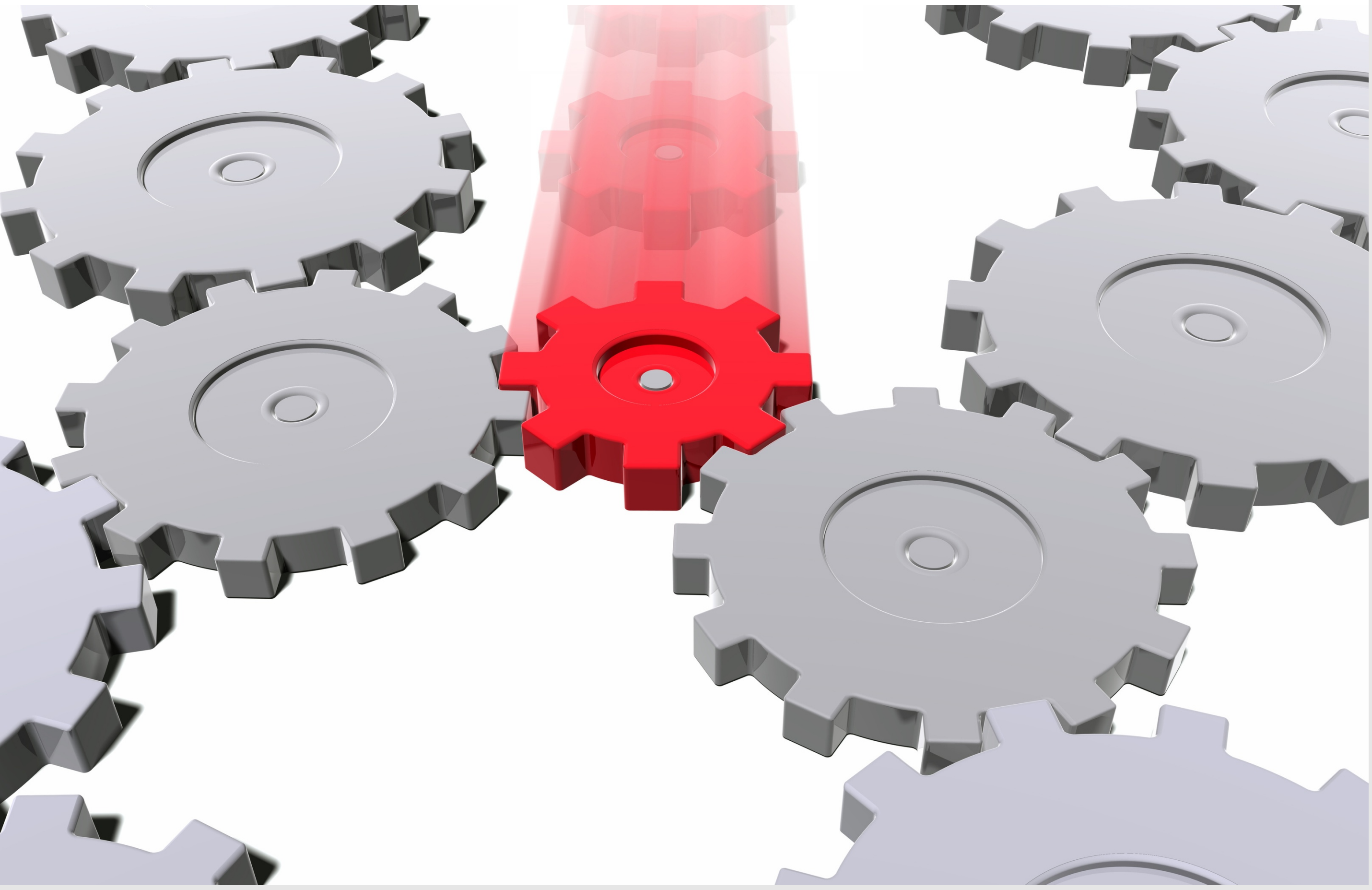
TU Clausthal



Harald Klein

Collaborative Processes of Enterprises

Supporting Global Development

SSE-Dissertation 6

Software
Systems
Engineering

Institut für Informatik
Lehrstuhl von Prof. Dr. Andreas Rausch

Collaborative Processes of Enterprises:
Supporting Global Development

D o c t o r a l   T h e s i s
(D i s s e r t a t i o n)

To be awarded the degree

Doctor rerum naturalium (Dr. rer. nat)

submitted by

Harald Klein

from Neumarkt / OPf.

Approved by the Faculty of Informatics
Clausthal University of Technology,

Date of oral examination

… … 2012

Collaborative Processes of Enterprises:

Supporting Global Development

Harald Klein

Abstract

Globalization has been one of the big trends for years in development. Due to diverse countries and cultures, it is still challenging to streamline all collaborating parties and to consolidate the existing expert knowledge in a way that "going global" brings value added to distributed projects. Therefore, global and distributed collaborations often lack effectiveness and efficiency, because workflows and processes usually do not fit together. This, in turn, leads to miscommunication and consequently to higher amount of rework and significant risk of failure in distributed development projects.

Original organizational processes of each collaborating entity should be kept as far as possible to benefit from each organization's productivity even in a collaborative environment. For this reason, this dissertation addresses the following research questions:

1. How does a structure of a process framework for globally defined development projects look like?
2. What is the added value of such a process framework for process- responsible persons in development organizations?

