relevant design decisions made for a system. It enables the management of the system through its lifecycle. The architecture has a significant effect on achieving the quality requirements set for the system. Therefore many software developing approaches lean strongly on architecture development.

2.1.1 Overview

Software architecture is used to provide a harsh abstraction of a system that satisfies requirements set for the system. The software architecture has almost as many definitions as people addressing it. Nevertheless almost all definitions include components or elements, relations between them and a structure. One commonly used definition from Len Bass states following [4]:

”The software architecture of a program or a computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements and the relationships among them.”

Four basic aspects can be derived from the definition:

1. Architecture defines software elements.
2. Systems can comprise more than one structure.
3. Every system with software has architecture.
4. Architecture includes the behaviour of the elements.

Elements or components can be seen as basic building blocks of the system. The architecture is needed to define relations between these components. Depending on a viewpoint or an abstraction level, a component can be seen quite differently. This leads into the second aspect, which states that a system can comprise more than one structure. Especially in large software projects, the system can be divided into several structures, or subsystems. The definition also states that every software system has architecture. This means that a software system created without any particular planning has architecture just as a software system developed with careful designing. The definition leaves out evaluation, which is needed to determine the quality of the architecture. To complete the definition, the behaviour of different component needs to be addressed so that the components understand and can interact with each other. The behavioural description of a component includes its functionality and interaction with other system components. To enable interactions between components, the external visibility of each component need to be described. A conceptual model of system architecture and its relations to other aspects of software development is illustrated in Figure 1.

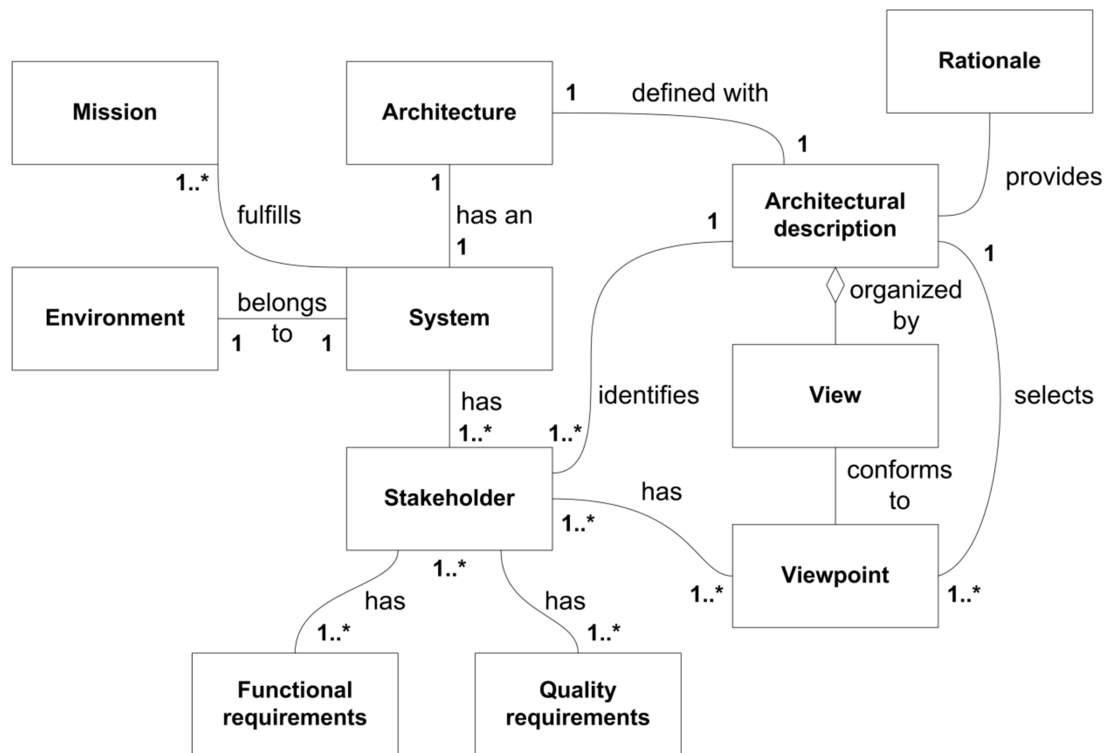

Figure 1. *Conceptual class diagram of a system. Adapted from [5].*

Every system has an architecture designed for a specific task into described environment. These are the basis for any architectural description. The architectural description includes also design decisions and their rationale. The design decisions are connected to requirements from stakeholders having all their own viewpoints.

2.1.2 Architecture views and viewpoints

A view is another important aspect in architectures. A view is used to represent a set of architectural components and relations between them. The architecture can be examined from several different viewpoints and each with different abstraction levels. The viewpoint usually depends on the role of the stakeholder such as a basic user, an architect, a developer, a manager or a support engineer. The viewpoints and their usefulness for different stakeholders are presented in [4, p. 206]. Commonly used 4+1 approach presents five different views to analyze software systems [6].

- *Use case view* concentrates describing the system from an external viewpoint. Relevant for the view is how the system interacts with its environment. This viewpoint is most useful for a basic user, manager or support engineer.
- *Logical view* digs in a bit deeper into the architecture and includes static software components, such as classes and interfaces, and the interaction between them. The view puts stress on the internal functionality of the system. Therefore it is most useful to software, design, test and support engineers.
- *Process view* takes into consideration the management and the interaction of parallel processes and threads. The view puts weight on the performance, scalability aspects of the system.
- *Implementation view* divides the system into physical parts such as files, which are merged to form the system. The view is useful especially for administrators and support engineers.
- *Deployment view* describes the distribution of the system. In this view different hardware and software components and the connections between subsystems are under evaluation.

2.1.3 Design patterns

A design pattern is commonly known as a proven and used design solution to a particular problem. The use of design patterns has six significant benefits:

- Providing a solution to a common architecture or design problem
- Bringing forth and sharing the design knowledge within the organization
- Decreasing the effort needed for documenting the system
- Providing means to a higher level abstraction
- Working as building blocks of the system
- Uniting the terminology used by different developers and designers

An extensive description of different types of design patterns can be found in [7; 8].

2.1.4 Features

Software systems are usually very complex. A complex system includes lots of requirements and functionality. Specifying a system only with requirements and functionality can be impractical. Thus the requirements are connected to the functionalities that are grouped into features. The definition of a feature according to Jan Bosch is [9]:

“a logical unit of functionality that is specified by a set of functional and quality requirements”

Often different products in a same product family share same functional and quality requirements to a certain point, but the rest are product-specific. This leads to situation, where features need to be categorized in order to separate common features from product-specific ones. Table 1 illustrates one possible categorizing for features.

Table 1. *Feature types. Adapted from [10].*

Feature type	**Meaning**
Mandatory	The feature must be included.
Optional	The feature may be included as an independent complement.
Alternative	The feature replaces another feature.
Mutually Inclusive	Including the specific feature requires other features to be included as well.
Mutually Exclusive	Including the specific feature hinders the ability of including related other features.

Features are the basis of several architecture modelling and analysis approaches. One of these is widely used Feature-Oriented Domain Analysis (FODA) and an extension of it is another known as Feature-Oriented Reuse Method (FORM). In the late 1990s introduced FORM is a popular approach especially in academic circles. [11.]

2.1.5 Functionality and architecture

The functionality stands for the ability of a software system to accomplish its specified tasks. In general this requires several software components working together. Therefore the responsibilities and the interfaces of components need to be defined accurately.

Software architecture describes how the functionality is achieved in a system. This is done by determining a structure in which functionality is allocated. For example, the same functionality can be achieved through implementing a single module or several modules. Generally dividing functionality into several modules is more effective

considering software development and especially from a reuse viewpoint. Approaching the same problem through the single module structure causes difficulties especially when several developers are building a software system simultaneously. On the other hand, dividing a system into too many structural components increases the amount of management needed to support the development. Thus architecture decisions create constraints for the system and for the organization. [4, p. 72.]

2.1.6 Architecture and quality attributes

In addition to functional requirements all software architectures need to address quality aspects as well. The quality and the functional requirements commonly go hand in hand. Thus designing and modifying architecture involves balancing between the two.

The quality attributes for architecture may include usability, modifiability and performance aspects. Usability includes also non-architectural aspects such as user interface (UI) designing. Modifiability is often the most important quality attribute for the architecture, because the architecture needs to be evolvable and adaptable to future system requirements. The performance is the most critical requirement especially in systems with safety-related or real-time requirements. These systems include both architectural and non-architectural quality requirements. For example, the basic structure of a real-time system is an architectural decision, but the implementation of specific algorithm is a non-architectural decision. [4, p. 73.]

2.1.7 Importance of software architecture

As discussed, architecture plays a key role in system designing. Architecture with its description is a powerful tool to model the complexity of different software systems. Architecture has three fundamentally important functions.

Firstly, it provides means to a better communication among stakeholders. Generally each stakeholder has different point of view and concerns addressing the system. The architecture description and models provides a common language among stakeholders.

Secondly, architecture contains the earliest design decisions. In most cases the architecture addresses the very fundamentals of the system being developed. Early decisions, heedless of the quality, set limits in the remaining software development process. For example, weak structural design decisions can have a tremendous impact on the implementation process. In addition the architecture can have an effect in organizational structure. Architecture provides a top-level abstraction to a system and thus can be used to divide workload between stakeholders. The simpler the architecture is to understand the simpler it is to assign personnel to implement different system components. Hence more accurate cost and time estimates can be achieved through well documented architecture.

Thirdly, as the architecture is an abstraction of a system being designed, it can be reused later on. This gives significant benefits especially in organizations building several similar products with small variations. Through the reuse of architecture and mastering its design decisions, all the previously discussed advantages can be gained. This explains the interest towards a software product line approach. The software product line approach is a way to increase reuse of software components by using a common core to create slightly different products in a product family. Better cost-efficiency and shorter time-to-market is also achieved with product lines compared to more common development methods. [4, p. 26.]

2.2 Variation in software systems

Software systems are based on requirements, which rarely describe exactly how the system is to be done. Therefore the system can be developed for example based on different hardware and softwar. This is the main reason for the existence of design choices used to specify a system being created. The design choices are the origin for variants, which specify differences in similar systems. Variation exists throughout a product lifecycle in different forms or types. Different phases in the product lifecycle are commonly known as variation levels. The variability of a software system is a critical factor when designing reusable system components. Therefore variation management is needed to handle dependencies and variations in software artefacts.

2.2.1 Variation management

One of the key activities in software engineering is to manage commonalities and variations between products. According to Schmid this activity, variation management, is defined as:

“Variability management encompasses the activities of explicitly representing variability in software artefacts throughout the lifecycle, managing dependencies among different variability, and supporting the instantiations of the variability.” [12.]

Variation management addresses different aspects of variability in software engineering. This includes identifying, modelling, storing, changing and initiating variations throughout the lifecycle of a product. Variations, variants and variation levels, and their relations need to be managed. To harness the full potential from software development, proper variation management methods are needed. Managing variations can further be divided into key elements: consistency, scalability, traceability and visualization.

Consistency stands for the standardization of processes meaning that variability is handled in the same way on all variation levels and throughout product lifecycles. Similar products may form product families in which products can be developed and maintained with same resources and processes. The product families can expand or shrink frequently, which means that the development methods need to be adapted to new needs. This aspect is commonly known as scalability. Traceability is an important factor when changes are to be done. The traceability needs to be supported both horizontally and vertically. The horizontal traceability ensures that variations can be traced within same stage of a product lifecycle. For example, changes designing functionality for one feature may cause changes in another. The vertical traceability means that variations can be traced from one lifecycle stage to another. Variations on different levels and lifecycle stages are connected to each other by a path activated with a design decision. These relations need to be mapped correctly in order to make controlled changes possible. Need for proper variation visualization grows with the complexity of the product. In complex systems several models and visualization methods are needed in abstracting variability and its dependencies into a more simplified form. [2; 13] In addition to visualization also effective processes are needed to identify, define, trace and manage variability [14].

2.2.2 Modelling variability

Variation types

Variations in a product line can be categorized by their type and meaning [10]. These are used to for example determine whether a specific feature is added or removed from the system. Table 2 represents an example variation categorization for implementation phase. This categorizing may also be applied, with small adjustments, to other phases of software development.

Table 2. *Variation types. Adapted from [10].*

Variability type	**Meaning**
Positive	Feature is added.
Negative	Feature is removed.
Optional	Feature is included.
Alternative	Feature is replaced.
Function	Functional changes occur.
Behavioural	Behavioural changes occur.
Platform / Environment	Platform or environment changes.

Another way to categorize variations is to split variations into external and internal variability. This approach is more suitable for marketing or sales functions due to the fact that it illustrates variations in a simplified form. For example, the external variations may be visible or selectable to all stakeholders, including customers, whereas the internal variations are hidden from all except the developers.

Levels of variability

Variability exists on different abstraction levels of system design. At product level variability can be seen as variations in system architecture. At component level variability addresses aspects such as, how to evolve and add new interfaces to a component, so that the reusability of the component is increased. Additionally a conventional component may consist of feature sets, which can vary on sub-component level. Nevertheless most of the system variability takes place at code level because of the implementation of functionalities varies depending on developers and their preferences. [15.]

Variation in time and space

As said variation exist in many forms. Variation can exist in time and space. Variation in time can be easily understood as different versions of an artefact or a product if preferred. Configuration management is the practice to address problems regarding variation in time. Variation in time is can be illustrated by dividing ordinary software development into domain engineering and to application engineering. The amount of variation behaves differently when developing a platform than an application on top of it. These differences are illustrated in Figure 2.

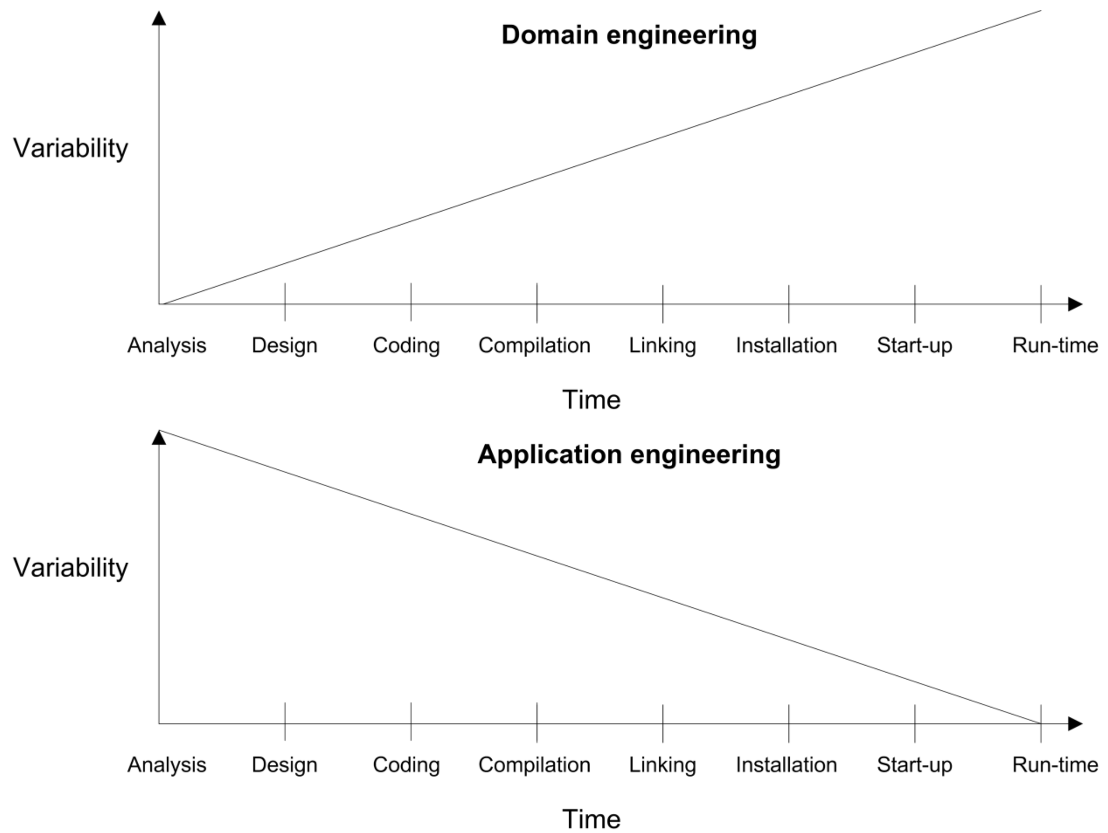

Figure 2. *Variation in time. Adapted from [16].*

Variation in space stands for artefacts existing in different shapes. In practice this means that basic system assets or components can be used slight differently in different products. For example, the behaviour of the component can change depending on variations in the quality requirements of a system.

2.2.3 Variation visualization

As earlier emphasized the visualization of the variability is one of the key aspects in variation management. Thereby proper representation methods are needed. Several different notations for modelling have been introduced, but no standard has been created. Many of the representation notations are feature-oriented and concentrate in representation of all possible valid product configurations, which is the main task of the variation visualization. Figure 3 illustrates three approaches in variability representation.

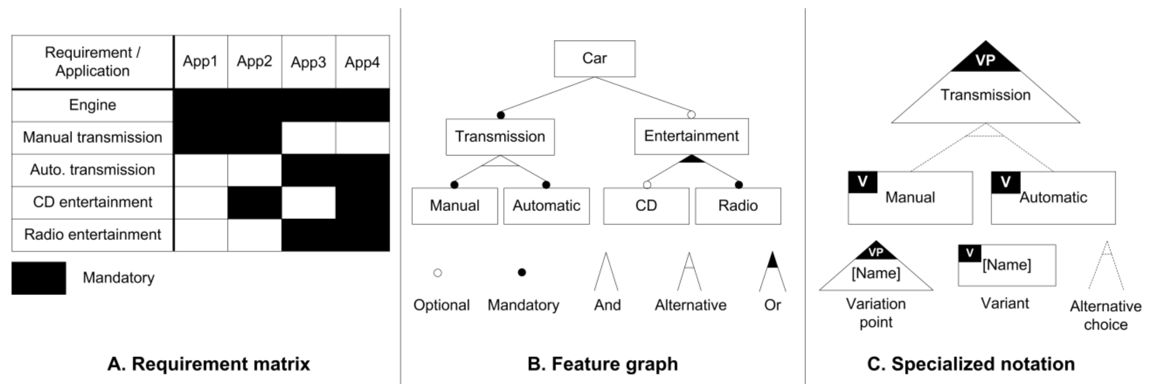

Figure 3. *Methods to visualize variation in a system. Adapted from [17].*

Different types of feature matrices and diagrams are a very common and effective way to illustrate variability in simple cases, but as the complexity of a product grows so does the amount of information needed to represent. Challenges in scalability are not the only ones. In addition especially largely heterogeneous product families require a massive amount of flexibility from visualization methods and tool support. Current recognized weak spots with the tool support are [17; 18]:

- Support for domain specific adaptations
- Traceability modelling
- Extension mechanisms
- Support for evolution
- Multi-team modelling capabilities
- Integration with sales processes

2.2.4 Implementing variation

Implementation of variation consists of two essential steps. First variation needs to be specified so that different variants, their behaviour and the method of implementation are defined. The second step in the process is commonly known as realization. At this step a realization mechanism is chosen for implementation of an artefact. There are several mechanisms available depending on the environmental and the architectural constraints of the system. Different mechanisms and their support for variability aspects are illustrated in Table 3. [10.]