This work offers a structured approach of how organizations are able to collaborate from a process perspective. For this purpose, several standard collaboration scenarios are created that help development organizations setting up an integrated process environment. These integration scenarios are *Integration with equivalent processes*, *Horizontal Integration*, *Additive Vertical Integration*, *Alternative Vertical Integration*, *Merging Integration*, and *Hierarchical Integration*. These scenarios are in its majority based on practical experiences from industry and literature review. The characteristic of each integration scenario is that it comes with a connector – a so-called *Mediator* – that can be used for collaborative process definition. A mediator is a process pattern that is used to connect two or more processes resulting in a new collaborative process by simultaneously keeping the original ones as much as possible. This ensures that organizations find themselves in the new collaborative processes environment which enables and motivates them to contribute to distributed projects with full capability.

The scenario approach is the preferred solution idea due to simplicity and intuitiveness when used in practice. Thereby, the number and type of defined scenarios represent most likely business constellations from practice. For modeling the Unified Modeling Language (UML) is used due to standardized and popular character.

The original contribution of this thesis encompasses a) a set of process patterns, b) comprehensive guideline deploying these patterns, c) mathematically formalization of these process patterns graph- based notation for accurate understanding and automation purposes, and d) practice- proven benefits documentation of this approach in two case studies.

The major benefits of this process integration approach are a) Velocity of Process Setup through pre-defined process integration mediators, b) Consistency of Process Integration by having pre-checked and valid process diagrams incorporating control- and objects flow c) Applicability of Approach in every process domain, d) Adaptability of Process Integration Approach to each collaborative scenario if necessary and e) Role Concept to clearly define responsibilities of any defined task.

The approach has been applied in two case studies. Given the limitations of case studies, the results indicate that communication effort was only twice as high as in a comparable co-located development projects that were not using the process pattern approach. This is 20% below the common standard, which says that communication is 2.5 times higher in distributed than projects running on one site. In turn, the amount of rework (15%) and risk of failure (progress of burn down chart) turned out to be same like or at least comparable to projects conducted co-locally.

Danksagung des Autors

An dieser Stelle bietet sich die Gelegenheit mich bei Allen zu bedanken, die mich bei der Erstellung dieser Dissertation unterstützt haben. Diese Arbeit entstand als externer Doktorand am Lehrstuhl Prof. Dr. Andreas Rausch an der Technischen Universität Clausthal.

Zunächst bedanke ich mich bei Prof. Dr. Andreas Rausch für die Übernahme der Doktorvaterschaft wodurch die Promotion überhaupt erst zustande gekommen ist. Seine tatkräftige Unterstützung bei der Meinungsbildung während der Konzeptionsphase hat wesentlich zum Gelingen der wissenschaftlichen Arbeit beigetragen. Gern erinnere ich mich zurück an so manch' sehr früh am Morgen oder spät am Abend stattgefundene Telefonkonferenz, um die Probleme bei weltweit, verteilten Kollaborationen zu diskutieren. Prof. Rausch hat es zu jeden Zeitpunkt verstanden, mich zu den jeweils nächsten Schritten und schlussendlich zur Einreichung dieser Arbeit zu motivieren.
Bei Prof. Dr. Jürgen Münch bedanke ich mich für die prompte Bereitschaft, die Erstellung des Zweigutachtens zu übernehmen.
Das Sekretariat – in persona Annett Panterodt – hat mich in allen logistischen Belangen stets freundlich und hilfsbereit unterstützt. Herzlich Dank dafür!

Des Weiteren gilt der Dank meiner Familie, meinem Vater, meiner Mutter und meiner Schwester, die ein besonderes Verständnis und Rücksicht auf meine Doppelbelastung genommen haben und sich stets um den Stand der Arbeit besorgt gezeigt haben.

Für sehr viele wertvolle Review-Kommentare bedanke ich mich bei meinem ehemaligen Kollegen der Siemens Corporate Technology, Dr. Günter Böckle. Er hat nicht nur für den nötigen Feinschliff der Promotion gesorgt, sondern ist auch maßgeblich dafür verantwortlich, dass ein Kontakt mit Prof. Rausch erst überhaupt zustande kam.
Des Weiteren gilt mein Dank meinem Kollegen Hilmar aus dem Spring von Siemens Energy für seinen wertvollen Blick als Außenstehender auf diese Arbeit.

Inhaltlich haben mir in der Anfangsphase Dr. Edward Fischer und in der Endphase Dr. Christian Bartelt vom Lehrstuhl Prof. Rausch wertvolle Anregungen gegeben. Ohne ihre Hilfe und Unterstützung wäre diese Arbeit bestimmt nicht das geworden, was sie heute ist.
Für die liebevolle Aufnahme als „Externer" an den Lehrstuhl von Prof. Rausch gebührt mein Dank auch allen Lehrstuhl-Mitarbeitern.

Im Besonderen danke ich meiner Lebensgefährtin Marion Neubauer, die besonders während der Erstellung der Promotionsschrift so manche Stimmungsschwankung mit viel Entgegenkommen erwidert hat und durch viele fürsorgliche Worte einige für mich hoffungslose Situationen zum Besseren gewendet hat.

Contents

1 Introduction

Development processes today concern all development disciplines like software engineering and hardware engineering, which include mechanical engineering and electrical engineering [133]. Explicit definition and installation of such processes are increasingly important since the developed products[1] are also growing with respect to incorporated features and non-functional requirements, e.g. performance, safety, security, etc., which results in much higher complexity.

Besides hardware engineering as the major development domain, software engineering emerged as an essential discipline within product development. Therefore, the software portion of products increased considerably in the last decades. Following a statement of Boehm [22], the ability of any organization to survive the rough market conditions will depend more and more on software in the future.

Software Engineering is a very young discipline compared to, e.g., Hardware Engineering. Since the expression was coined by Fritz Bauer at the conference in Garmisch-Partenkirchen in the late sixties, Software Engineering emerged intensively regarding importance in the last forty years. Significant effort and advances have been made in Software Engineering especially regarding better manageability, higher predictability, and, in general, the use of more systematic development approaches. This has been achieved by the improvement of software lifecycle models, sophisticated architectures, more effective planning and controlling methods, and better tooling [142].

A significant advantage of software is that it provides competitive differentiation and rapid adaptability to competitive change. This means that software facilitates rapid tailoring of products and services to different market sectors, which makes feature implementation through software solutions very attractive. For instance, the product creation of a mobile phone can be done by making use of software platforms. Once such a platform is defined and implemented, adaptation towards different mobile phones is relatively easy. Additionally, several mobile phone models might also be differentiated by just deactivating certain software features.

Another example comes from the automobile industry. The installed engines for cars are usually configured using software and appropriate algorithms, which allow the automobile manufacturer to simply change the engine characteristics, e.g., towards more power, without any more development effort. Especially, in this business software gains more and more importance. In 1990, only 16% of an automobile’s total value accounted to software; in 2001, this number has grown to 25%; today, we are at 40%. Premium class automobile vehicles contain up to one gigabyte on-board software [73].

Further industries are also concerned by growing software complexity, e.g., energy (e.g. instrumentation and controls (I&C) for power plant automation), or in the healthcare sector (e.g. computed tomography, magnetic resonance).

However, software engineering is also one of the most challenging disciplines during product development. Generally, organizations and their development processes are pressurized to react in a highly flexible market with innovative products dynamically and quickly on

[1] All statements made in this dissertation are valid for products and systems likewise; for defintion of terms please refer to chapter 1.4

steadily changing market requirements. Simultaneously, development departments have to meet quality goals, cost restrictions, and they have to fulfill country-specific, local standards [42]. These market challenges are the main causes that make (software) products more and more complex over time.

Growing complexity and challenges require expert knowledge for respective product features in certain domains, e.g., database technology, search algorithms, security etc.

Furthermore, highly qualified employees are indispensable for development organizations today. Especially the concentration on their core competencies is necessary to stay competitive in their respective business. This is due to the fact that, on one hand, rivalry in business capabilities increases steadily. On the other hand, it is too expensive for companies to build up every required competence in every existing development site. Additionally, development organizations benefit from having local business partners in those countries where they want to sell products.

Since specialized experts are not always available locally when they are needed, organizations are forced to expand their staff sourcing on a worldwide basis. Global sourcing is easily realizable these days within many engineering disciplines (e.g. software, hardware, mechanic etc.) due to the fact that development is majorly conducted computer-based, and engineers can contribute to products via Internet from anywhere all over the world. Thereby, any design software is installed on company internal servers that are accessible from wherever it is necessary.

Skilled employees coming from all over the world must be given a structured way and guidance of how product development and business is done in a respective company. Therefore, globally defined processes get increasingly important to integrate new or temporary employees into business environment properly. This approach expands local development to distributed development that comes along with globalization in order to get best talents available for specific tasks.

However, the major problem that development companies have is the lack of collaborative processes for distributed development. This simply results from the fact that processes in general are often seen as an unnecessary burden, which can easily be avoided by just omitting it.

A collaborative process connects two or more organization specific and local processes in order to make use of specific organizational capabilities for project challenges. Thereby, the original processes are not changed. A collaborative process works as a facilitator for organizations' success. This fact is well known and accepted among development organization so far, however, there are only a few collaborative processes or mechanism defined that describe how to connect different processes globally from various organizations. Organizations often argue that they never ever have needed explicitly defined processes for doing successful business. Furthermore, the organizations have seldom appropriate structures and capabilities, i.e. roles, responsibilities, and especially methods to address this issue. Well-defined processes for collaborative development will significantly reduce friction between participants, avoid double work, and reduce conflicts at the developed constituents and thus reduce development cost and time.

1.1 Statement of the problem

Global development environments where two or more separate organizational units work together to develop a product require collaborative processes that allow for integration of respective organizational processes. Process integration needs to be done in a way that original processes are kept as defined in the respective organization. This is crucial since development organizations can deploy specific skills and capabilities best by using a familiar process environment.

Therefore, collaborating organizations can only effectively work together if they are allowed to follow their specific organizational, institutionalized processes. If processes are changed rapidly and too often companies get inefficient in their day-to-day work. This is due to existing organizational culture, which processes are directly connected to. Cultural change would need time and wide acceptance throughout the entire workforce.

For this purpose, organizations must have an approach that connects respective organizational processes and explicitly indicates process issues, e.g., process inconsistencies like missing/wrong artifacts to be handed over etc.

Unfortunately, there are only a few – if any at all – approaches defined that are capable to address global and distributed development issues of how to define and set up processes for global collaborations. This work focuses on this problem and comes up with a methodology to define collaborative processes with two or more organizations that are able to be executed in a distributed project on a global basis.

1.2 Objectives and Research Questions

This dissertation pursues two goals.

1. The core of this work shall define and describe a process structure that allows two or more development organizations to collaborate in a globally dispersed project. This resulting structured methodology shall be easy to use and is intended to be handled intuitively.
2. Furthermore, the value add of the process framework will be proven in an illustrative case study that is based on consideration of real development projects and experiences.