Table 3. *Variation implementation methods and use cases. Adapted from [10].*

	Interface					Implementation					Initialization					Timing				Other		
	Positive	Negative	Optional	Alternative	Multioptional	Positive	Negative	Optional	Alternative	Multioptional	Positive	Negative	Optional	Alternative	Multioptional	Compile	Link	Run	Postrun	Scalability	Traceability	SoC
Aspect-oriented programming	Difficult / Ineffective	Possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective
Conditional Compilation	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Not possible	Not possible	Not possible	Possible	Possible	Difficult / Ineffective
Dynamic Class Loading	Not possible	Not possible	Not possible	Not possible	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Not possible	Not possible	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective
Dynamic Link Libraries	Not possible	Not possible	Not possible	Not possible	Not possible	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective
Inheritance	Difficult / Ineffective	Not possible	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Not possible	Possible	Difficult / Ineffective	Difficult / Ineffective
Overloading	Not possible	Not possible	Not possible	Not possible	Not possible	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Not possible	Difficult / Ineffective	Not possible	Not possible	Not possible	Possible	Possible	Possible
Parameterization	Not possible	Not possible	Not possible	Not possible	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Possible	Difficult / Ineffective	Possible
Static Libraries	Not possible	Not possible	Not possible	Not possible	Not possible	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Difficult / Ineffective	Not possible	Difficult / Ineffective	Not possible	Not possible	Possible	Possible	Difficult / Ineffective

Possible
Difficult / Ineffective
Not possible

The idea of parameterization is to create components whose behaviour or functionality can be configured by setting parameters. One specialization of parameterization mechanism is dynamic parameterization, which stands for the run-time modification of components. Parameterization is an effective way to enhance the reusability of components and increase traceability between design decisions. Templates are models created with generic code, which can be parameterized to accomplish a specified behaviour. Extensions and inheritance are mechanisms used to increase attributes and functionality of a reusable component to fulfil a set of requirements. [10; 15.]

2.3 Software product lines

Software product line is an approach to use a common architecture and asset base to develop several similar products. The software product line is based on product line architecture and several activities needed, for example, to manage variation points throughout product lifecycle. The product line engineering includes three essential activities: domain engineering, application engineering and product line management, which are described more thoroughly in this chapter.

2.3.1 Overview

Software product line engineering is a fairly new approach to increase the amount of reuse in software engineering. The basis of the software product line is architecture, which specifies commonalities as well as planned variability of different products in a product family. The advantages of the approach are studied to be increased productivity, time-to-market, customer satisfaction, better product quality and lower labour needs [1; 3; 19; 20; 21]. The approach fits well into organizations that have a need to produce several similar products with slight variations. The definition of software product line or software product family varies depending of the viewer. According to Clements and Northrop software product line is following [3]:

“A software product line is a set of software-intensive systems sharing a common, managed set of features that satisfy the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

Definition by Jan Bosch [9]:

“A software product line consists of product line architecture and a set of reusable components that are designed for incorporation into the product line architecture. In addition, the product line consists of the software products that are developed using the mentioned reusable assets.”

Though there are slight differences between the two definitions, both recognize the same concept. That is, in the product line there is a set of product specific assets and reusable core assets, which are shared by different products. Core assets refer typically to a reuse repository of a product line. This includes assets such as software components, product line architecture, requirements, documentation, schedules, budgets etc. In addition, there is a planned and managed way to use the assets to create products fulfilling set requirements. The basic concepts of product line are illustrated in Figure 4.

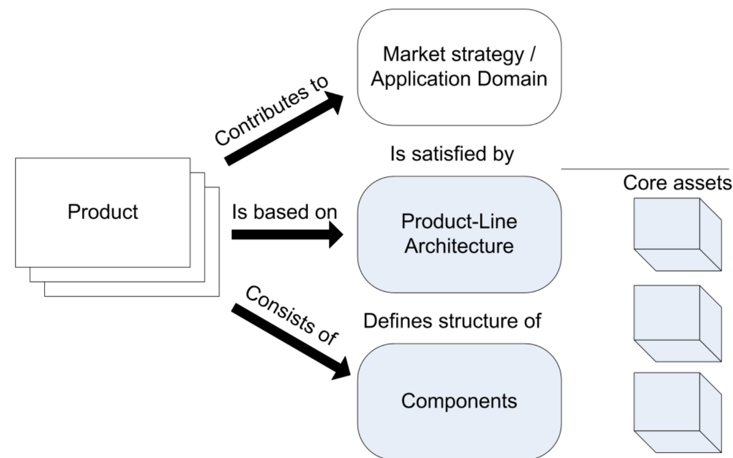

Figure 4. *Basic concepts of Software product line engineering. Adapted from [3].*

Another way to understand product line principles is through its activities. A product line consists of three major activities illustrated in Figure 5.

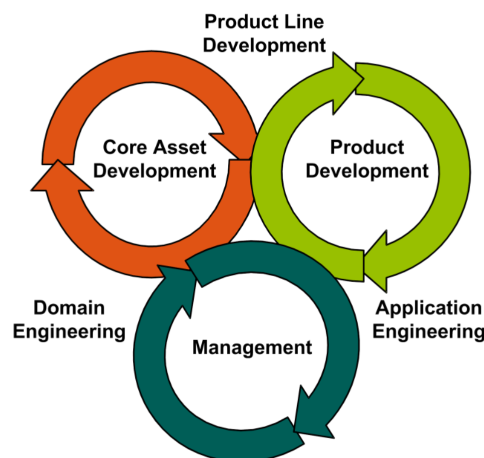

Figure 5. *Key activities in software product line engineering. Adapted from [3].*

All key activities include decisions that have an effect on the product being developed. The decisions and their effects need to be recognized and thus variation points are used.

2.3.2 Variation point

In software engineering design decisions are vital and need to be made when developing a system. In product lines these decisions are modelled through variation points. A variation point describes variability manifestation in development artefacts [22]. Each decision point (DP) correlates a variation point (VP). Each variation point is at first specified (VPs) and then later on realized (VPr). This conceptual approach enables variation points to be scattered on different abstraction planes. These planes are illustrated in Figure 6.

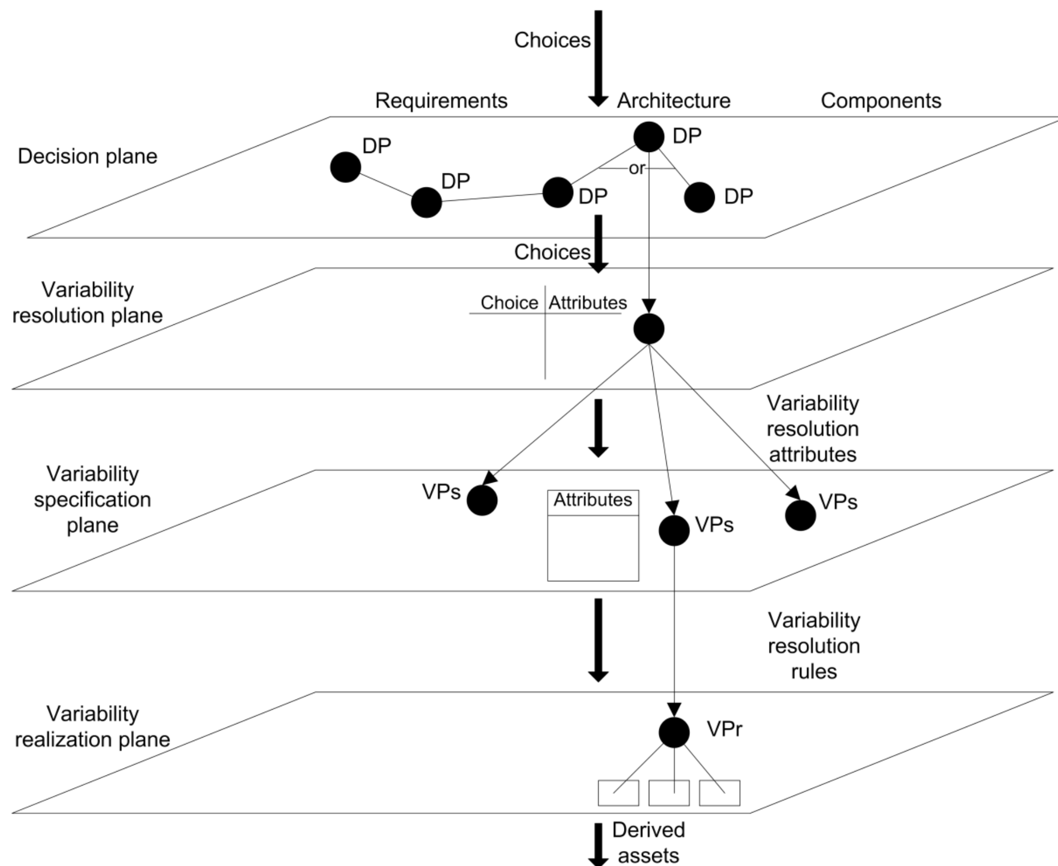

Figure 6. *Variability planes. Adapted from [23].*

A decision point consists of information about the transformations, constraints and rules of design decisions. These aspects are used to model the effects of selecting a certain variation point and implementing it to the system. For example, selecting certain features to a system defines its functionality and may create constraints in system design or implementation. Rules withhold justification and rationale for choosing a specific approach. The point in a product lifecycle at which, a particular variant for a variation point is bound to the system is commonly known as binding time. The binding time describes the moment when variation point is realized. After this moment changes are no longer possible to the specified point. This drives architects and developers to delay design decisions and thus creating more flexibility in the system development. The effects of delaying decisions are illustrated in Figure 7.

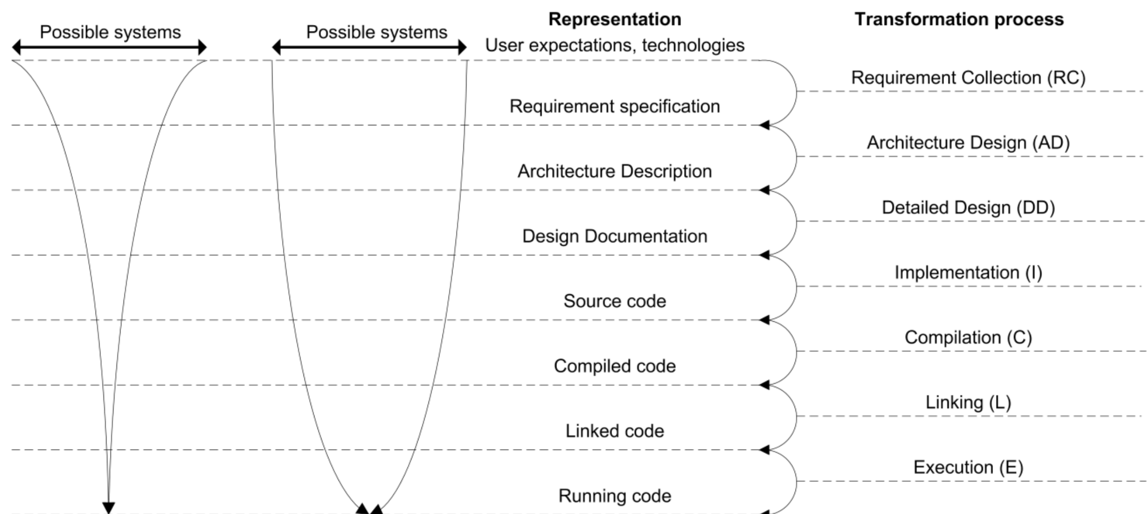

Figure 7. *Early and late variability funnel with variability levels. Adapted from [15].*

The amount of possible systems is significantly decreased if design decisions and thus variation points are realized in early development phase. Delaying the design decision and thus moving a specific variation point to another variation level enables more possibilities in the following development phases [15]. However having a great deal of flexibility increases the risk of rippling in the development. This is why sometimes the narrow funnel approach is chosen to have a more control in the development process and thus to avoid unmanaged variations in systems.

2.3.3 Product line management

As stated before, management is one of the key activities in product lines. Aspects in need of management are:

- Market and product strategy
- Economical aspects
- Scope and constraints
- Evolution and direction
- Domain and application engineering

A lucid market and a product strategy determine the outlines of a product line context. These require relevant context awareness, which can be achieved through collecting and analyzing data from the environment of the system being designed. The economical aspects include, among others, optimization of product line process so that production and maintenance costs are minimized, and time-to-market is as short as possible. Also product line scope and constraints need to be defined in order to control the size of a product family. An ambiguous scope and constraints result not in product family, but in random product population, which cannot be supported by a product line approach. A well-defined product portfolio and a product road map are key elements limiting the

scope and guiding the evolution of a product line. Domain and application engineering need to work seamlessly. This requires common guidelines and instructions, for example, to modelling, instantiation and derivation. In short the product line management can be seen as a support activity for the domain and the application engineering. [3, p. 45.]

2.3.4 Domain engineering

Domain engineering, also known as core asset development, is responsible to build and maintain the very basis of the product line. The domain engineering includes analyzing product and production constraints, and scoping for pre-existing assets. The analysis is combined with different design patterns, frameworks and a production strategy to create a production plan, a product line scope and core assets.

Product line scope is used to determine, which products and extensions are supported in a product family. Core assets are basic components, which are used to build different products. The core assets can also been as competitive advantages as implementation and maintenance efficiency of different products is increased. A production plan is an instruction to the correct way to derive products from the core assets. [24.]

2.3.5 Application engineering

The main activity of the application engineering, also known as product development, is to create a product based on the core assets created in the domain engineering. In the application engineering product requirements are analyzed to identify, which models from the domain engineering are most suitable as core assets to the development of a product. In practice all requirements cannot be covered by the core assets. Therefore these individual product requirements need to be refined into features that are later added to the derived core. The production plan from domain engineering is used to ensure that the features are added correctly into the product skeleton and only valid products are built. Because of the existence of product-specific requirements, the product line scope is needed to hinder the amount of unwanted rippling in a product family. [24.]

2.3.6 From requirements to a product

The main purpose of domain engineering is to develop reusable core assets, also known as product line artefacts, which are to be utilized in application engineering when deriving new products. The development needs to be managed and planned in order to be as efficient as possible. Therefore different sub-processes within domain engineering need to be specified and documented. These also need to be disclosed with different parties involved in product line engineering processes. Different sub-processes are illustrated in Figure 8.

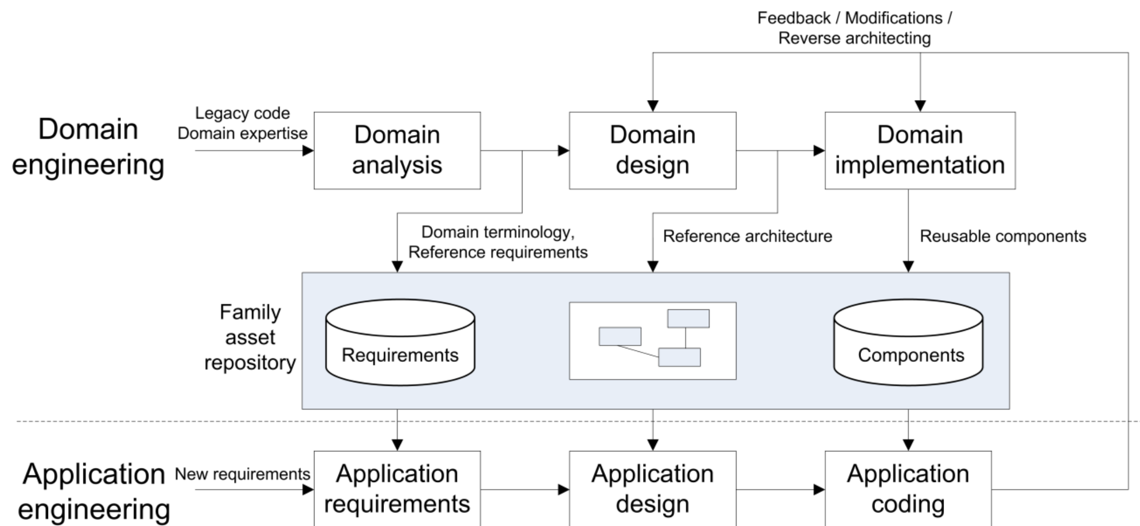

Figure 8. *Software development with product lines. Adapted from [24].*

Domain engineering involves domain analysis, design and implementation. The domain analysis covers the requirements of the specific domain including new requirements arising from application engineering. These requirements are processed into models, which are taken into account in domain designing. One of the key activities in the design phase is variability analysis, which includes requirement models being refined into architecture models. The architecture models can further be used to derive basic architecture for a product being created in application engineering.

Product derivation is the process of constructing products from product line artefacts using variation points and variants. The derivation is used to create the end product by combining core assets from domain engineering and product-specific functionality from application engineering. Derivation process needs to be supported by decision models to be effective. The process can be divided into two phases: initial phase, where initial configuration is created based on models from the core assets, and iteration phase, where the configuration is modified to match product specifications. [25.]

2.3.7 Modelling commonality and variability in product lines

Variations in product lines can be modelled similarly to ordinary software development by using parameterization, information hiding, variation points or inheritance. Each one of these has benefits and deficiencies. In most cases the amount of flexibility needed in the system determines the modelling approach. [26.]

As discussed variation occur in different forms. This creates the need for different models to address each form of variation. Commonality and variability models are used to model the core and the product-specific features in the system. Each model can further be processed into structural, functional and behavioural models. These are used to further unveil the type of the variation. [27.]

2.4 Initiating a product line

In addition to three major activities discussed before, product lines include a wide range of other activities to be addressed. The Carnegie Mellon Software Engineering Institute (SEI) has recognized a total of 29 different activities. These activities can be categorized under software engineering, technical management and organizational management. [3.] Clearly concentrating on all activities at once requires enormous amount of resources. Therefore to reduce manageable aspects, a specific approach needs to be determined. Additionally the scope for the activities needs to be narrowed down to a more manageable one. This can be achieved through using practice patterns [3, p. 356]. However these are inefficient alone. Also proper motivation is needed to complete the transition into the product line approach and to implement solid product line architecture. Architecture definition process consists of sub processes: scoping, product and feature planning, design of the product line architecture, component requirement specification and validation. This section follows widely the content of [9].

2.4.1 Approaching product line architecture

Regardless of the approach, initiating a product line depends on the starting point. In all approaches, significant effort is required from the organization adapting to product line approach. Different approaches are illustrated in Figure 9. [9.]

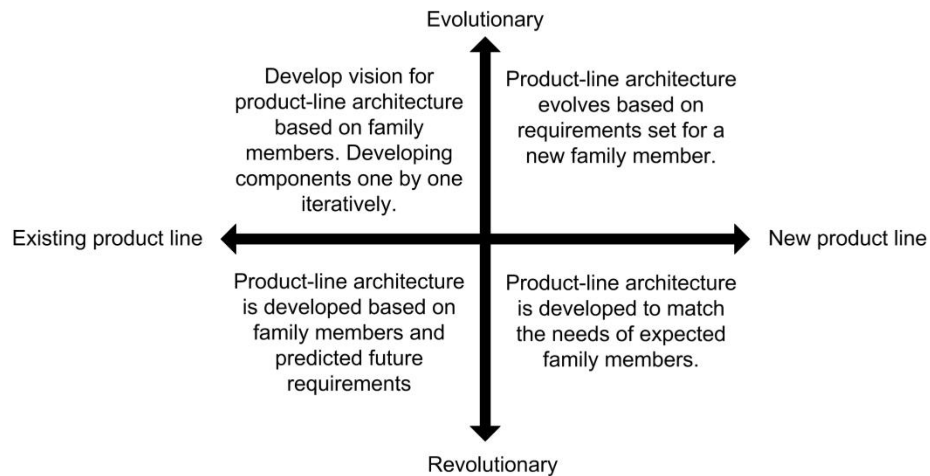

Figure 9. *Alternative approaches into product line architecture development. Adapted from [9, p. 167].*

Two fundamental decisions concerning the situation need to be made in order to define suitable approach for the product line. First, the scope of the product line needs to be defined. This means deciding whether to develop components into the architecture simultaneously, one by one, or parallel, side by side. Secondly, the decision considering the basis of the product line needs to be attended. The product line can either be based on existing products or built from scratch.

2.4.2 Business case analysis

The business case analysis is to determine the actual benefits of applying the specific product line approach compared to a possible existing approach. Additionally the benefits are always compared to the estimated cost caused by the change. A practical approach to analyze the situation from a business viewpoint is to lean to three primary driving forces: cost, time-to-market and personnel. [9.]

The costs include all expenses throughout product's lifecycle. The time-to-market measures time between the beginning of production process and the product release to market. The personnel aspects include factors such as the relevant skills, which may have significant effects to certain process decisions. For example some development may be outsourced if skilled personnel are otherwise unavailable. In general, the analysis should cover current situation, the predictions of future situations with the old and the product line approach, and investments required to make the change possible. The analysis should be clear presenting the benefits and the weaknesses of each approach.

2.4.3 Scoping

Scoping process is connected to the approach chosen for a product line and to the business case analysis. In the end it determines what products and product features are included in the product line. Pruning the products and the product features from all possible candidates in the scope requires a systematic approach. This approach includes establishing a feature matrix and a feature graph based on the products. However, before establishing either of these, different features need to be identified from the products. The features can be refined with the use of existing product documentation, such as requirement, functional and architecture specification. The approach used in this thesis is to skim through product requirements and other available documentation, and to use these to establish the feature matrix and the feature graph.

The feature matrix includes all individual products within the scope and their features. The matrix can yet be sorted and refined to illustrate commonalities and variations in product family. The matrix does not include information about the dependencies between features. Therefore the feature graph is needed to model the relations and feature types.

2.4.4 Product and feature planning

As discussed before, one of the key quality attributes of the architecture is modifiability. In product lines this shared core asset is under a constant change. This drives the necessity of estimating and planning for future changes. The changes can be caused by an old product reaching the end of its lifecycle or a new emerging product in need to be added into the product line. All the activities: removing, adding and updating a feature and its relation or a product; need to be planned properly in advance in order to have as flexible architecture as possible.

The architecture, when designed properly, is long-lasting and evolvable. To achieve this future analysis must be done by learning from the history of product family, road maps and other resources. These provide some direction where to the product line is developing and give clues what the future needs may be. These modification needs must be taken into account when the architecture is being designed.

2.4.5 Product line architecture design process

Product line architecture represents the core of all product architectures in a product family. The design of product line architecture consists of four key activities [9]:

- Deciding the approach.
- Defining the context.
- Identifying and defining of archetypes.
- Describing product instantiations.

Firstly in product lines, the product line architecture is usually extended to establish product architecture. The amount of extension needed depends on the approach, which is used to create the product line architecture. In a maximalist approach the product line architecture includes all possible features in the product family. These features can later on be removed or configured to meet the product requirements. In this approach the amount of extensions needed is minimal whereas in a minimalist approach the situation is quite the opposite. The minimalist way is to include only the most static and essential core features of the product family into the product line architecture. As a result of this the amount of extensions is much higher than in the maximalist approach. On the other hand the stability of the product line architecture created in the minimalist approach is increased due to the fact that the changes in an individual product have an effect only on the product architecture.

Secondly, the context of a product family can be very diverse. This can cause problems in deciding, which context aspects are covered by the product line architecture. If the context of product line architecture is selected in a minimalist way, the amount of reuse is greatly decreased, due to the overhead caused by different products.

Thirdly, the fundamental core concepts of product line architecture are represented with archetypes. The archetypes are commonly used for modelling the architecture and describing product instantiations in a product line. The variations and commonalities between products in the product line can be modelled with an efficient use of the archetypes. The relations between different archetypes need also to be defined. An instance of an archetype can be seen as a component in software architecture. To simplify the product line architecture, overlapping among archetypes need to be minimized.

Finally, product instantiation needs to be described in order to verify the validity of selected archetypes and their dependencies. The product instantiation process can be divided into two main activities: product specification and realization. The process is described in a production plan. The production plan should provide answers, how to build all products included in the product family. In addition the plan should both restrict building non-valid and allow building valid new products. The validity of a product stands for a possible combination of features.

2.4.6 Validation

The product line architecture describes the most significant design decisions in the product line. The architecture design phase is a relatively early phase to point out weaknesses and to make adjustments to the architecture. Later on weaknesses such as lack of flexibility in architecture may result in significant costs in product development and maintenance. This is why architecture evaluation is done before implementation.

Software Architecture Analysis Method (SAAM) [28] is one of the evaluation methods documented and used frequently in literature. SAAM concentrates in modifiability, variability and achievement of functionality aspects in architecture. The main goal of SAAM is to provide means to determine, how well the architecture serves needs of an organization. The method consists of five essential steps [28]:

1. Characterizing a functional partitioning for the domain.
2. Mapping the functional partitioning onto the architecture's structural decomposition.
3. Choosing a set quality attributes for architecture assessment.
4. Choosing a set of concrete tasks to test desired quality attributes.
5. Evaluating the degree to which architecture provides support for each task.

SAAM approach is chosen for the architecture evaluation done later on in this thesis.

3 STARTING POINT

The purpose for all automation systems is to reduce the need for human factors in a specified process. This is done with the use of control systems and different information technologies. Control systems are used to manage, command, direct or regulate the behaviour of different devices or sub-systems. In industry environment the use of automation applications is essential to relieve human labour from physically challenging tasks to a more observatory ones. Rock crushing automation stands for the use of control systems in heavy machinery situated in quarries, open pit mines and underground facilities. In practice all hardware and software need to be slightly adjusted to different use cases due to customer-specific requirements. These result in increased amount of heterogeneity in control system family that needs to be coped [29]. The harsh rock crushing environment has special needs to attend to. This chapter illustrates the environment and basic principles of the rock crushing automation software. Also the key business aspects as well as organization and processes behind software development are illustrated. This chapter ends with the illustration of a few examples of machine control systems.

3.1 Rock crushing automation

The amount and the level of automation in different industrial processes vary significantly. For example nowadays a paper factory requires only a few actions from an operator to work properly. In rock crushing and processing environment the level of automation can be very low as only part of the process chain might be automated. The rock crushing automation is perceived to be in an early phase moving from mechanization to automation. The pace of automation development is much greater in rock crushing automation than in other industrial applications. This is because several other industrial applications have been coping with similar problems already and the lessons learned need only to be adapted to the harsh environment of rock crushing automation.

A crushing plant includes stationary, portable and mobile machines. A machine consists of:

- Crushers
- Screens
- Conveyors

The main task of a crusher is to crush input material, comminution, to reduce the size of particles with different methods. However, the particle size distribution, grading, is hardly ever desirable when using a single crusher. This is why several machines are used together in a process to achieve a desirable output. Conveyors are used to move material between different types of crushers and screens. The screens are used to separate particles with different sizes and shapes to be moved to yet another machine or stock piles. In addition there is a lot of other heavy machinery moving around the plant site to feed material into the process chain and to move out the piled output material. A small crushing site with different machines is illustrated in Figure 10.

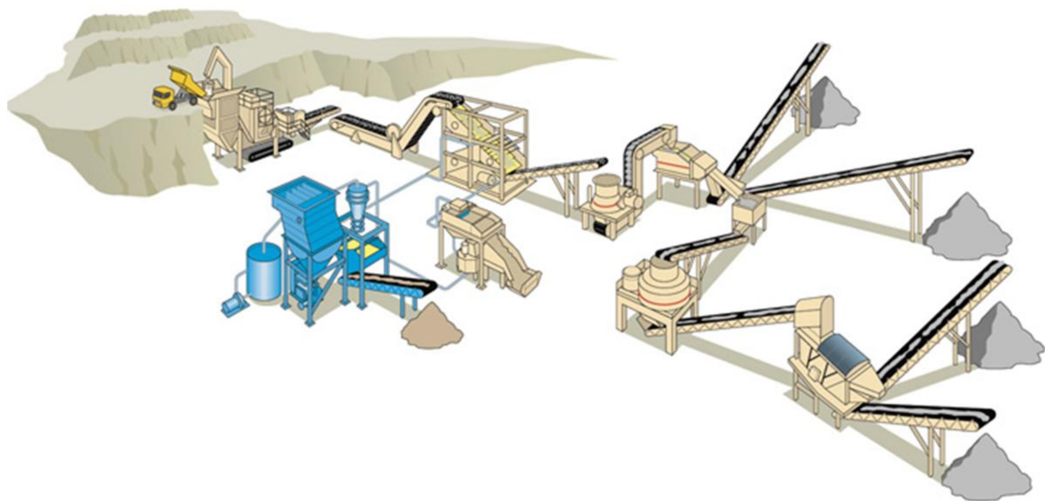

Figure 10. *Small open pit mine. [30].*

A site consists of several units working together to process input material into desirable outputs. Automation is needed to control and adjust material flow through the process chain. Thus the automation can be separated to three different levels. A trivial approach is to divide it into production control, plant automation and machine automation levels. Several sites can be managed with applications at the production control level. The plant level controllers, such as Supervisory Control and Data Acquisition (SCADA) system is required to manage the different machines and to optimize whole process chain within a site. Intelligent Controller (IC) is a product family for machine level controllers providing required information to enable the use of SCADA and other upper level systems. The tasks and different systems typically involved are illustrated in Figure 11.

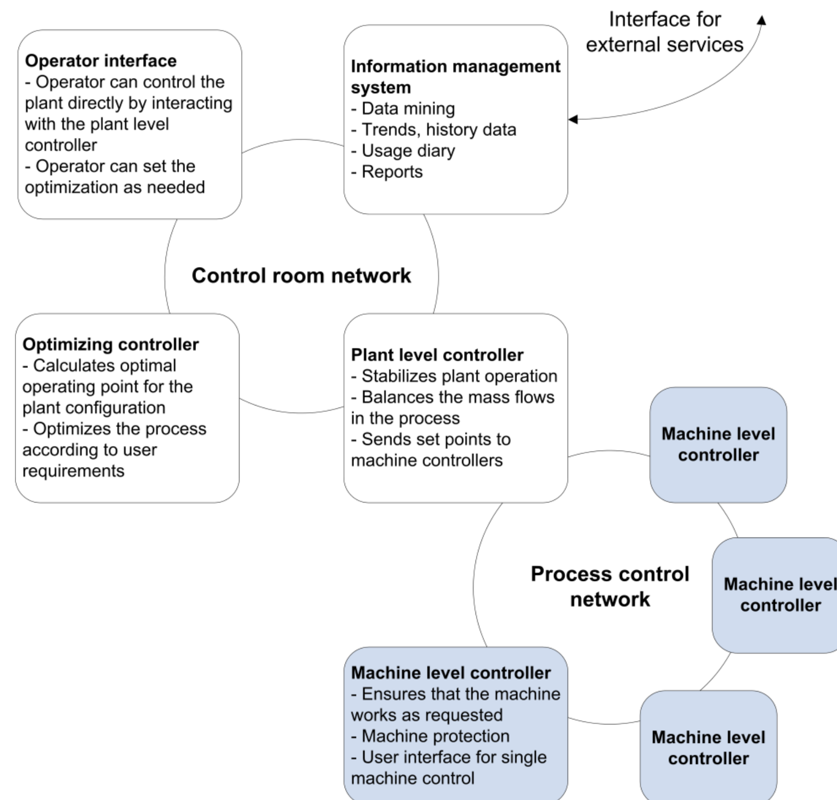

Figure 11. *Controller tasks in rock crushing automation. Adapted from [31].*

The key for success is interoperability between different systems. Thus the idea of machine controllers having similar electrification and same connectivity to upper level systems is highly admissible. Additionally the vision includes having similar documentation and UIs in all systems. This contributes into better usability and easier maintenance.

3.2 Business

The main business topics in Metso software development are to decrease time-to-market, reduce the maintenance costs and to increase overall quality. Shorter development schedules increase the amount of resources available to other on-going tasks and projects. The maintenance of products is very expensive especially with products having long lifecycles. This is why systems need to be as easy as possible to maintain and to upgrade. The long product lifecycles also increases the risk of technologies, such as specific hardware, becoming obsolete or otherwise unavailable. Thus hardware and operating system independence are seen as key concerns. Also the quality factors are important, because customer references are one of the key issues to make a selling product. In most cases customers require solid proven solutions that need to be as attractive as possible from their viewpoint. The customer-specific needs are best served with an extensive selection of easily modifiable products.

3.3 Organization

Organization creating new rock crushing automation software in Metso is distributed, sometimes even in global scope. In some products the requirement specification for the software is made on one continent with the customer, the designing and implementations by different stakeholders on a second and testing and commissioning on a third continent with the end-user. In addition hardware is usually provided by several stakeholders. For example one provides machinery and other devices and another provides necessary automation hardware for the software.

Diversity of different stakeholders involved creates challenges in management, which need to be addressed in order to make software development as flexible and efficient as possible. Product lines and product line architecture are tools, which can be used to introduce guidelines, manage variations caused by human behaviour or predilections and finally to gain stability and to standardize the development process.

One of the key concerns in global software development is how to manage and define the responsibilities of different parties involved. In order to make development efficient clear responsibilities and guidelines are required. Nevertheless the most important issue in global development is the communication between parties. The responsibilities can be defined only until a certain point, after which effective communication needs to be initiated to solve possible uncertainties and other difficulties. Global software engineering issues are discussed more in depth in [32].

3.4 Process

Process of creating and launching a new product into market consists of seven main phases separated by gates from each other. The gates are used for steering and managing the process and to give approval to proceed into the next phase. The seven phases are illustrated in Figure 12.

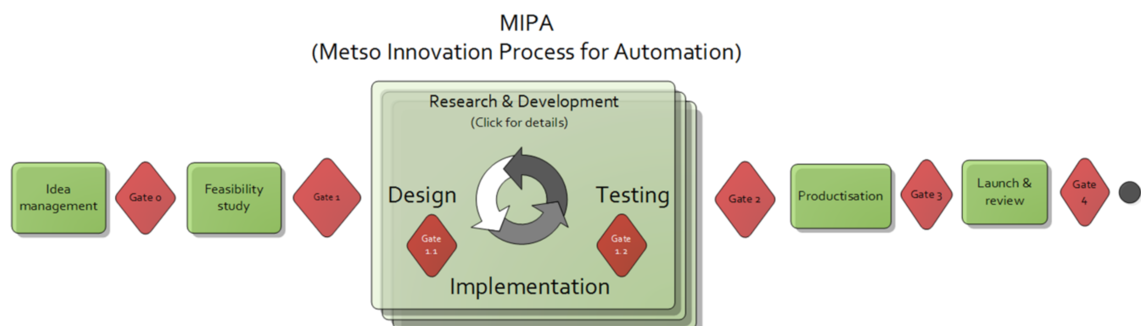

Figure 12. *General view of MIPA. Adapted from [30].*

First two phases are used to trim out unsuitable and refine suitable emerging ideas. Idea management and the feasibility studies are vital for the success of the innovation process. These phases include making preliminary analysis for the project scope, costs, benefits, required resources and timetable. The analysis done properly saves enormous amount of resources possibly lost in latter phases.

As illustrated in Figure 13, iterative software development process begins after feasibility studies. Designing, implementation and testing are done in a cycle, so that possible errors in specification, design or implementing phases can be fixed. Functional, technical and HMI specifications are done in the design phase. In the implementation phase more detailed designing is done concerning the structure of software and unit testing. Implementation includes also programming the software, after which unit and integration testing is done. After implementation phase comes testing phase, where system testing is conducted. If significant problems arise during the system testing, the iterative process moves back to design or implementing phase. Otherwise the software meets the set requirements and is ready for productisation phase.

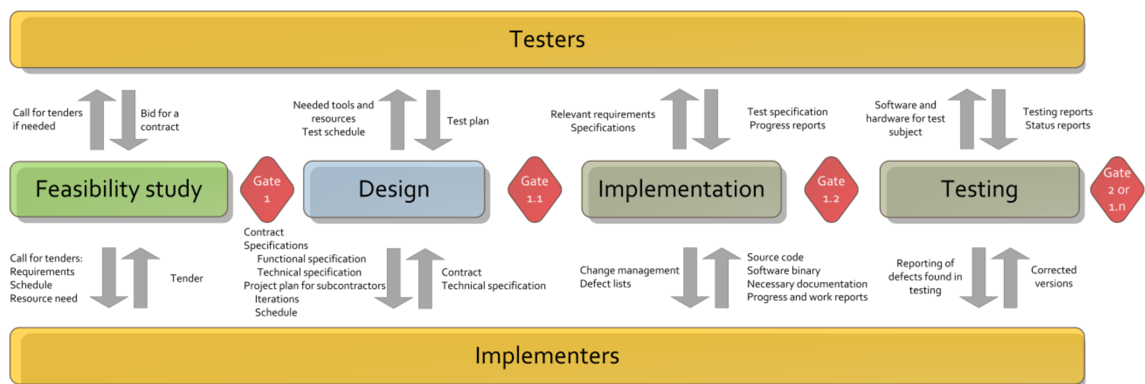

Figure 13. *Software development phases and deliverables. Adapted from [30].*

The productisation phase includes generally prototype testing with a pilot customer. Pilot projects are vital in order to get references from a working system before entering the market with a new product. The pilot projects also increase the overall quality of the product about to be released. Also an important factor is getting feedback from the pilot customer so that the product can be adjusted or modified to better meet customer needs. After the pilot project a well documented, solid and proven product should be ready for the launch and review phase. This phase includes inspecting the product, marketing and releasing it to open market.

Several inefficiencies have been noted with current development process. The ratio of reusing assets from previous projects and creating new ones is 30-70 with stationary products and 50-50 with mobile products. Also the reused assets have been utilized

optimistically resulting in errors for example in specifications. This is due to the incompatibility of mobile crusher specifications to stationary solutions. Additionally the effectiveness of the process significantly reduces if developers change. This way the previous developers leave with all the know-how and lessons learned and the new ones redo the same mistakes again. These are, among others, the motivation for constant improving of the development processes.

3.5 Technology

Technology used to create new automation software into Metso DNA environment is a mixture of understanding the basic principles of Metso DNA distributed control network and the use of different tools for development and testing. Also the long history of Metso DNA still has an effect on certain requirements in software development.

3.5.1 Metso DNA

Metso Dynamic Network for Applications, also known as Metso DNA, is an automation and information platform including process control, optimization, quality and condition monitoring. The platform consists of three essential activities illustrated in Figure 14.

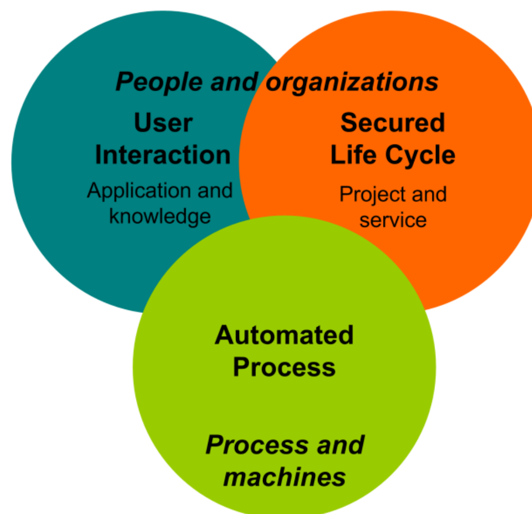

Figure 14. *Three activities in Metso Dynamic Network for Applications. Adapted from [33].*

User Interaction activity consists of tools designed to be used by different users or communities to interact with the process. Automated Process makes sure that the process and machines run automatically. Additionally its responsibilities include reliable data acquisition, maintaining comprehensive information systems and communications between different systems. Secured Life Cycle is to cope with all the challenges related to long product lifecycles. Compatibility, especially with older

systems, is seen as a key issue in Metso DNA. This is why backward compatibility of Metso DNA extends all the way to first Damatic Classic created in the 1979. The activity also provides tools for developing and maintaining Metso DNA platform.

Architecture

Metso DNA architecture is based on several concepts making sure of high availability, easier maintenance, scalability, open standard communication and efficient engineering. The fundamental idea of Metso DNA is to provide a single platform for all applications. Hardware Abstraction Layer (HAL) is used to abstract various hardware to seem alike for applications. Therefore applications are less hardware dependant and thus less vulnerable for hardware changes. Metso DNA architecture is illustrated in Figure 15.