Therefore, the research questions can be formulated as follows:

1. How does a structure of a process framework for globally and distributed defined development projects look like?
2. What is the added value of such a process framework for process- responsible persons in development organizations?

1.3 Scope of the Dissertation

Global development is a potentially huge field of different activities. Besides processes, various people and cultures, these activities include also collaborative tools that are crucial for global and distributed development, e.g., configuration management, design, and simulation etc. This work focuses primarily on the definition of processes that are used in a global development project environment. Thereby, a mechanism shall be developed that connects two or more processes in a way that a new collaborative process is created. This new collaborative process is then executed to run a globally defined project.

In order to illustrate the added value and the validity of the process framework, a case study will be conducted and also documented in this work.

In terms of the considered case study, the dissertation focuses mainly on software development. However, the process framework is not limited to software development process, but can also be used to connect processes in any environment, e.g., hardware development.

1.4 Definitions and Terms

Artifact:	Any type of documented process output, e.g., descriptions, plans, code
Cross Enterprise Engineering (CEE)[2]:	Collaboration beyond any organizational borders
Collaboration:	Working together without being located in one place, e.g., room, building, or site
Efficiency:	"Do things right", i.e., to achieve a defined goal by using minimal resources (inputs)
Effectiveness:	"Do the right things", i.e., to do those things that lead to the desired goal
Feature:	Performance characteristic or attribute of a (software) product as a result from development
Globalization:	This term is used in the sense of "global collaboration"
Mediator:	A mediator is a pattern that is used to connect two or more processes resulting in a new collaborative process (See chapter 3.3.1)
Process Tailoring[3]:	Making, altering, or adapting a process description for a particular end.
Process Owner:	A role that is fully responsible for a respective process in terms of execution, result generation, and maintenance
Process Engineer:	A role that designs a process according the constraints and requirements of the process owner
Product:	A "thing produced by labor or effort" or the "result of an act or a process"
Scenario:	A scenario is an environment that applies and illustrates the use of a mediator by using exemplary processes
System[4]:	- A set of interacting or interdependent components forming an integrated whole

[2] refer to [42]
[3] refer to [133]

	- A set of elements (often called 'components' instead) and relationships, which are different from relationships of the set or its elements to other elements or sets
Truck Factor[5]:	Number of people the project could lose before it gets into serious trouble
XP:	e**X**treme **P**rogramming: An agile software development methodology intended to better react to changing customer requirements by advocating frequent software releases

1.5 Structure of this Dissertation

The structure of this dissertation is as shown in Figure 1. Thereby, the left side shows the chapters on theory; whereas, the right side illustrates practical chapters that have been “derived” from the respective theory. Chapters depicted in the middle are the general ones.

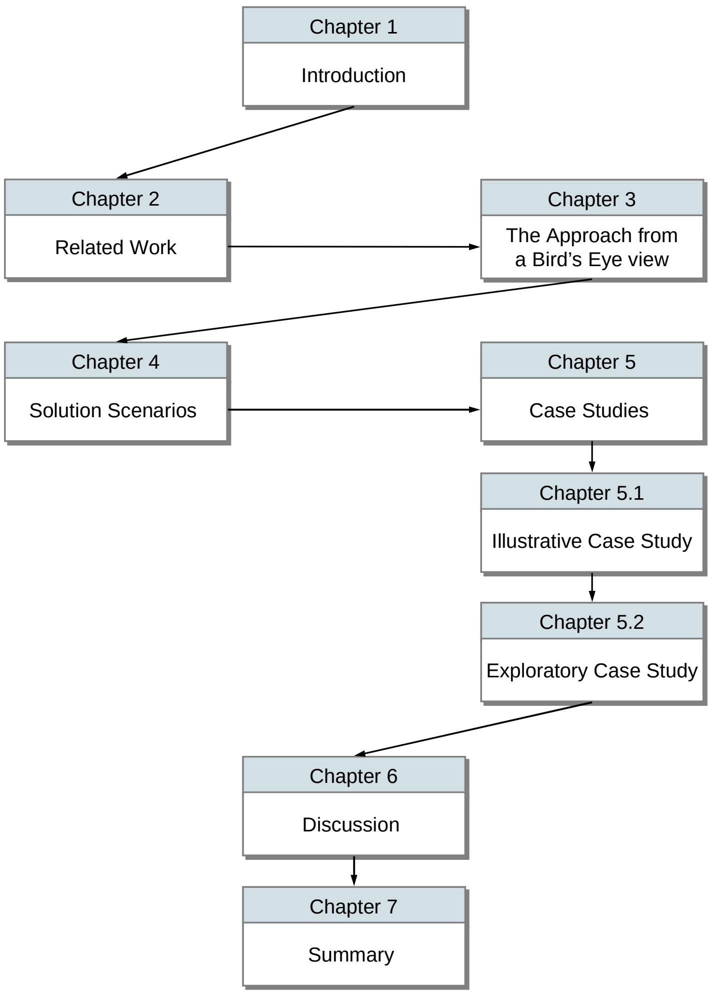


Figure 1: Structure of Dissertation

4 http://en.wikipedia.org/wiki/System
5 http://www.agileadvice.com/archives/2005/05/truck_factor.html

1.6 Assumptions

The solution concept of this dissertation assumes that those organizations, which participate in collaborations, have (development) processes already in place prior to the definition of any collaborative process. The respective organization automatically follows those defined processes in their day- to- day work. Processes are an inherent part of their business. Thereby, a collection of sub-process descriptions is also accepted as a kind of organizational process.