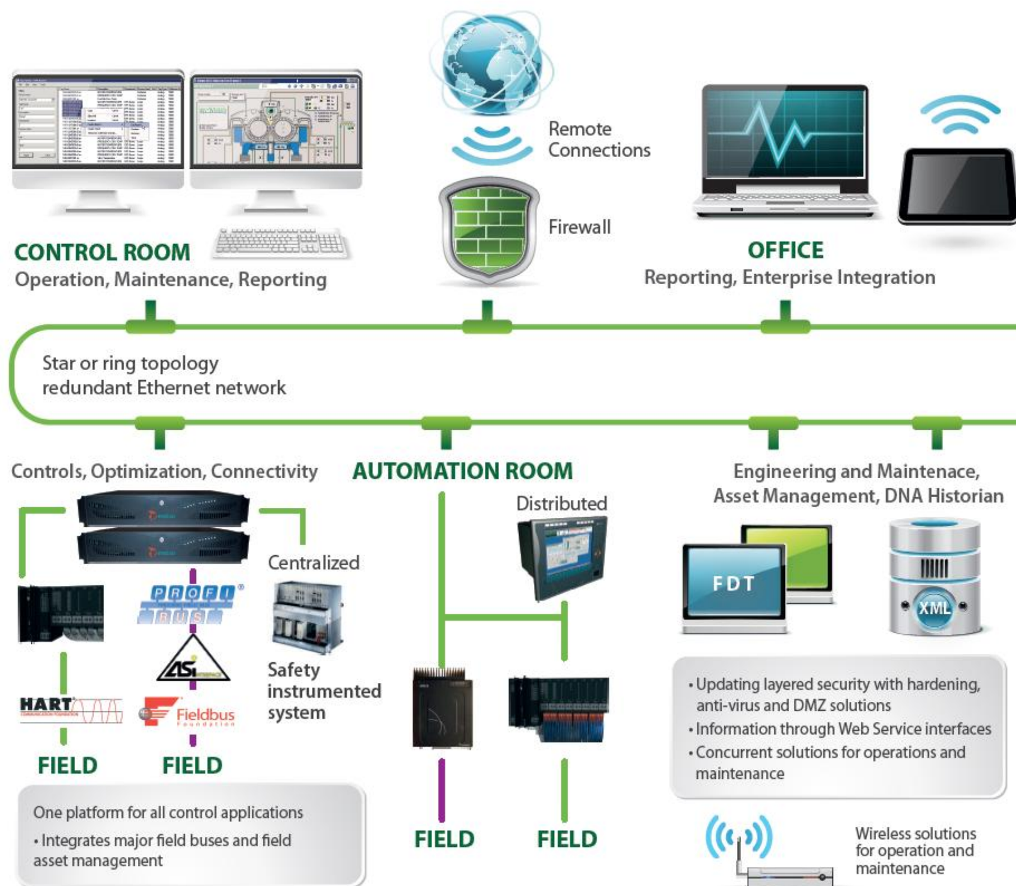

Figure 15. *Metso DNA architecture. Adapted from [33].*

Conventional Metso DNA based applications are required to be available at all times and thus systems need to be fault tolerant. The high availability is achieved through designing for redundancy, using common hardware components and implementing a secure run-time environment. For example redundancies are taken into consideration when choosing between different network topologies and designing process control

hardware. The use of common hardware components simplifies support functions as broken hardware can be replaced with identical hardware on site. Many of the conventional applications also have real-time requirements. This is why in Metso DNA there are slots for different applications. The slots are like sandboxes used to separate applications from disturbing each other. Commonly real-time systems require different tasks to be run in a specified order and within limited time frames. Metso DNA environment is used also to scheduling applications so that the availability of the system is ensured.

One platform to all applications simplifies also maintenance tasks. The easier maintenance is achieved through a simple spare part concept and with several diagnostic tools, which are designed for specific stakeholder needs. Having several diagnostic tools, serve the needs and abilities of different roles in an effective way, but increase the resources needed to maintain the tools. Basically the functionality of different diagnostic tools could be united into one effective tool, but the usability might suffer from this.

Metso DNA ranges from small machine automation applications to extensive pulp, paper and power automation applications. The amount of operating stations, process controllers, I/O-cards and other hardware can vary enormously in different systems. Also the distances within a site have an effect especially in network designs. Scalability of the system also addresses the needs to include new systems into an old one. Within the industry this means for example establishing a quality control or an optimization system to the existing process control system.

Communication and compatibility issues with other systems are also seen important as customers typically have some own systems, which should be able to attach to the Metso DNA. To make this connection as easy as possible the network supports several verified communication standards. An abstract view of Metso DNA basic principles is illustrated in Figure 16.

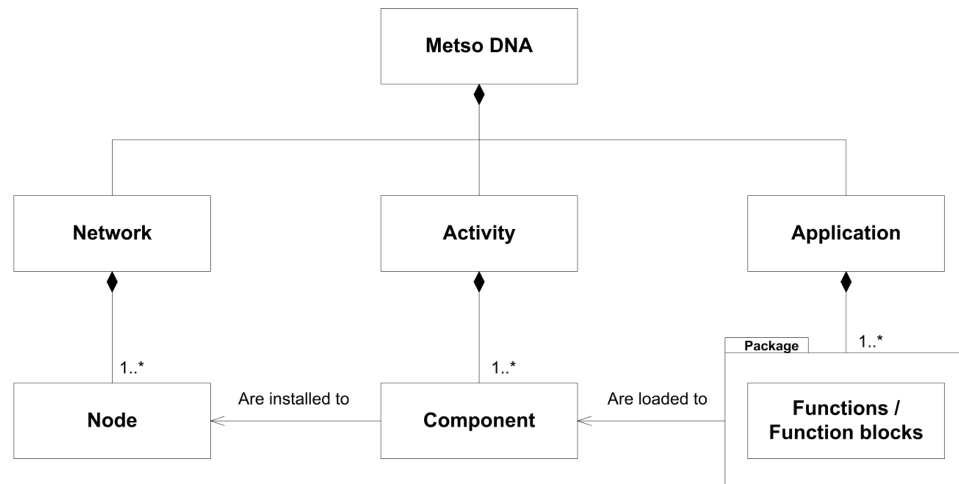

Figure 16. *Metso DNA concepts. Adapted from [30].*

Metso DNA is composed from applications, activities and networks. An application is composed from packages containing functions and function blocks. These packages are loaded into components, which are installed to different nodes in a network.

3.5.2 Developing tools

As discussed before, one of the key activities in Metso DNA is to provide efficient tools for engineering and maintenance purposes. Several tools are required to serve different user needs to produce and maintain an application for Metso DNA environment.

The layout of a system is designed using Metso Engineering Network Designer. The tool is used by a designer to illustrate all hardware, software and network aspects of a system. Software is specified only at a license level and assigned to a specific hardware in the system. The network designs provide an overview of the system and create basis for cost estimates.

The functionality of software is created typically with Metso DNA Engineering Function Block CAD. The tool has a graphical user interface (GUI) for creating software with Function Block Language (FBL). The tool includes all symbols, menus and commands that are needed to design, modify and to test an application. Most of development is done by combining adaptation and parameterization into templates from previous projects into a new one. After each project, reusable templates with potential are stored into Template library for future use. Template library is a repository for reusable assets in Metso. The templates can be created for different purposes. Low level templates usually cover the functionality of a very simple device. Thus the parameterization is simple and the reusability options are increased compared to a more complex template. A more complex template is more difficult to design and to implement, because of the amount of different parameters needed to make it as reusable as possible. [34.]

The essential parts of automation software are start and stop sequences. These sequences make sure that the process is started and stopped in a safe way. This means determining the order that each machine is started and stopped. Metso Engineering Sequence CAD is a tool, which is used to define these sequences.

User interaction nowadays is seen as one of the most significant topics in software development. Metso DNA Operate Graphics Designer is used to create the graphical interface of an application. Views are composed of elements connected to function blocks. Elements exist only for user interaction and for the visualization of the information.

Metso DNA Explorer is core of Metso DNA engineering and maintenance. The tool is used to configure all applications in the system. New applications are added to Metso DNA network by first importing them with Explorer tool. Secondly imported packages are moved to repository, a database, where all applications are stored. Thirdly and finally, application packages are downloaded to specific servers. The configuration and the update of an application are done by first retrieving application packages or at least their parameters from the servers. The parameters define the functionality and the behaviour of the application. Therefore it is essential to retrieve the parameters before making changes to the application. The application can be downloaded back to the servers, when parameter configuration is completed.

Several tools exist for software testing in the environment. FTest is used with Metso DNA Engineering Function Block CAD in testing and simulating function blocks. Debugger and WebDebugger are alternative means to gather diagnostic data from function blocks.

3.6 IC product family

The purpose of an IC control system is to protect the user and the machine, ease the utilization, the service and fault diagnostics of the machine. Additionally the system is used to extend the lifetime of a product. All these are used to maximize the productivity of a machine. The current IC product family consists of control systems with quite similar functionality, but different implementation. Variations can be found in hardware, platform and application levels. The functionalities of different applications are very similar, but mainly the structure and the visualization of information have significant differences caused by different hardware and platforms. The automation level of IC family and its relations to upper and lower level systems are illustrated in Figure 17.

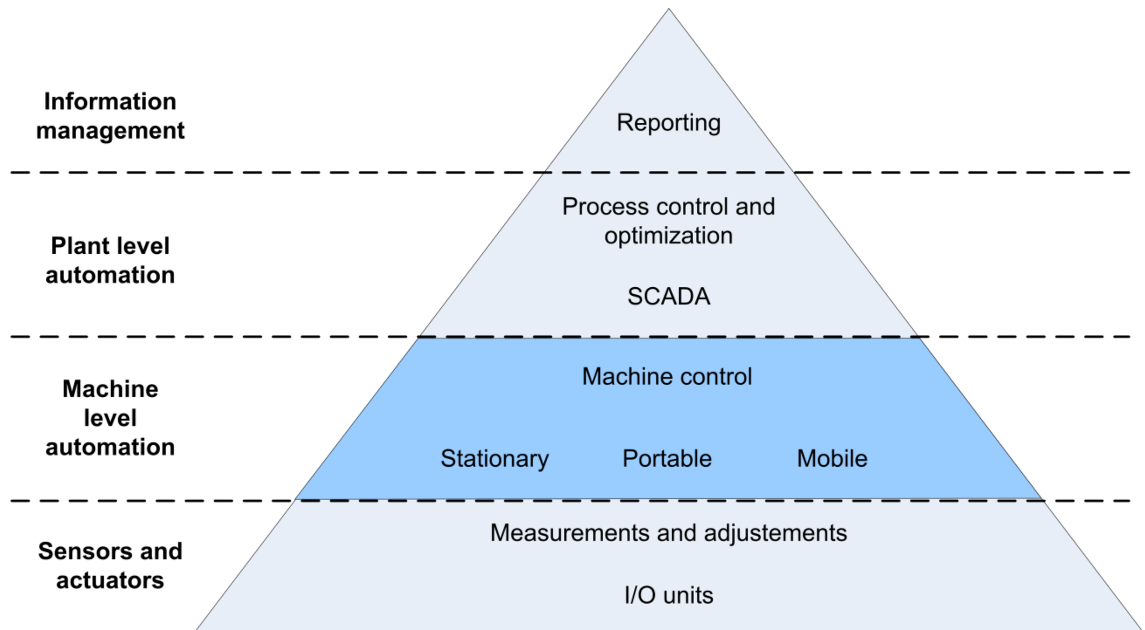

Figure 17. *The automation level of IC control systems.*

IC family provides machine control systems to stationary, portable and mobile machines. All systems have different I/O unit amounts and types units to enable machine control, but not all have upper level connections to plant level automation systems, such as SCADA. Especially earlier even simple information exchange between machines within a site was limited.

3.6.1 Overview

IC control systems are used for machine control and condition monitoring of different devices in a machine. A simple machine is a composition of different devices, such as a feeder, a crusher, an engine, a conveyor, several sensors, lubrication and hydraulic units. Also several additional devices, such as a dust removal, water spraying and magnetic separation units, can be integrated into the system. Different products in current IC control system family are illustrated in Figure 18.

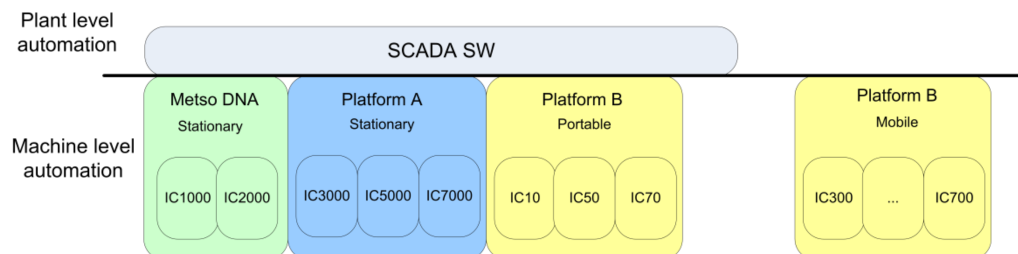

Figure 18. *IC product family.*

IC family has a great deal of hardware and software variations. The hardware variations can be explained by differences in environments. For example the ICx00 series for mobile crushers are conventionally independent machines, which need to be able to move frequently. Thus mobile machines must not have cablings and external electric cabins as in stationary solutions. This results in more demanding hardware requirements as all control modules and necessary cabling need to be attached on to the machine. Other reasons for hardware and software variations between stationary solutions are mostly because of product family history. Previously the Metso DNA platform was not applied to the rock crushing applications. Thus platforms A and B with different hardware composition were used to meet the requirements set for stationary and portable machines.

3.6.2 History

The control systems for rock crushing machines have been developing on many fronts. Usability issues such as a more user friendly UI and operating manual have been required. The maintenance of machines is more and more seen as a necessity and thus the latest demands are to add service views and other features to provide means for pre-emptive maintenance. Also the machine safety regulations have been taken more carefully than in early days.

Variability problems within IC product family have also been noted. Therefore the documentation has been required to be more uniform and well-defined. However this alone has not decreased the amount of uncontrolled variations in the product family. This is why the stress has also been set on the structure and reusability of the software.

3.6.3 Hardware

The basic hardware of a machine controlled by IC can be abstracted into a controller, local display unit and I/O units connected to sensors and actuators. The amount of different hardware depends on the complexity and the size of the system. In large factory systems the I/O unit count can be in thousands whereas Rock crushing automation system typically consists of a few hundred I/O units. An abstraction of a machine control system is illustrated in Figure 19.

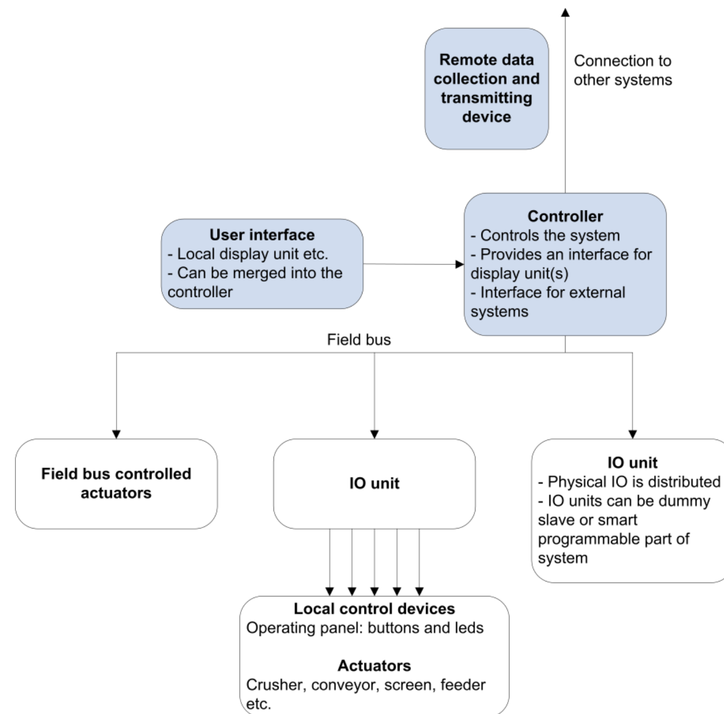

Figure 19. *Hardware abstraction of a machine control system. Adapted from [31].*

Actuators are used to modify the process output with different units within the system whereas sensors are used to retrieve data from the units. Both are connected to I/O units that mediate data through field buses to a controller. The controller or controllers contain the logic and intelligence of the system. The controllers analyze the data from sensors and make adjustments through actuators. Additionally controllers usually provide connectivity to other devices, such as local displays needed to enable the human computer interaction (HCI). The controller also provides the gateway to external systems such as other machines and upper level control systems.

3.6.4 Communication

The communication links of an IC control system can be divided into three categories: Upper, Peer and Lower level communication. The complexity of the communication needed reduces when moving from the upper towards the lower level communication. The different levels of communication are illustrated in Figure 20.

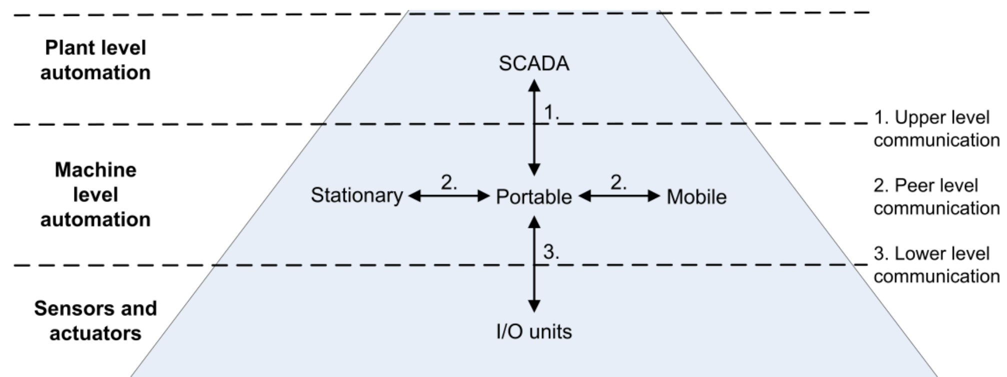

Figure 20. *Communication levels with IC control systems.*

Upper level communication

The upper level communication interfaces of a machine can be very limited if the unit is designed to run only independently. However the upper level control is needed to optimize the whole process and to make the machines in a process chain to work seamlessly. This is why the external interface for upper level control is becoming more and more important. For example if the type and the characteristics of input material vary in the process chain, particular crusher settings need to be adjusted to compensate the change. This is done in order to produce as much as possible end product with certain shape and size. The upper level control does not limit control devices to typical personal computers (PC) used in operator room, but also extends to Personal Digital Assistants (PDA).

Peer level communication

Interaction between machines is typically very simple. So far machines have only been able to give run permissions and relay alarm messages to other machines in process chain. Within the industry it is quite typical to have units from different providers in the process chain. This diversity of units, the lack of communication standards and the lack of interest adapting these create significant challenges in implementing proper communication between the units. Thus the black-box is a conventional approach used to define the interfaces of third party machines. This issue has a significant effect also on plant level control as the optimization of the process becomes more difficult.

Lower level communication

Communication between a controller and the devices subjected is the most simple of all. This is due to the fact that the measurement devices are not so sophisticated and complex as other machines or plant level control systems. The controller is only able to send command signals to actuators and receive measurements from sensors through the field bus.

3.6.5 Software

Typically the software of a machine control system consists of several key elements. The structure can be modelled with the black-box approach to define only needed interfaces and related functions or more thoroughly by modelling all significant components of the system. A basic machine control system and its software components are illustrated in Figure 21.