1.7 Limitations and delimitations

1.7.1 Limitations

Due to the huge field of potential study possibilities, this work is limited.

The dissertation will not focus on tools, meaning software applications or databases to support a global project set-up. These software applications usually support the product lifecycle management or specific sub-processes like requirement engineering, e.g., ‘Dynamic Object-Oriented Requirements System’ (DOORS), ‘Requisite Pro’ or testing, e.g., Test Director. Tools are supportive key elements for successful distributed collaborations, e.g., for configuration management or requirement engineering and should not be neglected during collaboration set-up.

Mechanisms on how to implement quality assurance are not considered in this work.

Tools environment are also used in project environments called Virtual Reality (VR). These tool environments support virtual product development and reduce the complexity of a project [42]. Although this is strongly related to this dissertation, it will not be considered in detail.

Applicability of this approach is limited by the different processes used in various process domains, i.e., it will be difficult for, e.g., a software design process to get input from configuration management process.

Development collaborations typically use a certain type of development model. Ten years ago, the ‘Waterfall Model’ has been the most commonly applied model. Nowadays, modern development uses ‘Agile Development’ or at least iterative development, e.g., using the Rational Unified Process (RUP). This dissertation will not discuss such models. Also benchmarking models like the Capability Maturity Model Integration will not be subject of this work.

Cultural aspects are not considered in this work, although it is common sense that it is very important to be considered during the set-up of global distributed development projects. For a short abstract of cultural aspects, please refer to section 2.7.

1.7.2 Delimitations

This dissertation mainly considers software development processes, but is not limited to them. The used approach is also applicable for any other types of processes, e.g., hardware development, marketing, human resources. The nature of processes from an atomic point of view makes that possible, i.e., every process step – no matter in which environment defined and used – consists of an activity, in-/outputs, roles, methods etc. Since the approach of this work deals with that level of granularity, it is predestined to be used in various domains.

This dissertation does provide mechanism to handle recursive processes, since not all processes are created and defined prior to execution.

2 Related Work

The purpose of this chapter is to introduce and explain related terminology and work of other authors related to this dissertation. Additionally, this section motivates processes, which is supported by data based examples in terms of necessity and benefits. Furthermore, typically process issues in (globally distributed) development organization are addressed. Additionally, several statements are proven by data examples.

2.1 Organizational Business Processes

2.1.1 Fundamental Terminology

The term “process” is very manifold, and the meaning depends on the environment used. In a business environment, several process terms or combinations of that are established. But what is a process? Before collaborative processes are discussed, this work gives some basic definitions.

2.1.1.1 Process

Basically, a process is a transformation that consists of several activities. This transformation gets a defined input and creates a defined output. Input factors could be machinery materials, equipment, manpower, raw materials, energy, or information. The generated output of the process encompasses products and services [122]. The **I**nternational **S**tandard **O**rganization (ISO) defines “process” as a set of interdependent activities, which converts inputs to results [40]. A process is determined by various parameters, such as scope, content, or structure.

2.1.1.2 Software Processes

A software (development) process is a structured approach to create a software product. Software is special insofar that it is immaterial and cannot be touched. In contrast to traditional hardware development, software processes use their own proprietary methods and tools for creating the product. These methods/tools are incorporated in a structure, which typically consists of several phases that include but are not limited to:

- Requirement Engineering
- Architecture and Design
- Implementation
- Verification and Validation (Test)
- Maintenance

In order to improve software process performance, a personal software process (PSP^{SM}) in general has been developed by Humphrey. Having requirement specification as input PSP^{SM} defines several phases, which provide data-oriented, disciplined methods based on improved planning, commitment, and quality [66].

2.1.1.3 Business Processes

‘Business Process’ is an umbrella term, which defines all processes that occur in any business environment, e.g., marketing, sales, controlling, quality assurance, supplier management, as well as development or human resources.

Using a formal definition, a business process consists of a set of activities that are performed coordinatively in an organizational and technical environment. These activities typically realize the business goal jointly. Each business process is enacted by a single organization, but it may also interact with business processes performed by other organizations [143].

The challenge for any business process is the way of its definition in order to generate or produce valuable output for either internal or external recipients/customers. These outputs might satisfy, for instance, a requirement specification from a customer or an internal quality review checklist from the quality management process.

Figure 2 exemplary illustrates the terms ‘process’ and ‘business process’. However, this is by far not a complete (sub-) process definition, but rather a general process activity model that can be used to define executable processes.