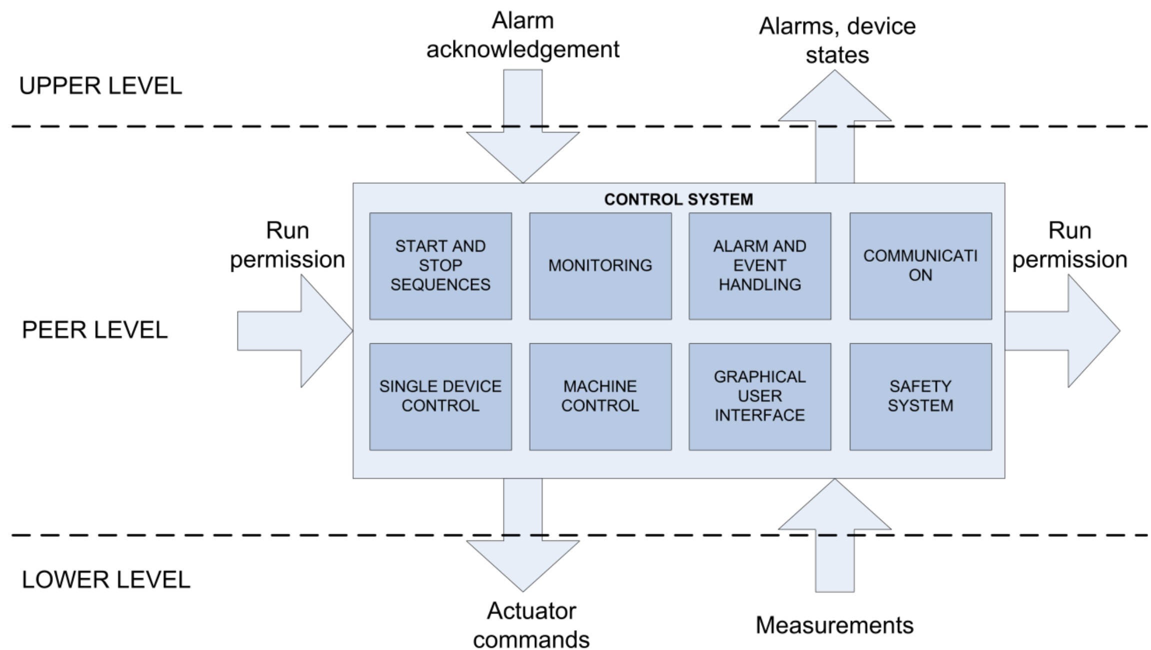

Figure 21. *An abstract software structure of IC control system.*

As seen the analysis of IC control system family reveals several logical components that are used to provide the functionality for monitoring, alarming, device control, messaging and the user interaction among others. The device control manages and controls all submitted units. This includes establishing controlled start and stop sequences. The monitoring is used to log events, warnings, alarms and unit states. The alarms are handled by another component. The communication or the messaging component is responsible for reliable and safe message delivery to upper, peer and lower level systems.

Internal business logic is similar in most control systems. However functional and behavioural changes are done with parameterization. Almost all functionality is defined at compile time and only some run-time modifications are necessary. The run-time modifications are mostly needed to change the behaviour of the system. For example if one of the devices is removed, it needs to be taken in account in start and stop sequences among others.

3.6.6 Example IC control systems

This section provides a brief description of three different control systems. The control systems are implemented using different hardware and based on different platforms among other variations.

IC7000

IC7000 is a control system for a stationary cone crusher. IC7000 controls the operating devices of the cone crusher, which include a hydraulic system, lubrication unit and different types of sensors to monitor the characteristics of the machine. The components of a crusher with IC7000 are illustrated in Figure 22.

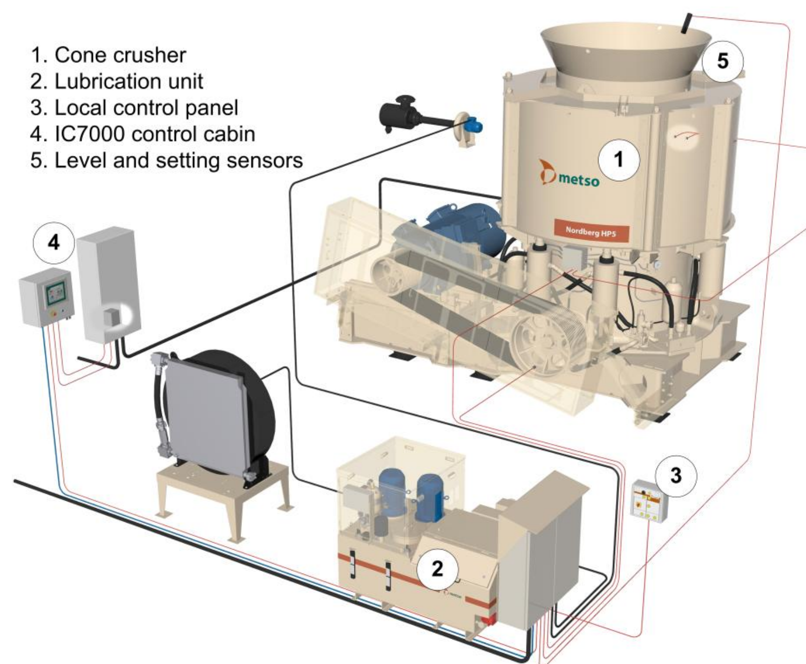

Figure 22. *The hardware of stationary cone crusher. Adapted from [30].*

IC7000 is built on Platform A, which has been used in several other applications in different industries. The software of the system is distributed between processing server in a control cabin and a simple display unit. These are connected to I/O units by a field bus solution. Different automation hardware components are illustrated in Figure 23.

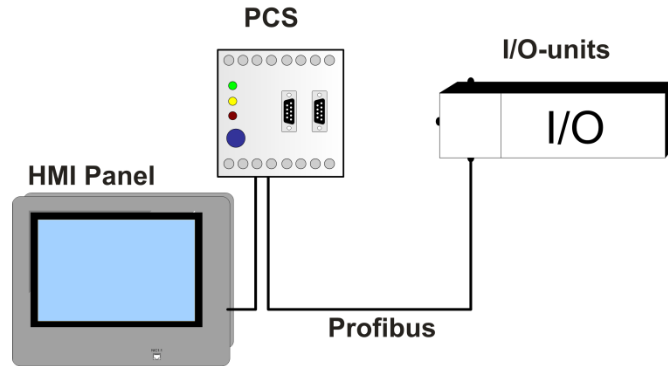

Figure 23. *The automation hardware of IC7000. Adapted from [30].*

Even though the software is somewhat distributed, the machine control is mainly centralized to the processing unit. The display unit only provides visualization of the data and the connections to upper levels. These include providing a HTTP server so that the machine can be accessed from a plant level system.

The advantages of the product are well-known hardware and the provided HTTP server access. The deficiencies are weak support for developing tools and environment, difficulties the creation of a UI and the use of third-party hardware and software. Modifications to the hardware and Platform A are difficult if not impossible to achieve with the third-party provider. These factors create challenges in application development.

IC10

IC10 development involves several parties. Functional and requirement specifications are defined by Metso, but the software design and implementation is mostly done by a subcontractor. Thus further development and maintenance of the system is more complicated than in a system developed internally. IC10 controls the operating devices of a jaw crusher that include a hydraulic unit, a feeder, a discharge conveyor and a jaw crusher illustrated in Figure 24.

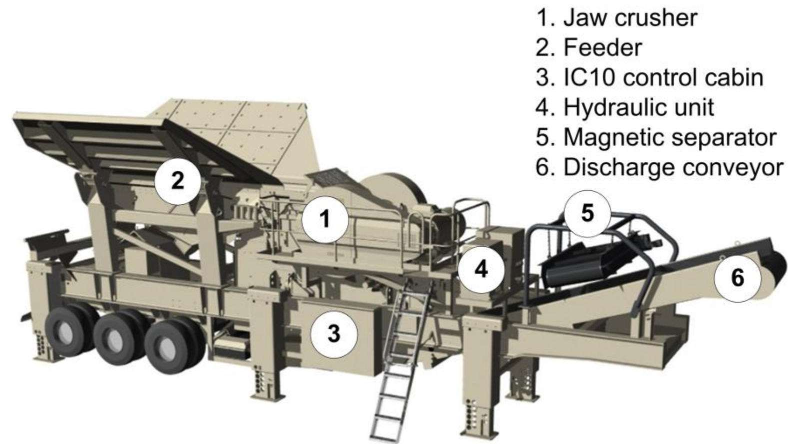

Figure 24. *The hardware of a portable jaw crusher. Adapted from [30].*

IC10 software is distributed between HMI panel, Crusher Control Module (CCM), Main Control Module (MCM), Feeder Control Module (FCM) and cOnveyor Control Module (OCM). A system layout of IC10 is illustrated in Figure 25.

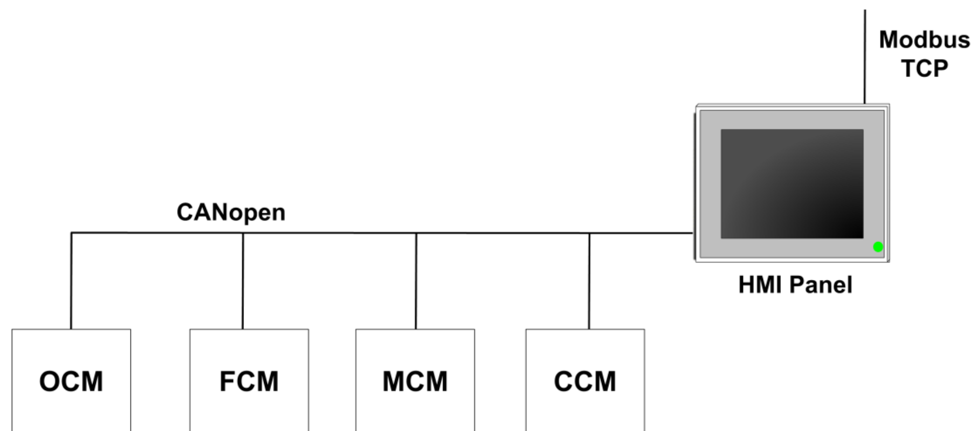

Figure 25. *The automation hardware of IC10. Adapted from [30].*

Control functions are distributed into four modules: CCM, MCM, FCM and OCM. The HMI Panel provides interface to upper levels and enables local machine control. Even though hardware used in the system is highly affordable, durable and proven, software development and maintenance confronts significant challenges. IC10 is built on Platform B, which is creates limitations in software development. A low level platform with only few libraries and only limited data storing and processing capabilities increase the amount of resources needed to develop the control system. Additionally the software development and its maintenance can only be done with a subcontractor.

The advantages of the product are well-tested, solid and well-known hardware. The deficiencies include the use of third party hardware, poorly managed product lifecycle, low level software platform and limited processing and storage abilities. The problems with the lifecycle management and the platform are due to the fact that the interests of the hardware provider do not extend into providing extensive software libraries and lifecycle support.

IC1000

IC1000 is the first Metso DNA based control system in the product family. IC1000 is a solution for same crusher types as IC10 with a benefit that subcontractors are no longer needed in the software development or in maintenance. IC1000 controls the operating devices of a jaw crusher, which include a hydraulic system, lubrication system, different types of feeders, several conveyors, a scalping screen, a magnetic separator, water spraying system and a bottom heater. A control centre conceals basically all intelligence of the system. The core of the system is a small rail-mounted controller (ACN SR1) and the I/O units, which provide the connections to actuators and sensors. A panel pc for GUI is a typical addition to the system even though not a vital one. The system can be controlled remotely from plant level or by using local control panel for the very basic operations. The functional layout of IC1000 is illustrated in Figure 26.

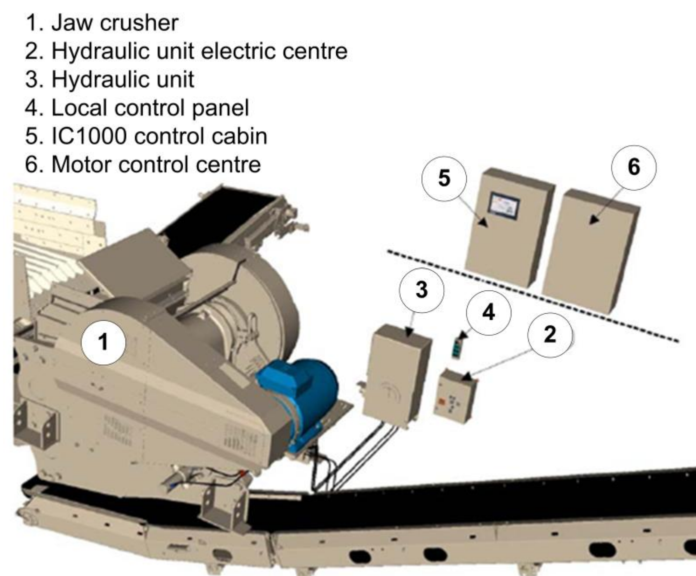

Figure 26. *The hardware of a stationary jaw crusher. Adapted from [30].*

Software is distributed between the panel PC and the ACN SR1 so that latter contains all of the essential components of the system. Thus the panel PC and GUI is not needed to include at all times. The hardware and the distribution of automation software in IC1000 are illustrated in Figure 27.

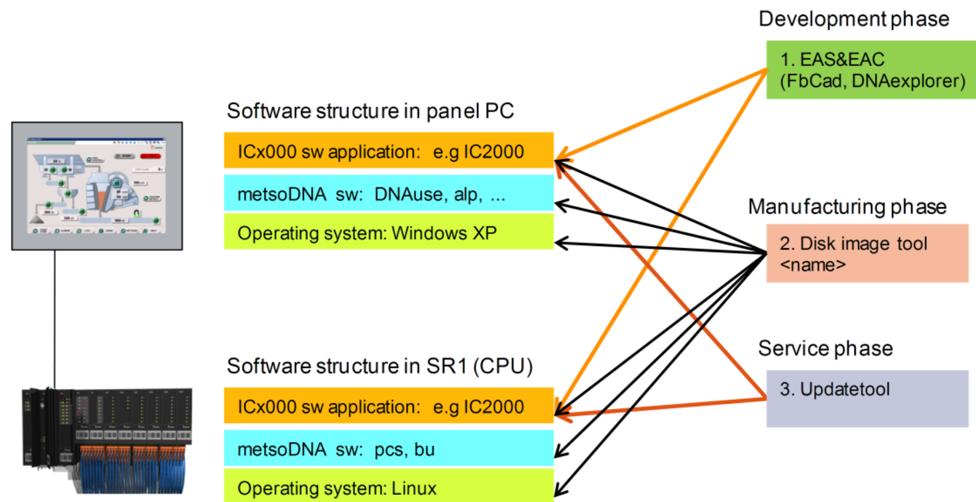

Figure 27. *The automation hardware of IC1000. Adapted from [30].*

ACN SR1 has Metso DNA software, including process control server (PCS) and backup server (BU) packages, running on a Linux operating system. IC1000 application is built on top the Metso DNA that handles the basic communication within Metso DNA network and the hardware abstraction for applications.

Software on the panel PC is built on a Windows operating system. The software includes Metso DNA software for GUI (DNA Operate) and an alarm processing server (ALP) packages. IC1000 application is built on top of Metso DNA similarly to ACN SR1.

The advantages of the product are own hardware and tools, the ability to develop the environment. The weaknesses are difficulties in creating the UI and increased hardware requirements by Metso DNA.

4 FROM INDIVIDUAL PRODUCTS TO A PRODUCT LINE

This chapter is based on the structure of Chapter 4 in [9]. Firstly, a feature matrix of IC product family is introduced. Secondly, the impacts of product line approach are analyzed from a business view. In section 4.3 current and estimated future needs are introduced and their potential impact into product family is briefly evaluated. After these a version of the product line architecture is introduced. Then the organizational and process changes needed to support the product line approach are explained. Finally a few guidelines how to implement the change towards a product line approach is given.

4.1 Scoping

Scoping process to acquire requirements and features for product line architecture is conducted by skimming through requirement and technical specification as well as UIs and manuals of different systems. Recognized requirements and features are then categorized according to possible variations. Finally these are visualized through a feature matrix and a feature graph.

4.1.1 Common requirements

Main architecture principles from Metso DNA provide the basis of requirements for IC family architecture. In addition several IC family specific requirements are recognized.

Human-machine interaction (HMI)

Human-machine interaction (HMI) or more commonly known as Human-computer interaction (HCI) is one of the focus areas in the IC family. Usability issues with UIs have been studied extensively to achieve intuitive, easy to learn and effective control systems. These issues are promoted by “Same look and feel” -theme, which drives similarity in control system UIs. A more detailed study regarding usability issues with IC family is reported in [35].

Performance

Price-performance ratio is one of the main drivers in IC development. Automation is still seen as a nice addition on the machine, but is not expected to show in the price tag. Thus the procurement of automation related hardware is done with as low costs as possible, but still getting hardware that meets the set requirements. A high Ingress Protection (IP) class is one of strict requirements for hardware because most of environments include temperature changes, dust, vibrations, and moisture that could harm the hardware and thus prevent the machine from working properly. Typical requirements for hardware are operating temperature from -25°C to +60°C, vibration resistant of 5G and IP65, which ensures product to be dust sealed and protected against low pressure jets of water from all directions.

The aim for low costs has also an impact in software development. Hardware decisions limit the choices in software development. Additionally software development should consume as few resources as possible. To minimize the resource costs the architecture and other reusable assets must be suitable and flexible for different projects. Other concerns address availability, reliability and real time issues in control systems. These are taken seriously especially when safety is a concern.

Configurability

IC family history suggests that most of the customers have specific needs or preferences concerning the software or the hardware configuration. Usually software needs to be configured at least in two phases: compilation and run-time. For compilation the basic features for the software are chosen. These features can further be modified during run-time to achieve required functionality. These adjustments can be seen as variations in the software. Currently the behavioural adjustments are done mainly by parameterization and the functional changes are the difficult ones to implement. Different product configurations need to be supported in order to meet the customer demands. The customizations in products continue also after the handover of a product. This upraises more challenges especially in product and application lifecycle managements (PLM and ALM).

Development

New prototypes of Metso DNA based control systems have been developed quickly without significant problems. However the amount of reusable component used and created is not seen to be sufficient. Even though Template library system is available and designed for planned reuse of software components, it has not served its purpose so far within the product family. A reason for this is that product specifications are done without references to prior projects. This traceability problem has led to a situation

where specifications and thus features vary even in very similar cases. Other difficulties include UI designing and testing. Both of these consume a significant amount of time from projects. In addition to time consumed, the testing has been troublesome because of the fact that the automation software and machine hardware are tested first time under the eyes of an end-customer. Proportions of hours spent in control system development are illustrated in Table 4.

Table 4. *An hour estimate of an IC software development project.*

Task	Percentage of total hours spent
Specification, design and implementation	34%
Visualization and localization	27%
Testing	22%
Corrective actions	16%

Significant amount of time is consumed by common development activities, but a lot is also wasted in visualization and localization. The effort invested in visualization and localization issues should rapidly decrease after the first project if these are done properly. These aspects should be highly reusable from a project to another especially when the "Same look and feel" theme is introduced. The amount of hours put in corrective actions is also one of the key issues when optimizing software development. One of the disturbing issues currently is insertion of ad hoc solutions into well-designed and implemented software. Gradually, when enough ad hoc additions are done, the well-designed software structure is lost. These emphasize the need for effective change management. The change management is a managerial function to cope with different modification needs and to reduce the amount of ad hoc solutions being implemented.

Maintainability

The importance of support functions in software engineering is commonly recognized. However the role of the support stands out especially in products with a long lifecycle. More and more updates are needed because of hardware changes, some new functionality is required and others become obsolete and so forth. Even personnel working with the product may move on and leave a gap in know-how. This is also the case with IC software, which is required to last almost as long as the crushing machine. Thus logical software structure and modifiability are key issues from a maintenance perspective. The software must be configurable to new emerging technologies, such as new types of actuators and sensors, which are under constant change. In short the maintenance tasks of an IC control system is mainly creating sporadic bug fixes and reacting to new technologies with add-ons.

Health-Safety-Environment

Health-Safety-Environment (HSE) tripod is a key concern within the industry. The safety and security issues require special attention when heavy machines are mixed with a human factor. One of the main goals for IC family is to get site workers as far from the machines as possible. Remote control for machines on site is a must have characteristic of a system. Also the conditions of a site worker significantly improve when moving from a dusty and noisy outdoor environment to a clean and silent control room. There are also several standards and directives instructing how the machine safety needs to be taken in account to avoid dangerous situations and to achieve constant safe use of the machinery. An approach is to evaluate safety-related systems according to IEC 61508 [36] standard and to classify them by Safety Integrity Level (SIL). Also relevant is the mechanical engineering directive 2006/42/EC on machinery when market scope includes Europe.

4.1.2 Product feature matrix and graph

The results of the scoping process are refined into a feature matrix illustrated in Table 5 and further to a feature graph illustrated in Figure 28. These illustrations are incomplete due to the extent of control system hardware and software. Thus the amount of different hardware and software aspects is reduced to the extent, which gives the base for different architecture decisions later on in this chapter. The refining from documentation into the feature matrix is done both at conceptual level and then more in depth analysis. The feature matrix is created by first analyzing available documentation at conceptual level and secondly analyzing the concepts and their context more in extensively.