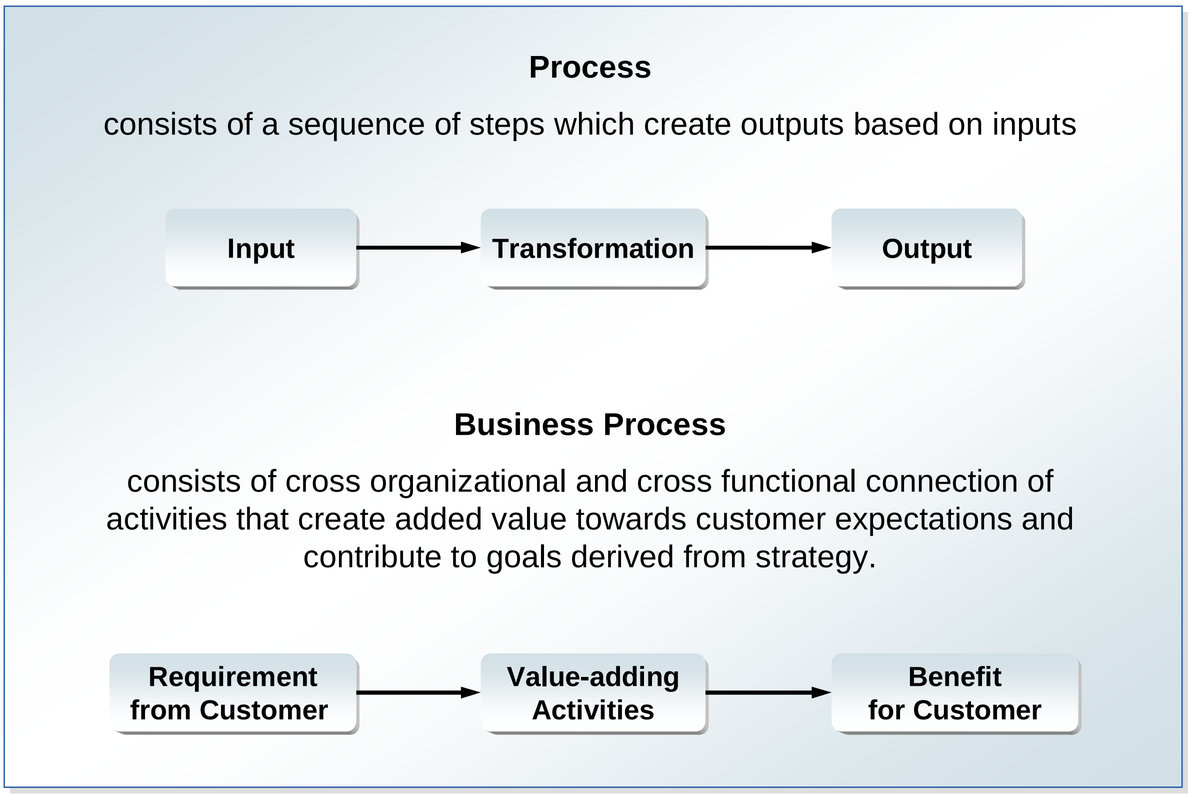

Process
consists of a sequence of steps which create outputs based on inputs
Input
Transformation
Output
Business Process
consists of cross organizational and cross functional connection of activities that create added value towards customer expectations and contribute to goals derived from strategy.
Requirement from Customer
Value-adding Activities
Benefit for Customer

Figure 2: Definition: Process and Business Process

Many definitions of business processes have emerged over time. For better understanding, some of those, which support this work best, are described in the following.

Davenport defines in [40] business processes as a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on *how* work is done within an organization, whereas, the product’s focus on *what*. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. Processes are the structure by which an organization does what is necessary to produce value for its customers.

Hammer and Champy see business processes as a collection of activities that takes one or more kinds of input and creates an output that is of value to the customer [59].

A definition that considers cross functionality in organizations is given in [114]: “A business process is a series of steps designed to produce a product or service. Most processes (...) are cross-functional, spanning the ‘white space’ between the boxes on the organization chart. Some processes result in a product or service that is received by an organization's external customer. We call these primary processes. Other processes produce products that are invisible to the external customer, but essential to the effective management of the business. We call these support processes. “

In [137] Scheer and Zimmermann describe a business process as an activity, which is important for the added value of the company from project kick-off until termination.

For further definitions please consider [100], [133], [53], [77].

The value adding activities need to be connected beyond functional and organizational borders. These coordination activities generate significant cost and issues in organizations. Therefore, the discipline Business Process Management (BPM) has been defined to attend to these issues [122].

2.1.2 Business Process Management

Business Process Management (BPM) has reached significant importance in business and development organizations in the last years. Referring to a survey conducted around IT decision maker 67% of the interviewed organizations are *strongly* or *very strongly* engaged in doing BPM [69]. Most of them – about 95% - consider BPM to be *important* or even *very important* [49]. This is also reflected in the trend that development organizations are strongly interested to initially define or re-engineer their defined process in order to decrease cost and reduce the overall lead time. This gets more important as (software) product and the appropriate development effort get more and more complex.

This shows that business and development processes are key instruments for the understanding and the successful execution of a business. Consequently, processes visualize their own business and make it easier to control and improve the business, especially in very complex environments.

A BPM concept consists of several aspects, which need to be fulfilled to successfully do BPM. Figure 3 shows an overview of such a concept. The major framework of BPM is the (1) organizational strategy mainly driven by (2) customer. All activities are oriented towards these two aspects.

Figure 4 gives some more explanation and definition of what the single duties of BPM are and what they are responsible for [122].