The feature matrix however does not illustrate the variations within different concepts at the table below such as hardware variations in display units and thus software variations in GUIs. Also a significant amount of hidden variations are located in the machine controls, which depend on the current machine configuration. Different types of crushers require various sensors and actuators and thus trends, reports, alarms and other features vary.

Table 5. *Partial feature matrix illustrating supported hardware and features by control systems for different crusher types.*

	IC1000 (Jaw)	**IC2000 (Impact)**	**IC3000 (Impact)**	**IC5000 (Cone)**	**IC7000 (Cone)**	**IC10 (Cone)**	**IC50 (Cone)**	**IC70 (Cone)**
Platform, Tools and Connectivity								
Platform A			x	x	x			
Platform B						x	x	x
Metso DNA	x	x						
Atools for engineering						x	x	x
Btools for engineering			x	x	x			
Metso DNA engineering tools	x	x						
CAN/CANopen	x	x	x	x	x	x	x	x
Profibus			x	x	x			
CAN radio control						x	x	x
Upper level communication	x	x	x	x	x	x	x	
Units and devices								
Display unit	x	x	x	x	x	x	x	x
Localization and general UI settings	x	x	x	x	x	x	x	x
Touch screen calibration			x	x	x			
Lubrication unit		x	x	x	x		x	x
Hydraulic unit (Manual/Automatic)	x	x	x			x		
Winter mode(inc. Timer)	x			x	x		x	x
Feeder control (on/off, Type A, Type B)			x	x	x		x	x
Feeder control (Manual, Automatic)	x	x				x		
Feed heater	x					x		
Conveyors (with control)	x	x				x		
Conveyors (only information)			x	x	x		x	x
Belt scaling for conveyors	x	x	x		x	x	x	x
Water spray control	x							
Magnetic separator control	x	x				x		
Crusher control								
Manual (local control box)	x	x	x	x	x	x	x	x
Auto (including PA control)	x	x	x		x			
Local (GUI)	x	x	x	x	x	x	x	x
Crushing modes								
Cascade (throughput adjusted)			x					
Power (power level)					x			
Load (power and pressure levels)				x			x	x
Setting				x			x	x
Cavity level	x					x		
Machine control								
Start / Stop sequences (configuration dependant)	x	x	x	x	x	x	x	x
User levels	x	x	x	x	x	x	x	x
Diagnostics	x	x	x	x	x	x	x	x
Parameters	x	x	x	x	x	x	x	x
Alarms, events and warnings	x	x	x	x	x	x	x	x
Trends and reporting	x	x	x	x	x	x	x	x
Crusher type-specific								
Calibration , Settings, measurements	x	x	x	x	x	x	x	x

A more in depth analysis based on Table 5 reveals the relations of different product aspects and their variation types. The feature graph in Figure 28 illustrates these aspects more thoroughly.

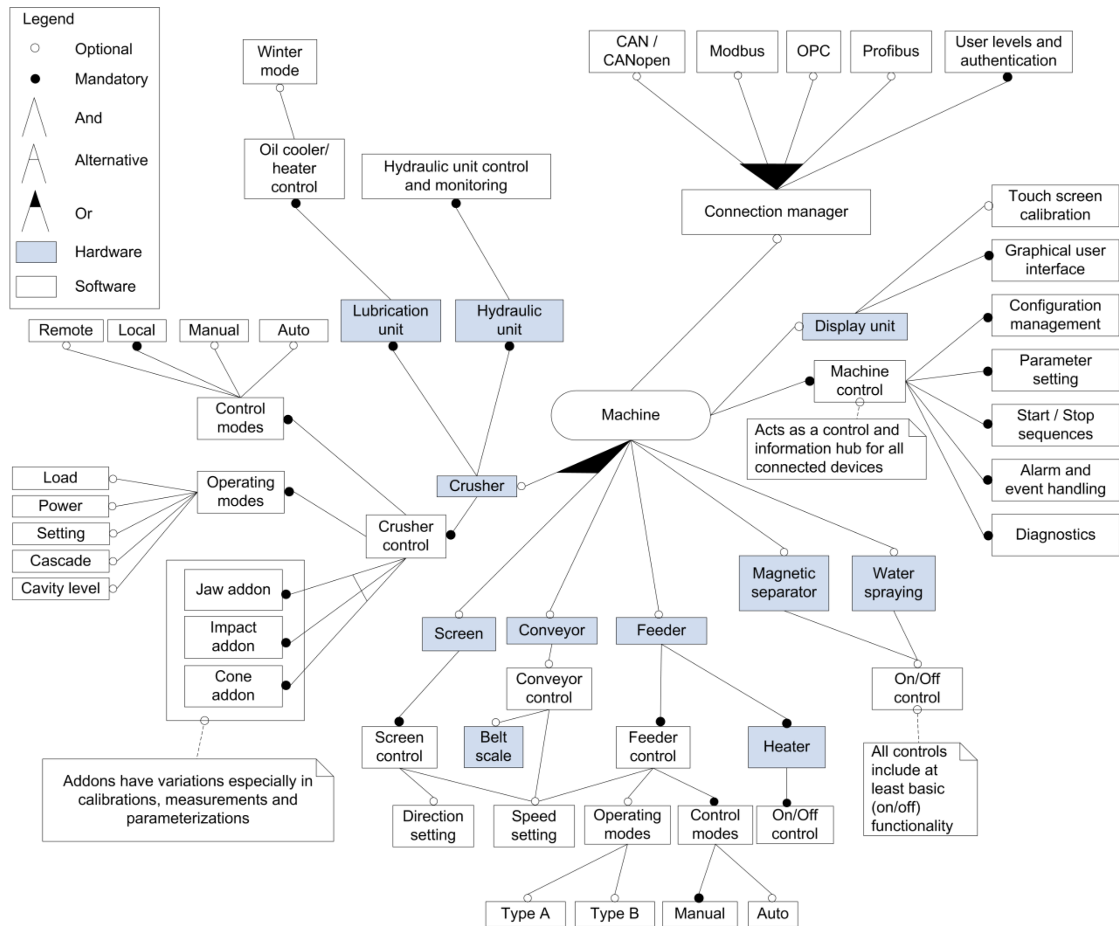

Figure 28. *Partial feature graph of current IC family.*

As shown in the graph, a machine consists of several units that all have sensors and actuators to provide functionality. Currently the core of the machine is a crusher unit, but also screens have increased their intelligence due time. This is why a machine is defined to consist of at least a crusher or a screen and possible other more simple units. A display unit is described as an optional device because of the fading need for local control panel. This is due to the trend for having a plant level control and requirements to optimize the whole process chain. Additionally UIs are reported to be troublesome to implement because of localization, common look, primitive methods and other reasons.

The crusher must have some engine, lubrication and hydraulic unit. These units as well as the crusher itself require some kind of control. In addition these controls commonly have different modes or states. For example lubrication unit has winter mode to enable its functionality even in cold conditions. Respectively the crusher has several different

control modes to determine whether the machine is controller through local display (Local), local control panel (Manual) or automated (Auto), for example remotely. The control modes are used to enable only one control mode at a time and thus to achieve safe usage for the crusher. Safety standards demand that a machine can be controlled from one and only one control location at the time, but one must be able to stop the machine from any possible control locations at all times. Operating modes define how the crusher is run. For example Load mode tries to keep the crusher on maximum capacity at all times where as Power mode changes the crusher settings in order to keep the power consumption at a certain level. The crushers also have type-specific software, because of the differences in hardware and basic principles. For example the crushing principle of a primary jaw crusher differs significantly from a fine tertiary cone crusher. Also calibrations, sensors and other characteristics vary. This is why a basic crusher control is extended with add-ons.

As described before, a machine consisted of several units. This requires a general management and control. This is why all machines have machine control, which is used to provide the common functionality to different machines. Configuration management is used to keep track of all connected units so that these are taken in account in start and stop sequences among other features. Parameter settings are used to change the behaviour of the machine and different units. For example an upper level control may find the need to change the parameters according to a specific recipe. The recipes are used to modify the settings on different machines in the process chain to achieve desired end products. Alarm and event handling is needed to keep history log and to relay fault information to upper levels. Respectively Diagnostics decrease the effort needed to find the reason for the fault situation.

The machine also needs to be able to interact with other machines, actuators, sensors and upper level systems. Thus Connection manager is established to handle the communication with different levels. The requirement for better compatibility and extensive interface emphasizes the importance of Connection manager. Better connectivity and increased functionality through an interface also create a need for proper authentication and user levels. Previously these have been implemented mainly for local UIs to hide advanced settings and features, intended to support engineers, from a basic user. With improved connectivity to upper levels, the users must be recognized in order to ensure safe and secure working environment.

4.2 Business case analysis

The change from current product portfolio into a product line based set involves several business questions and compromises. The most significant drivers for the reformed control system architecture design are simplicity and compatibility. Features that no longer are seen as a must have are extracted from the product line architecture. Additionally some features need to be integrated into the architecture in order to meet estimated future needs.

The product line approach, in this thesis, is Metso DNA based though it could as well be based on the two other platforms or even a whole new one. A more detailed study needs to be performed to find out the customer preferences with different systems so that the true impact of harmonizing product portfolio into one platform can be evaluated. Choosing Metso DNA as a platform requires also work because most customers have fear for new or unfamiliar technologies. Additionally customers demand proven solutions. These two factors emphasize the importance of pilot customers so that the proven solutions and new technologies can be introduced to the market area.

Need for proper process automation has been increasing within the industry. More and more inquiries are made by customers to find out how to increase the efficiency of their crushing sites. This includes both minimizing expenses and maximizing throughput. In addition the process needs to be as predictable as possible, because downtimes are unaffordable. In most cases the answer lies in process optimization, which requires different machines to be working as one. The diversity of machine manufacturers within a single site and the lack of communication standards increase the need for better compatibility and connectivity of a single machine. The interfaces to other machines and especially to plant level automation systems are needed to optimize the process. Compatibility achieved through proper control system interfaces can become a cutting edge over other products in the market area. This is because most customers have old sites and are only interested in buying automation tailored into their current configuration and possibly buying a few machines to replace the obsolete ones. At the moment plant-size deliveries are concentrated mostly to developing economies. By providing interoperable machines the sales and thus the amount of references are to increase. The references can further be used to distinguish from competitors and thus support the sales. However one must remember that upon the arrival of the communication standards the cutting edge becomes slowly blunt as less and less machines will require tailoring in order to be connected into an upper level system.

Vendor independence, or at least maximizing it, is another key issue in the business scope. So far control systems are developed only partially within Metso. Outsourcing is a trend, which has been overdone recently in many organizations. This has reduced the overall effectiveness and profits of organizations. This business topic needs to taken in

account, when deciding whether the product line approach is suitable for current situation. The product line approach can be almost impossible to enforce in software development when only requirement specifications and possibly integration testing is done within the organization, especially, when stakeholders vary in every delivery. In an ideal situation for product lines, all development activities are done in the same organization to minimize the friction between stakeholders.

4.3 Product and feature planning

The first steps towards plant level automation systems have been taken. This advancement causes some aspects to become obsolete and creates a need for others into the architecture. Earlier control systems were designed to be as robust, reliable and independent machine as possible. The requirements for a robust and reliable system have stayed the same, but the overall environment has been under constant change. Earlier the independency of a machine meant for example having a local display for HMI and a database for each machine.

Local displays next to each machine become unnecessary when plant level control is implemented. In some solutions local displays are extended from the machine to an operator room. This includes having several small robust touch screens, one for each machine, to enable process control. However in this case the operator is demanded to be the mediator between machines. The problem is similar to everyday problem of having several remote-control devices in the living room to enable the control of different devices. This is far from automated process control. Fortunately nowadays a majority of stationary and even most portable plants have some kind of centralized process control system. This enables the possibility of moving most if not all of the machine data and its visualization to the upper level, where UIs can be provided for example through light web clients. This way a significant amount of work is reduced in control system development and the “Same look and feel” UI principle can be enforced more effectively. Additionally the hardware costs are cut down as there is no longer need for a robust, high IP classed display and a PC for database next to each machine. Even in the service purposes the display unit beside a machine can be replaced for example by a tablet device.

Previously, data logging and alarm handling created a need for a local database. With a plant level automation and centralized database there is no longer need for several local databases. Each machine only needs some storage to buffer its data in case the connections to an upper level system are offline. The centralized database approach also simplifies the bigger picture as all data can be mined from a single location instead of skimming through numerous machine control systems. This approach also creates a basis for future maintenance applications that enable pre-emptive maintenance based on mining data from all sites around the world.

4.4 Design of the product line architecture

Product line scope is limited to stationary and portable machines. Thus mobile machines are left out of the product line architecture. However if product line approach is seen efficient in practice the mobile control systems may also be migrated into the product line. Architecture for mobile crushers has been modelled from several viewpoints before by Taavi Kytömäki [37], Ville Lehtinen [38] and Jukka Kaartinen [39]. Kytömäki addressed the hardware aspects whereas Lehtinen and Kaartinen concentrated into the software aspects of architecture. The research in mobile crushers also contributes in the architecture of stationary and portable applications. The very basic view to the product line design and its connections to other systems are illustrated in Figure 29.

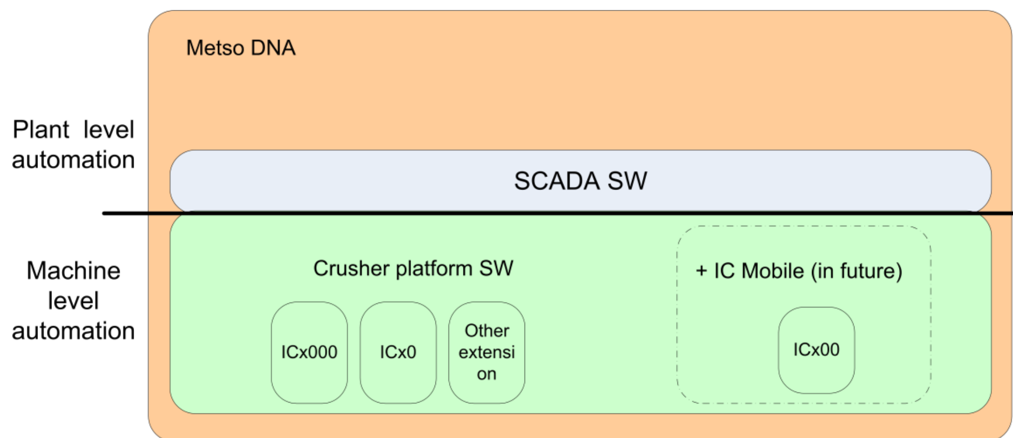

Figure 29. *IC product line overview.*

The product line is based on the Metso DNA environment. The product line consists of Crusher platform and application engineering units. Platform unit provides the core assets to the application engineering unit that is needed to develop the actual control system software. The figure above illustrates stationary ICx000 and portable ICx0 applications still separate, but in practice these can be quite similar. Previously, the portable applications have been seen similar to the mobile case, due to the mobility option of machines. Thus a more robust hardware has been used resulting in software different from stationary solutions. The fact that portable units are relocated at most few times a year, the lack of ability to move independently and the crushing site being quite similar to stationary bring the portable machines closer to the stationary ones. Therefore the portable machines can be seen also as stationary on wheels from a software viewpoint. The figure above also enables the possibility of integrating the mobile applications to the product line so that one day all crusher control systems could be based on the same core assets.

Logical view

The new proposal for architecture includes only machine management and none of visualization aspects as discussed before. The common components of all IC control systems include the communication, diagnostics, machine controls, unit management and data logging. The future needs for more open interface emphasizes the importance and complexity of CommunicationManager, which is responsible for user management and authentication, protocols and providing an extensive interface to upper level controls. The importance of CommunicationManager increases as more and more machines are required to be accessed remotely and connected to public network such as Internet. Therefore the need for proper security is emphasized as machines become more vulnerable against malicious software.

Local display unit and all related software are excluded from the design due to the fact that GUI software can be created to access IC through CommunicationManager. AlarmEventHandler is still relevant component for data logging and handling different events. Some events and alarms are handled internally and others are mediated to plant level controls. LogManager is subjected to AlarmEventHandler and used to store all relevant data into database. Database capacity of a single machine is somewhat limited and thus the database is required to be synced with the upper level system periodically. The database operations can be done with AlarmEventHandler and LogManager components. MachineController component is the core of all functionality. It is responsible providing the external functionality to upper level systems and internal functionality such as the safety features, which are not be dismissed.

Additionally the architecture has three somewhat optional components. ProcessOptimizer is a new addition to the architecture. It is responsible for adjusting different units to work according to recipes provided by RecipeManager. The recipes are commonly used to optimize a plant to produce desired products, but the fact that a large and complex portable machine can include for example a feeder, primary and secondary crusher, a screen and several conveyors, creates the demand for recipes. The third optional component is AbstractUnit. AbstractUnit is marked as optional because of the variability it withholds. The logical view to IC product line architecture is illustrated in Figure 30.

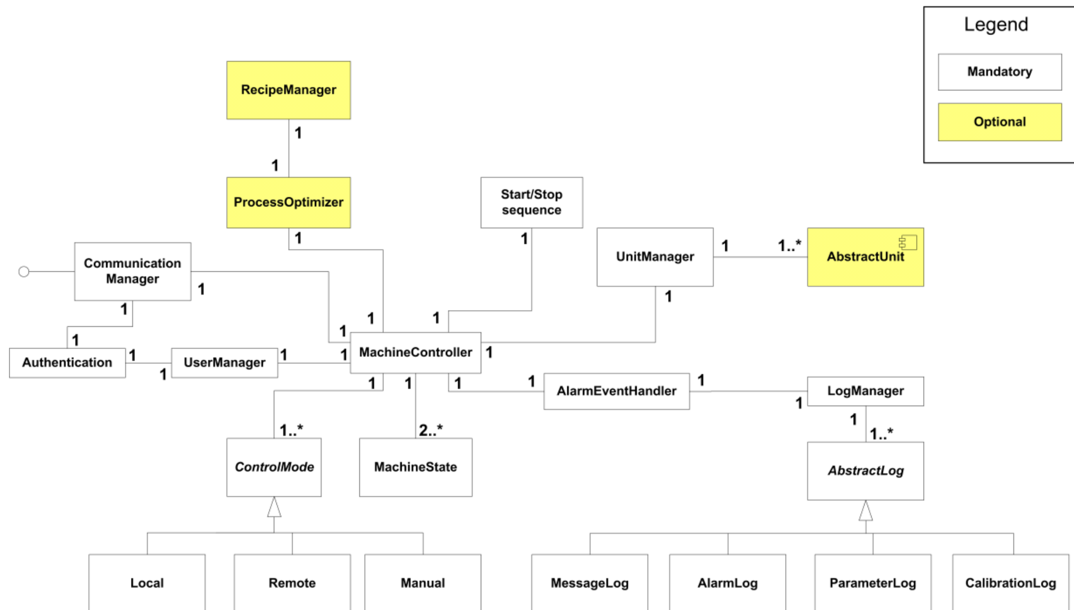

Figure 30. *IC product line architecture as a class diagram.*

IC control system is designed to manage a machine consisting of different units, which all have different types of actuators, sensors and other aspects. AbstractUnit is based on the principle of that every unit has some basic UnitControl, State, SensorManager and ActuatorManager. Additionally the unit has some greater functionality, which is derived from the basic functionality of AbstractUnit. AbstractUnit is illustrated in Figure 31.