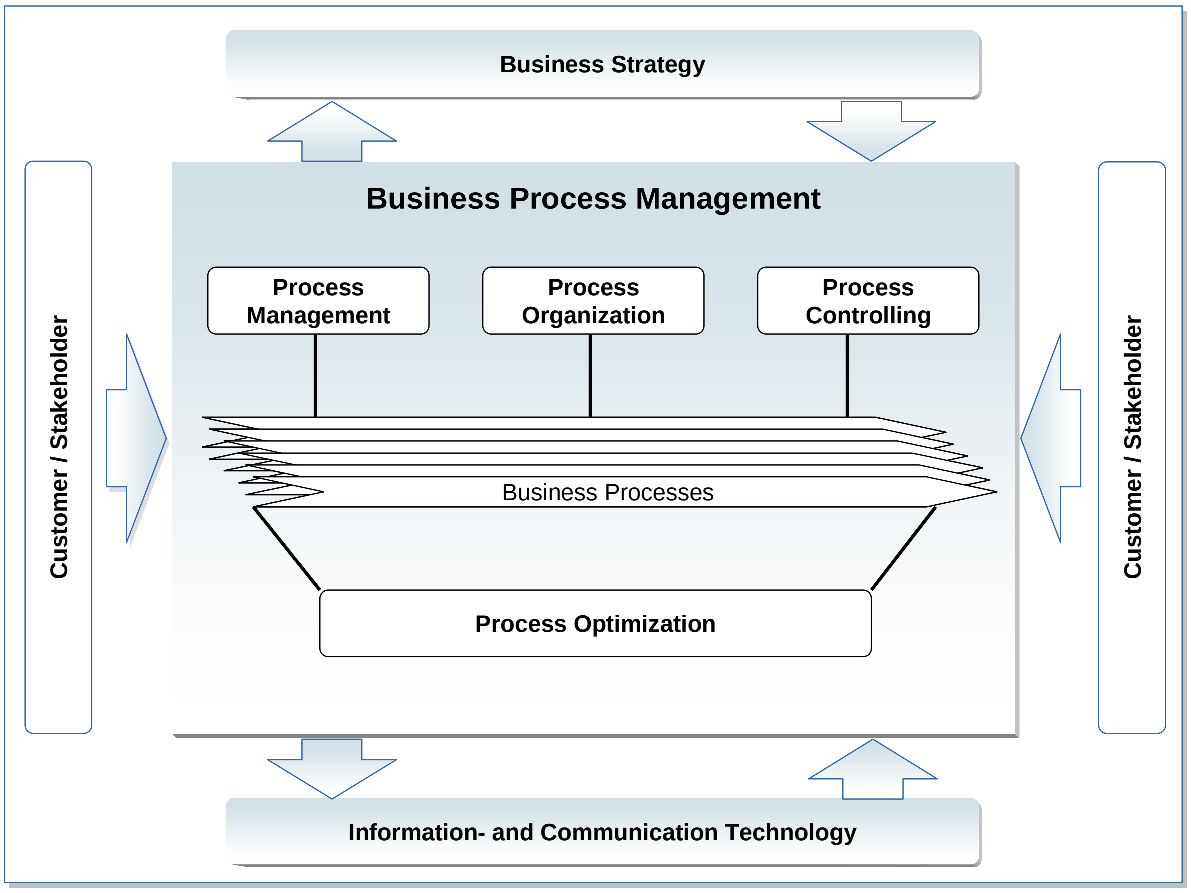


Figure 3: Integrated Business Process Management [122]

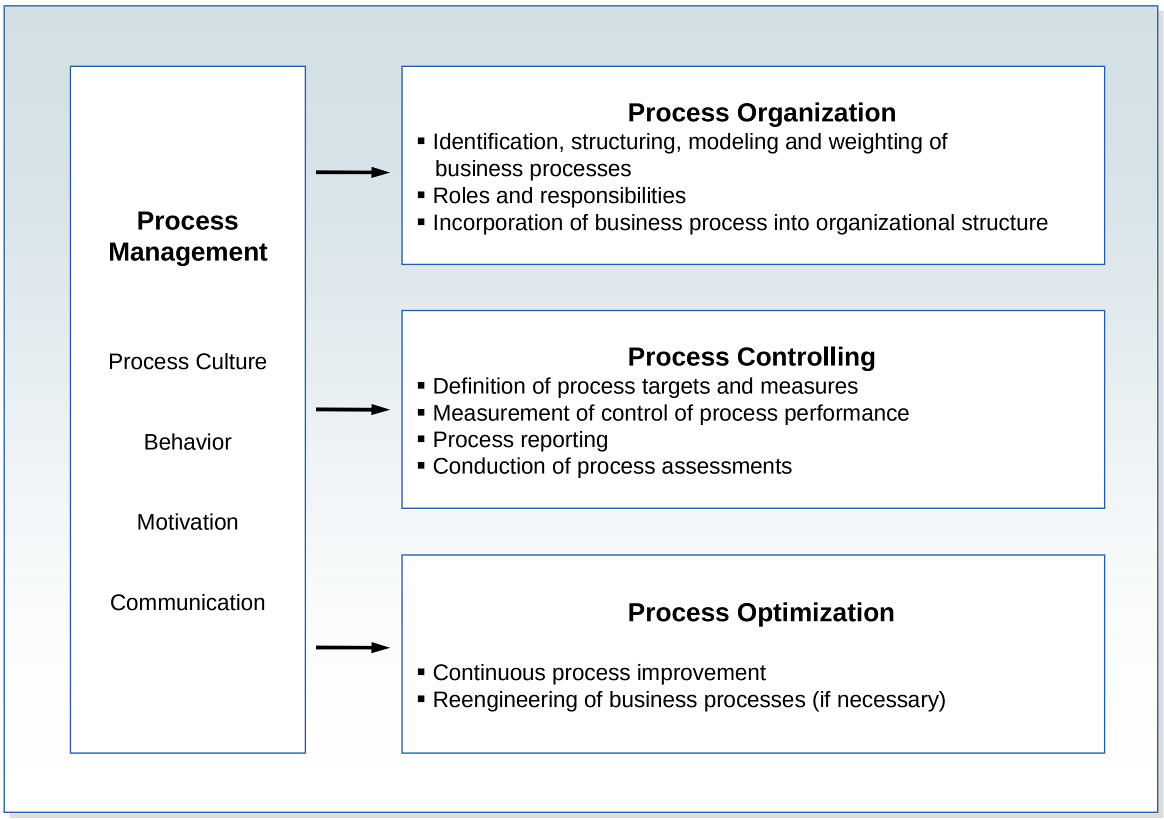

Process Management
Process Culture
Behavior
Motivation
Communication
Process Organization
▪ Identification, structuring, modeling and weighting of business processes
▪ Roles and responsibilities
▪ Incorporation of business process into organizational structure
Process Controlling
▪ Definition of process targets and measures
▪ Measurement of control of process performance
▪ Process reporting
▪ Conduction of process assessments
Process Optimization
▪ Continuous process improvement
▪ Reengineering of business processes (if necessary)