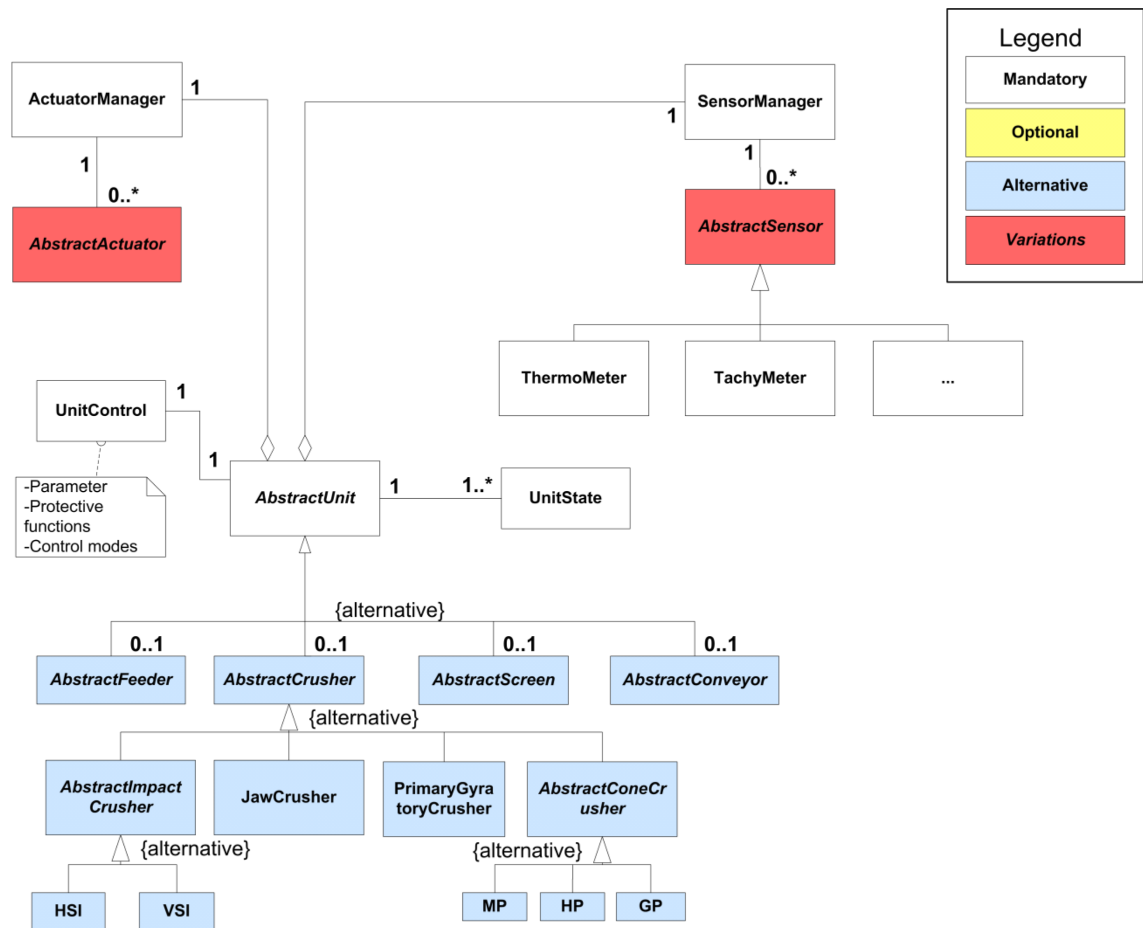

Figure 31. *A structure design for an abstract unit controlled by IC.*

SensorManager and ActuatorManager are responsible for controlling and managing different types of sensors. These provide an abstraction for UnitControl. Actual AbstractUnit is alternatively a feeder, a screen, a conveyor or some type of crusher. All of these have variations in UnitControl and some have more or less complicated states. The figure above does not illustrate the different units extensively, but gives basic principles for harvesting the similarities between units to different abstractions.

4.5 Organization

From an organizational viewpoint software architecture should provide better means for communication among stakeholders. Typical problems, such as mistakes in the requirement specification phase, can be avoided if the stakeholders share the same language. In product families, sharing the same language is vital to avoid rippling effect, which ends up creating un-controlled variations within the product family. For example when product line architecture is not established properly, sales may agree into customer demands that are out from product line scope and thus troublesome to implement. This may result in the sale being unprofitable or worse.

Bringing variations under control is a key issue in optimizing organizational efficiency. In addition of making development ineffective, uncontrolled variations end up leeching extra resources from support functions. Implementation of fixes, updates and other maintenance tasks require significantly more expertise when working with several tool sets, platforms and thus slightly different applications. However for product maintenance, the biggest challenge is how to manage all of the applications that are specified, designed and implemented uniquely. To reduce the variations and the rippling in development, the product line approach provides product line architecture and thus more defined development process and lucid responsibilities between different stakeholders. Another incentive for having common software architecture is establishing easier governance and thus steering organization in making the right things correctly. Different organizational units and their tasks in the product line approach are illustrated in Figure 32.

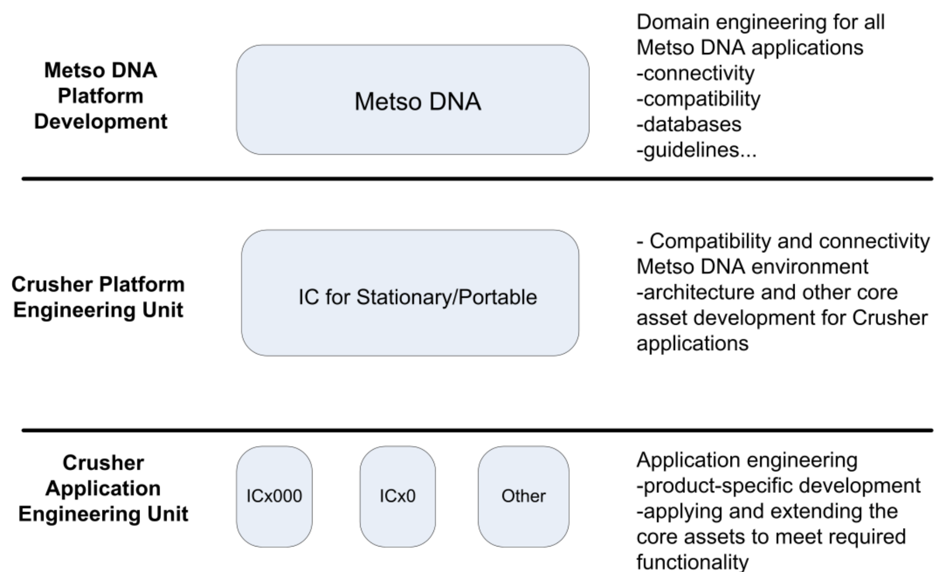

Figure 32. *The engineering unit hierarchy in IC product line.*

The organizational changes towards the product line involving domain and application units are not overwhelming because of the organization being somewhat familiar with basic product line principles. The product lines are known from a general business viewpoint, not especially from a software development viewpoint. The development of Metso DNA platform has been subjected to a single business unit and thus responsibilities of the domain engineering units are known. The application engineering unit concept is more undefined due to the fact that applications have been developed mainly in separate projects without proper guidelines and core assets. On the other hand the organization currently consists of product managers and teams each having responsibility over certain products. This gives a good starting point for establishing different Crusher application engineering units. The biggest organizational challenge is

to establish Crusher platform engineering unit responsible for crusher core asset development. The unit is also needed to enable proper communication between different stakeholders in organizational hierarchy. The communication problems have also been noted previously and tried to be met with weekly progress meetings and more active cooperation with the customer.

Introducing and training new personnel to the field of Rock crushing automation is a long term project and thus requires commitment. The long training period has both advantages and disadvantages. The amount of know-how and experiences from different projects increase the work efficiency later on. The downside is that the know-how and experiences from different projects is highly personified. This is why special attention needs to be addressed to the openness of work environment so that the amount of shared hidden knowledge is increased and distributed throughout the organization.

4.6 Process

A project-oriented process is reported [9] not to be an ideal for product line concept. Common problems include limitations in time and other resources in the creation of reusable assets. This is why a product line usually has separate business units for creating reusable core assets and derived products. Having a unit for creating reusable assets for the product family increases the overall quality of products. The use of familiar assets increases the predictability of a project and thus contributes to easier and more accurate time-estimates. These are linked to project risk management aspects. Instead of creating specific components and hoping them to be reusable in other cases, the components are planned and implemented for a more general use. This includes more thorough testing among others.

Another benefit is having the same tools for designing, implementing, testing and maintenance. Having one set instead of three, one for each platform, also significantly reduces the competences required from new recruits. Simplification of the processes creates also a common ground and more lucid responsibilities among stakeholders. Thus the commitment of developing a product into product family scope is improved. This way the products are not only similar in functionality, but also in structure and technology. Having a clear perspective and ground rules for creating an application for the product family is vital especially in global software development. However choosing only single development route increases the risks and decreases the flexibility of implementing an application. The threshold of moving from one design and implementation method to another becomes much greater. A proposal for Metso development process adapted to product line approach is illustrated in Figure 33.

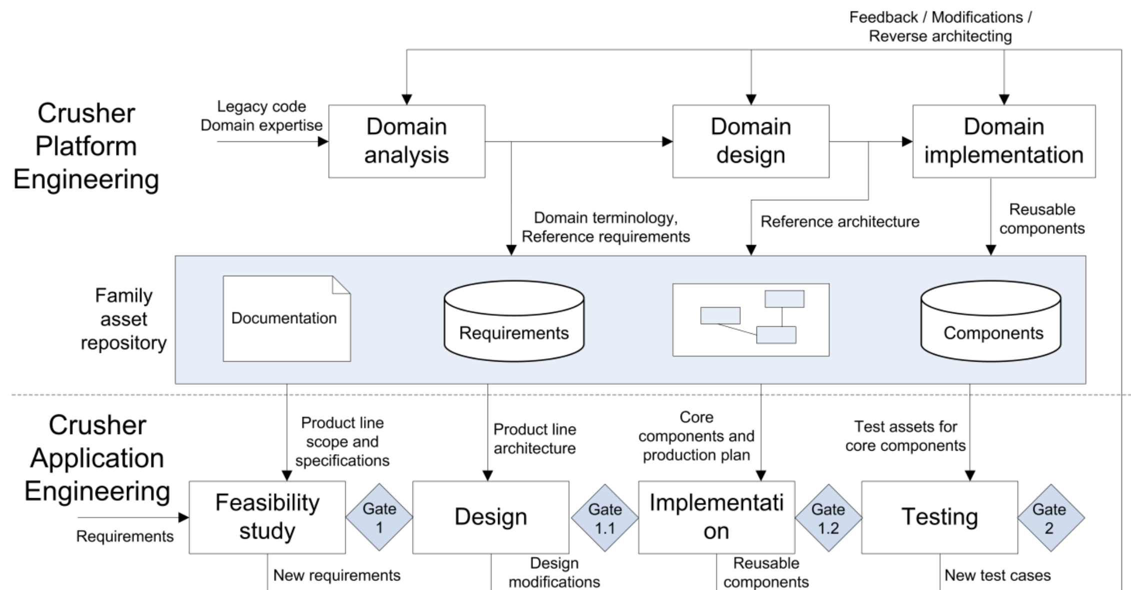

Figure 33. *Product line development applied to Metso needs.*

Separate domain and application engineering units reduce also the amount of overlapping work in documentation and testing. In a product line approach core assets are documented and tested and thus only the extensions in derived end products need product-specific tests and documentation. The amount of overlapping work also reduces as new product requirements are evaluated and added into core assets, if perceived beneficially to other products as well, instead of implementing into each application separately. The change management process must be efficient to gain maximum use from ideas emerging from the application engineering. Suitable ideas can later on be merged into family assets.

Possible downsides in the product line approach are increased costs for implementing the change, a braced development process and increased management. The increased costs are consequences of building the common core for the product family, establishing needed business units and the guidelines for the development process. Even though product line initial cost may be greater in the product line, the cost of developing several similar products from common core assets is drastically lower. The need for increased management and thus more formal approach developing software is required to reduce the rippling effect within product family.

4.7 Best practices

Implementing a product line approach to a product family is no trivial task. Thus several studies [20; 40; 41] have been done in order to find out the key success factors for implementing the change. Different guidelines for implementing the change are listed below.

- Plan the product line implementation effort to achieve immediate benefits.
- Establish clear business goals and incentives.
- Improve risk management through constant progress measuring.
- Product managers for different products using the product line architecture should synchronize their individual needs.
- Define roles, responsibilities and ways to share technology assets.
- Perform the change to product lines through incremental transitions.
- Ensure seamless communication between technology core team and implementation team.
- Use tool support for more thorough dependency analysis.
- Use architecture documentation to improve architectural integrity and consistency.
- Carefully define variation points and realization mechanisms.
- Use the described method iteratively to handle software evolution.
- Early product line applications should have robust UIs and functionality. Effective product line is based on a solid architecture.

The importance of incremental transitions in implementation of a product line concept must not be understated. Even though the product line ideology can be integrated into the organization rapidly, the transition can consume even several years [21].

5 EVALUATION

The evaluation of this thesis consists of architecture and product line analysis. The architecture and its modifiability are evaluated with Software architecture analysis method (SAAM), which was presented in section 2.4.6. The scenarios used in the architecture evaluation are based on architecture analysis done for mobile crusher control system in [39]. After architecture analysis, a Strength, Weaknesses, Opportunities and Threats (SWOT) -analysis [42] is applied to the product line approach.

5.1 Architecture assessment

This section consists of four parts. Firstly, the design decisions regarding product line architecture are listed. Secondly, the quality of the architecture is evaluated through its modifiability. This is done by testing the architecture against several scenarios. Thirdly, all scenarios are analyzed to find out how the architecture copes with different problems. Finally, a few significant scenarios are analyzed more in depth and final assessment is given.

5.1.1 Architectural design decisions

Functional partitioning involves first two steps from SAAM. The purpose of the steps is to identify relevant functionalities and their relations to design decisions from architecture. These need to be presented so that architecture can be evaluated more thoroughly. Key architectural decisions are listed below.

- A machine consists of different units.
- A unit consists of an alternative device (feeder, crusher, conveyor etc.), actuators and sensors.
- Each unit and machine has state and some level of control.
- Machine controlled by a single class.
- Same abstract class used for each unit having control, state, sensors and actuators.
- Same abstract class used for different types of actuators and sensors.
- Same abstract class used for different control modes and logs.
- Abstract units managed by a single class.
- Machine API, peer and upper level communication managed with single class.
- Authentication and user management moved closer to the API.
- Alarms handled both at the unit and machine level.

5.1.2 Modifiability

Most architectures are evaluated based on modifiability attributes. The attributes illustrate, how the architecture reacts if some part is changed, removed or replaced, or a whole new addition is implemented. A quality tree is used to manage and to evaluate the importance of different scenarios. In addition difficulties to implement the change specified in each scenario is evaluated. The quality tree used in this thesis is presented in Table 6.

Table 6. *Architecture modifiability scenarios.*

	Scenario	**Importance**	**Difficulty level**
Add	1. Adding new alarm.	H	M
	2. Adding new functionality (control).	H	L
	3. Adding new hardware (new unit type).	H	M
	4. Adding new sensor to control process.	H	M
	5. Adding new product (mobile crusher).	M	H
	6. Adding new interface to upper levels.	L	M
	7. Adding GUI.	L	H
Change	8. Changing embedded computers to PC hardware.	M	H
	9. Changing the functionality how to react to an alarm.	H	M
	10. Changing the functionality of a control.	M	M
	11. Changing the unit order in start and stop sequences.	H	L
Replace	12. Replacing peer and upper level communication methods with wireless technologies.	M	M
	13. Replacing a crusher type.	M	H
	14. Replacing an actuator type.	M	L

The scenarios are grouped into three categories: add, change and replace. The importance and difficulty level of each scenario is assessed to low (L), medium (M) or high (H). The scenarios and their importance estimates are based on work done earlier by Kaartinen and his co-workers in [39]. Few scenarios are modified to evaluate the functionalities of presented product line architecture more accurately.

5.1.3 Scenario analysis

Scenario analysis includes all scenarios described in the Table 6. Stimulus and response are described for each scenario. In addition architecture decisions and challenges regarding the scenario are listed. At the end of each evaluation, the difficulty of the scenario is evaluated based on the architecture decisions and estimated challenges. A more detailed analysis regarding scenarios is listed below in tables.

Scenario	1. Adding new alarm.		
Stimulus	Need for a new alarm.		
Response	New alarm increases the safety of the machine usage.		
Architecture Decisions	• A single class is used to handle events and alarms on machine level. • Units are responsible of notifying machine control.		
Challenges	• Alarm hierarchy between needs to be defined (plant-machine-unit)		
Importance	High	**Difficulty**	Medium

Scenario	2. Adding new functionality (control).		
Stimulus	Need for a new machine control mode.		
Response	New alarm increases the safety of the machine usage.		
Architecture Decisions	• Control modes are based on abstract class. • Machine control modes are separate from user management.		
Challenges	• None seen.		
Importance	High	**Difficulty**	Low

Scenario	3. Adding new hardware (new unit type).		
Stimulus	New unit type needed to increase functionality of a machine.		
Response	Increased machine functionality.		
Architecture Decisions	• Abstract class used to model different units. • One class to manage all units in a machine.		
Challenges	• May create changes also in sensors and actuators.		
Importance	High	**Difficulty**	Medium

Scenario	4. Adding new sensor to control process.		
Stimulus	Need for a new sensor type.		
Response	New sensor type provides more accurate measurements.		
Architecture Decisions	• Abstract class used to model different sensors. • One class to manage all sensors in a machine.		
Challenges	• May create a need for changes in unit functionality. • A sensor is subjected to a single unit.		
Importance	High	**Difficulty**	Medium

Scenario	5. Adding new product (mobile crusher).		
Stimulus	Need for a new machine type.		
Response	Mobile crushers are included into product line.		
Architecture Decisions	• Machine consists of different units. • Modularity of machine software is enforced. • Architecture designed for specific product scope (portable and stationary crushers).		
Challenges	• Tracks need to be implemented as a new unit. • Changes into machine API to enable track drive control.		
Importance	Medium	**Difficulty**	High

Scenario	6. Adding new interface to upper levels.		
Stimulus	Need for a new interface type.		
Response	New interface enables the machine to be accessed more efficiently.		
Architecture Decisions	• A single class is used to handle communications. • User management and authentication used to prevent misuse. • Interfaces in the product family need to be standardized.		
Challenges	• Adding a whole new interface is seen as a drawback.		
Importance	Low	**Difficulty**	Medium

Scenario	7. Adding GUI.		
Stimulus	Need for a local GUI.		
Response	Local UI makes troubleshooting easier.		
Architecture Decisions	• Local GUI was seen as obsolete feature. • UIs should be developed separately from machine. • UIs access machine data through proper API.		
Challenges	• Localization and graphics. • User interaction with display unit.		
Importance	Low	**Difficulty**	High

Scenario	8. Changing embedded computers to PC hardware.		
Stimulus	PC hardware is seen as more efficient way to operate.		
Response	Machine applications are moved to PC hardware with Metso DNA.		
Architecture Decisions	• Metso DNA HAL increases hardware independence.		
Challenges	• Metso DNA has limited support to different platforms.		
Importance	Medium	**Difficulty**	High

Scenario	9. Changing the functionality how to react to an alarm.		
Stimulus	An alarm is seen less or more essential than before.		
Response	New alarm increases the safety of the machine usage.		
Architecture Decisions	• Single class to handle alarm handling at machine level. • Alarms handled also in unit level.		
Challenges	• Changes possibly in unit and machine level in alarm handling. • Alarm hierarchy between needs to be defined (plant-machine-unit).		
Importance	High	**Difficulty**	Medium

Scenario	10. Changing the functionality of a control.		
Stimulus	New safety regulations require changes in operating.		
Response	Machine control is modified to meet the regulations.		
Architecture Decisions	• A machine is controlled by a single class. • An abstract class is used to model control modes.		
Challenges	• Dramatic changes, such as changes in basic machine operating rules, may considerably increase the difficulty of the scenario.		
Importance	Medium	**Difficulty**	Medium

Scenario	11. Changing the unit order in start and stop sequences.		
Stimulus	The start conditions of a unit have changed.		
Response	The start and stop sequences have been altered to meet the current configuration.		
Architecture Decisions	• A single class is used to start and stop sequences. • Units are managed by a single class that has the knowledge of ones relations to others.		
Challenges	• None seen.		
Importance	High	**Difficulty**	Low

Scenario	12. Replacing peer and upper level communication methods with wireless technologies.		
Stimulus	Wireless communications are seen more beneficial as the maturity of the technology increases.		
Response	Wireless communication methods are created as an alternative way of communicating.		
Architecture Decisions	• A single class is used to handle basic communications. • Metso DNA platform is responsible for enabling communications within a site. • The homogeneity of machine API's in product family need to be increased.		
Challenges	• The lack of communication standards increases the difficulties in machine communications. • Underground facilities are challenging for wireless applications.		
Importance	Medium	**Difficulty**	Medium

Scenario	13. Replacing a crusher type.		
Stimulus	A crusher type is seen as obsolete and needs to be replaced.		
Response	The old crusher type is replaced with a new one.		
Architecture Decisions	• Crushers are based on an abstract base class. • Units are responsible of notifying machine control.		
Challenges	• Abstract base class may be too limited. • The new crusher need to be operated differently from the previous ones.		
Importance	Medium	**Difficulty**	High

Scenario	14. Replacing an actuator type.		
Stimulus	An actuator has become obsolete.		
Response	New actuator type is used to get more precise unit control.		
Architecture Decisions	• Actuators are managed by a single class. • Actuators are based on an abstract base class.		
Challenges	• None seen.		
Importance	Medium	**Difficulty**	Low

5.1.4 Results

A more detailed analysis consists of four most relevant scenarios. The scenarios were chosen based on their importance, difficulty and estimated probability to happen. The class diagrams for the architecture can be found in Figures 33 and 34 at pages 61 and 62.

3. Adding new hardware (new unit type)

The addition of a new unit type is simple to the architecture as long as the unit consists of sensors and actuators. The functionality of the unit should also be similar to existing unit types. If not, the unit may not be suitable for the product line or at least a significant amount of work is needed especially in UnitControl class. The class is responsible for basic functionality and protective functions needed for each unit and thus probably the most vulnerable to changes. Modifications in UnitControl reflect also in UnitManager as the unit probably has more advanced capabilities compared to predecessors. Commonly new unit types are introduced together with new actuators and sensors. Thus the modifications required in implementation do not limit only to the new unit type, but also reflect to other parts of the architecture. In the end, the modifications are mostly bound to AbstractUnit class.

5. Adding new product (mobile crusher)

The product line architecture is designed to the needs of stationary and portable solutions due to their similarity in application environments and hardware solutions. Mobile crushers were intentionally left out from the scope. Mobile crushers and their architectures [39] are similar to the product line architecture in many parts. The most significant difference is that the mobile crushers need to be able move with a remote control or with use of local control panel. Also the environment of mobile crushers differs from stationary and portable sites as plant level automation is seldom used. This emphasizes the need for GUI that was left out from the product line architecture. However the GUI for the machine control can be created through proper API's so that an attached display unit is not required. Before implementing the product line, the architecture should be evaluated against the needs of mobile crushers so that in future they can be added to the family.

9. Changing the functionality how to react to an alarm

Alarms are handled in both unit and machine level. The functional changes may limit on to unit level or to machine level depending on their extensiveness. For example, some alarms may be handled automatically with adjustments without any notification to the user. At the machine level alarms usually have a more significant effect concerning the

functionality of several units and thus both actions and acknowledgement is required from a user. In any case the modifications limit in few classes, which are used to handle alarms and events, and to API used to mediate alarms to the user.

13. Replacing a crusher type

The hardware of crushers is developing constantly. Frequently a crusher type is evaluated to be obsolete and a follower is introduced. This seems similar case as adding a new hardware, but is somewhat different. The obsolete models are still in use around the world and thus require maintenance or to be updated until the end of their lifecycles. Overall the replacement of a crusher type is a rare incident and usually the basic operating principles and functionality of crushers stay the same. Therefore a more likely scenario is that the crusher type is updated or modified. For example, new technologies enable more accurate measurements and adjustments.

Risks

The architecture is designed to environments, where object-oriented or similar component-based development is possible. However this is not always possible and especially concepts like inheritance are unavailable in more primitive implementation languages. However in FBL this can be achieved through specialization, which is a close match. Another architecture related risk is its inadequacy to meet all needs of the product family. The architecture design was done by one individual with only some previous knowledge from the rock crushing environment. This is why the presented architecture needs to be evaluated and developed iteratively with a larger forum.

The product line architecture provides only a limited solution to interoperability and communication issues of the machines, because the problem is how to connect all machines to a plant level system. Providing a similar API for each machine within product family does not give significant results especially in the short term as sites are still expected to consist of previous solutions from Metso and competitors. However implementing plant level automation systems will be somewhat easier as the heterogeneity of machine API's is reduced. The effort invested into creating similar API's may not pay back in the long run as the entry of communication standards are to be expected. The opposition of the standards is likely to be greater if every solution provider has invested resources to create its own standard. This is why at least the major providers should invest in creation of a common machine API. The argument is supported also by the fact that a customer sees and buys machine, plant and upper level solutions. The customer is not interested in the solution details such as how the communication is implemented. Competition in the market area has been concentrated in the machine hardware and is expected to move more and more to the plant and other upper level solutions making investments into machine control systems less inviting.

5.2 Product line assessment

The product line approach presented is compared to the conventional development method used in Metso. The product line approach is based on common reusable assets that can be used to uniform both the family products and the product development. The current project-oriented method has not given enough support for reuse and thus has created unmanageable amount of product variations.

5.2.1 Strengths

The product line approach has four significant benefits. Firstly, product development is simplified and more uniform. In project-oriented development, products have been developed uniquely. Therefore both tasks and responsibilities must have been defined with each project again. This has created difficulties also in communication between stakeholders. In a product line approach tasks and responsibilities are pre-determined and thus more lucid responsibilities are gained.

Secondly, the use of reusable family assets reduces the work needed to be done to each product. This is because the common core can be derived from the family assets and only a small amount of tailoring is needed to create a new product. Thirdly, the use of product line architecture and reusable components also reduces the amount of overlapping work done with different products. In a project-oriented development this has been a problem as products have been isolated from others.

Finally, the decision of using a same platform for each product in the family simplifies development and maintenance tasks. Same tools can be used to create and maintain all products. This simplifies the tasks and reduces the amount of know-how required from a developer or a maintenance worker.

5.2.2 Weaknesses

A product line cannot be established instantly. The product lines require time and other resources, but most of all commitment. In Metso case the product line can be based on the latest Metso DNA based control systems and thus the architecture can be developed iteratively by abstracting software components into family assets one by one. Nevertheless the resources needed to create the family asset base and establishing a new development method within the organization is challenging. The product line approach can also be seen negatively from a developer point of view as one simply can no more implement different products with ad hoc principles as all products should also contribute to the product line.

5.2.3 Opportunities

At its best, product lines provide a workmanlike and effective product development. The product lines can also create a basis for better lifecycle management as products are no longer different from each other. This makes different support activities easier as products can be updated, replaced and modernized similarly. However one must remember that the products already around the world will still be a burden for decades.

A workmanlike approach can be achieved with product lines through using the same tools and one standardized development process throughout the organization. This contributes also to the orientation of new employees. Standardized processes are easier to learn and the productivity of the new employee is increased. The productivity is also increased with the use of core assets as the hidden knowledge from different developers is concretized into components and other solutions. In a project-oriented development these are much harder to attain.

5.2.4 Threats

Every action has a reaction. This can also be witnessed when establishing the product line approach. The common opposition towards the change is most likely to be seen when product lines are introduced. The resistance can be minimized if the product line is designed and finalized by a larger forum. Lack of commitment also slows down the establishment of product line approach. This is why the key persons in the organization must be convinced from the benefits. If not, the product line may be introduced only to a part of the organization and thus the greater benefits will be harder to achieve.

One of the most significant risks addresses the environmental concerns of a product line. Commonly product lines are suitable to stable environment that evolves with an even pace. However rapid and radical changes in environment diminish the benefits gained from the reuse. This is because characteristics of an unstable environment include lots of new requirements, which create a need for whole new solutions. Therefore a successful establishing of the product line requires suitable environment.

6 CONCLUSION

The harsh environment of rock crushing control systems is challenging both software and hardware. The hardware is required to be robust and reliable to enable the use of automation with heavy machinery. From a software viewpoint the control systems are quite simple. Hardware variations do not seem to affect the functionality of the software even though variations do exist especially in implementation methods. Requirements for each control system in a control system family are very similar even though some variations exist due to development history. However the use of and ineffective software engineering and different hardware has created unnecessary variations in the family. The unmanaged variations have reduced cost-effectiveness, reusability, maintainability and increased time-to-market in the family.

Product lines combined with proper variation management is a one way to cope with the presented problems and may even extend to other substantial benefits throughout products lifecycles. Studies [20; 43] have proven that the use product lines can increase quality and productivity in product families. However the increased productivity alone may not have an effect on time-to-market. To reduce time-to-market, the critical path in product development needs to be shorter in time. Product lines emphasize strategic reuse, which is a way to have the desired effect. Easier maintainability can be achieved through deriving all products from the same product line architecture and using common core assets. Initial costs for product line approach may seem high as the common reusable asset base needs to be built and both organizational and process modifications need to be executed. However the benefits in the long run are worth pursuing, because in time the amount of reuse significantly increases as the core assets become more advanced. Especially in Metso the amount of hidden knowledge and expertise in different areas need to be harnessed in order to achieve stability and continuity for the organization. With a proper product line approach these can be stored into the core assets and thus the productivity of a new developer is increased more rapidly and known mistakes are avoided.

The product line approach in this thesis was designed into Metso environment. The environment had certain limitations such as building the product line on Metso DNA platform and with project-oriented processes. Designed product line is far from complete and thus requires more detailed planning of different aspects. The product line can be used as a starting point for product line evolution. The product line architecture was developed through scoping available documentation and features from

implemented control systems. These revealed both up-to-date and obsolete features. The features becoming obsolete are due to the environmental changes, which are inevitable as technology progresses. Nevertheless both up-to-date and obsolete features can be used to estimate future requirements for the software and thus both are important. The design of architecture involved stripping all unnecessary features, taking in account the variations with different control systems and estimating future needs.

Organizational modifications, which were required for the product line approach, included three different hierarchy levels. At the highest level, a domain engineering unit is responsible for the development of Metso DNA platform. Crusher Platform Engineering Unit is designed to be responsible for the development of core assets for the use of Crusher Application Engineering Units. The responsibilities also include harmonizing the family to be suitable for Metso DNA. Crusher Application Engineering Units are responsible for developing actual products by customizing the derived core assets to customer-specific demands. The product line is a hybrid of ordinary product line and Metso project-oriented gate-based development processes. The tasks of domain engineering are unchanged, but the application engineering tasks are adapted to a process more familiar in Metso. This enables a more gentle approach to the product line.

Product line was evaluated with SAAM and SWOT-analysis. According to the evaluation the architecture may be feasible, but that might not be sufficient for implementation. Therefore more iterations, viewpoints and studies are required in order to define the final product line composition. For example, the chosen Metso DNA platform may not be an ideal approach from a customer viewpoint. A market study should be conducted first to find out, which of platforms and the applications are the ones most preferred by customers. It may be that none of the current platforms serve the environment requirements and thus a new one should be developed for the control systems. Nevertheless the platform and applications have the same requirements and features that were scoped in this thesis.

Another important issue is to develop needed core assets and a production plan. This way application development is able both to exploit the core assets and to create more reusable assets for the product line. The approach applying the product line should be made iteratively to minimize the risks involved. The risks of failure are greater when the production line evolution includes several products parallel. Other tasks parallel with the evolution are testing [44; 45] and optimization of the product line [46]. Proper testing contributes also to the optimization of the product line as the weak links are found out. The process must be optimized in order to gain the reported benefits from the product line approach. However the process alone will not enable the benefits and thus the focus must also be set on core assets. For example all components must be evaluated and this requires proper metrics and methods [47]. Applying the product line approach into organization culture is a long term project, which requires commitment above all.

REFERENCES

[1] Eriksson Magnus, An Approach to Software Product Line Use Case Modeling, Licentiate Thesis, Department of Computing Science, Umeå University, 2006.

[2] Reiser Mark-Oliver, Tavakoli Ramin, Weber Matthias, Unified Feature Modeling as a Basis for Managing Complex System Families, First International Workshop on Variability Modeling of Software-intensive Systems, 2007, pp. 79-86.

[3] Clements Paul, Northrop Linda, Software Product Lines: Practices and Patterns, Addison-Wesley, 2001, ISBN 0-201-70332-7.

[4] Bass Len, Clements Paul, Kazman Rick, Software Architecture in Practice Second Edition, Addison-Wesley, 2007, ISBN 0-321-15495-9.

[5] ANSI/IEEE 1471-2000, Recommended Practice for Architecture Description of Software-Intensive Systems, Institute of Electrical and Electronics Engineers, 2000.

[6] Kruchten Philippe B., The 4+1 View Model of Architecture, IEEE Software, 1995, Volume 12, Issue 6, pp. 42-50.

[7] Gamma Erich, Helm Richard, Johnson Ralph, Vlissides John, Design Patterns – Elements of Reusable Object-Oriented Software, Addison-Wesley, 2000, ISBN 0-201-63361-2.

[8] Eloranta Veli-Pekka, Koskinen Johannes, Leppänen Marko, Reijonen Ville, A Pattern Language for Distributed Machine Control Systems, Tampere University of Technology, Department of Software Systems Report 9, 2010, ISBN: 978-952-15-2319-9.

[9] Bosch Jan, Design & Use of Software Architectures: Adopting and evolving a product-line approach, Addison-Wesley, 2000, ISBN 0-201-67494-7.

[10] Anastasopoulos Michalis, Gacek Cristina, Implementing product line variability, Proceedings of SSR’01, pp. 109-117, 2001.

[11] Matinlassi Mari, Comparison of Software Product Line Architecture Design Methods: COPA, FAST, FORM, KobrA and QADA, Proceedings of the 26th International Conference on Software Engineering, 2004.

[12] Schmid Klaus, John Isabel, A customizable approach to full lifecycle variability management, Science of Computer Programming, 2004.

[13] Berg Kathrin, Bishop Judith, Muthig Dirk, Tracing software product line variability: from problem to solution space, SAICSIT ’05, 2005.

[14] Alves de Oliveira Junior Edson, Gimenes Itana M. S., Huzita Elisa Hatsue Moriya, A Variability Management Process for Software Product Lines, CASCON ’05, 2005.

[15] Svahnberg Mikael, Variability in Evolving Software Product Lines, Licentiate Thesis, Department of Software Engineering and Computer Sciences, Blekinge Institute of Technology, 2000, ISBN 91-631-0265-X.

[16] Bosch Jan, Florijn Gert, Greefhorst Danny, Kuusela Juha, Obbink J. Henk, Pohl Klaus, Variability Issues in Software Product Lines, Lecture Notes in Computer Science, 2002, pp. 303-338.

[17] Nestor Daren, O’Malley Luke, Quigley Aaron, Sikora Ernst, Thiel Steffen, Visualisation of Variability in Software Product Line Engineering, First International Workshop on Variability Modeling of Software-intensive Systems, 2007, pp. 71-78.

[18] Dhungana Deepak, Grünbacher Paul, Rabiser Rick, DecisionKing: A Flexible and Extensive Tool for Integrated Variability Modeling, First International Workshop on Variability Modeling of Software-intensive Systems, 2007, pp. 119-127.

[19] Belategi Lorea, Sagardui Goiuria, Etxeberria Leire, Variability Management in Embedded Product Line Analysis, Second International Conference on Advances in System Testing and Validation Lifecycle, 2010.

[20] Cohen Sholom, Dunn Ed, Soule Albert, Successful Product Line Development and Sustainment: A DoD Case Study, Carnegie Mellon Software Engineering Institute, 2002.

[21] Brownsword Linda, Clements Paul, A case study in successful product line development, 1996, Technical Report CMU/SEI-96-TR-016 ESC-TR-96-016.

[22] John Isabel, Lee Jaejoon, Muthig Dirk, Separation of Variability Dimension and Development Dimension, First International Workshop on Variability Modeling of Software-intensive Systems, 2007, pp. 45-49.

[23] Salicki Serge, Farcet Nicolas, Expression and Usage of the Variability in the Software Product Lines, Lecture Notes in Computer Science, 2002, pp. 173-210.

[24] Van der Linden Frank, Software Product Families in Europe: The Esaps and Café Projects, IEEE Software, vol. 19, no. 4, pp. 41-49, 2002.

[25] Deelstra Sybren, Sinnema Marco, Bosch Jan, Experiences in Software Product Families: Problems and Issues During Product Derivation, SPLC 2004, pp. 165-182.

[26] Webber Diana L., Gomaa Hassan, Modeling variability in software product lines with the variation point model, Science of Computer Programming, 2004, vol. 53, iss.3, pp. 305-331.

[27] Kim Youngbong, Moon Miyeong, Yeom Keunhyuk, An Aspect-oriented Approach for Representing Variability in Product Line Architecture, First International Workshop on Variability Modeling of Software-intensive Systems, 2007, pp. 41-42.

[28] Kazman Rick, Bass Len, Abowd Gregoyry, Webb Mike, SAAM: A Method for Analyzing the Properties of Software Architectures, Proceedings of the 16th international conference on software engineering, 1994, ISBN: 0-8186-5855-X.

[29] Reiser Mark-Oliver, Tavakoli Ramin, Weber Matthias, Unified Feature Modeling as a Basis for Managing Complex System Families, First International Workshop on Variability Modeling of Software-intensive Systems, 2007, pp. 79-86.

[30] Metso Inc. internal sources.

[31] Jaatinen Antti, Lehtinen Ville, Yli-Paunu Pekka, Lehtonen Aleksi, Petteri Kylliäinen, Flink Nina, Aro Petri, Mäkinen Mikko, EFFIMA / RESPO Report of Task 1: Concepts of a modular control system architecture, 2011, Metso Inc. and Cargotec Inc.

[32] Välimäki Antti, Pattern Language for Project Management in Global Software Development, 2011, Department of Software Systems, Tampere University of Technology, ISBN: 978-952-15-2581-0.

[33] Metso Corporation. [WWW]. Referenced at 11/2011. Available at: http://www.metso.com.

[34] Karaila Mika, Domain-specific Template-based Visual Language and Tools for Automation Industry, Tampere University of Technology Publication 938, 2010, ISBN 978-952-15-2481-3.

[35] Flink Nina, Operator Needs for Mobile Crushing Plant Control System, 2010, Master's Degree Programme in Information Technology, Tampere University of Technology.

[36] International Electrotechnical Commission. IEC 61508 standard. [WWW]. Referenced at 11/2011. Available at: http://www.iec.ch/-functionalsafety/.

[37] Kytömäki Taavi, Automation Hardware Architecture on Mobile Crusher Plant, 2007, Master's Degree Programme in Automation Technology, Tampere University of Technology. Available in Finnish.

[38] Lehtinen Ville, Software Architecture for Mobile crushing and screening machines, 2008, Master's Degree Programme in Automation Technology, Tampere University of Technology.

[39] Kaartinen Jukka, Software Architecture for mobile stone crusher product family, 2006, Master's Degree Programme in Information Technology, Tampere University of Technology.

[40] Breivold Hongyu Pei, Larsson Stig, Land Rikard, Migrating Industrial Systems towards Software Product Lines: Experiences and Observations through Case Studies, 34th Euromicro Conference on Software Engineering and Advanced Applications, 2008.

[41] Jaaksi Ari, Developing Mobile Browsers in a Product Line, IEEE Software, 2002, vol. 19, pp. 73-80.

[42] Pahl Nadine, Richter Anne, SWOT Analysis - Idea, Methology And A Practical Approach, GRIN Verlag, 2009, ISBN: 978-3-640-30303-8.

[43] Lim Wayne C., Effects of Reuse on Quality, Productivity, and Economics, IEEE Software, 1994, vol. 11, no. 5, pp. 23-30.

[44] McGregor John D., Testing a software product line, Lecture Notes in Computer Science, 2010, vol. 6153, pp. 104-140.

[45] Pohl Klaus, Metzger Andreas, Software product line testing - Exploring principles and potential solutions, Communications of the ACM, 2006.

[46] Loesch Felix, Ploedereder Erhard, Optimization of Variability in Software Product Lines, IEEE 11th International Software Product Line Conference, 2007.

[47] Faust D., Verhoef C., Software product line migration and deployment, Software: Practice and Experience, 2003, Volume 33, Issue 10, pp. 933-955.