Figure 4: Scope of Duties of integrated Business Process Management

2.1.3 General Process Problem of Organizations

Although organizations are aware of many potentials resulting from process management, many companies suffer from process deficiencies. Schmelzer [122] describes that major problems in organizations originate from two aspects: **Effectiveness** and **Efficiency**.

The term 'Effectiveness' can be paraphrased with "do the right things" and means, e.g., defining the right success factors, develop appropriate core competencies, penetrate the right markets, and develop the right products for those markets. However, many organizations suffer from deficiencies concerning these effectiveness factors. Examples are:

- missing persuasive vision
- unclear strategic goals
- unclear market goals
- insufficient knowledge about (potential) success factors
- inadequate knowledge about customer problems
- unclear product- and process goals

These deficiencies typically result in unsatisfied employees and especially unsatisfied customers, which are even more critical [122].

In contrast to effectiveness, the term 'Efficiency' can be paraphrased with "do things right", which is more economic- oriented than 'Effectiveness'. Figure 5 shows some problems that result from a low efficiency in an organization.

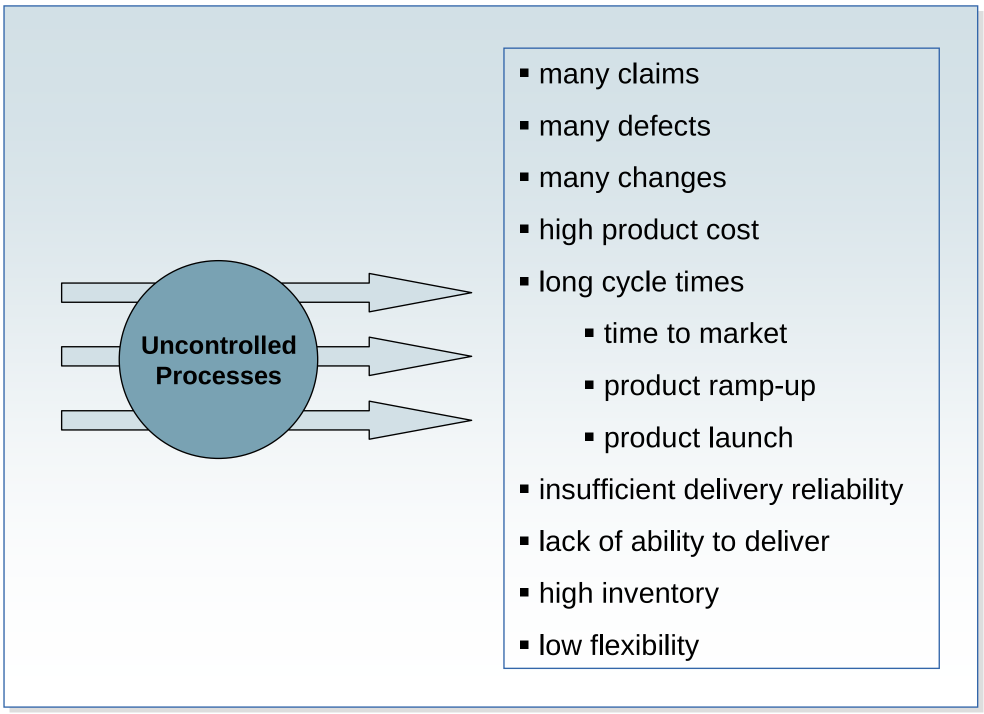


Figure 5: Problems resulting from non-controlled processes

This lack of efficiency is mainly driven by the problem that processes are not designed efficiently and not controlled, although they could be controllable. In several cases, processes are executed that generate almost no additional business value. Too many interfaces are defined that need tremendously more coordination than are actually necessary, which increases cost and lowers profitability. Therefore, cost, time, and quality are the major parameters that drive efficiency in an organization [122].

The empirical studies mentioned in the previous section also give some evidence for process deficiencies in organizations. Fink [43] states that process-orientation is a major driver to increase efficiency and effectiveness. However, only 56% of the considered organizations have defined a process-responsible person; 36% have the process views with appropriate Key Process Indicators (KPI) incorporated into the controlling. Only 22% of the interviewed organizations stated to have transparent process lead times and process costs [43].

Referring to [49] only 9% of the interview organizations have a process- responsible person with business responsibility (budget) and only 7% use a reference model comprehensively.

The deficiencies in BPM should be identified and addressed by the integrated BPM approach shown in Figure 3 and Figure 4.

2.1.4 Benefits of Business Process Management

The responsibility of BPM is to optimize the organizational process and, thereby, improve effectiveness and efficiency of processes that result in sustainable increase of the companies' value. Organizational processes and process management respectively in organizations are major drivers of business' success in organizations and gain essential benefits. Schmelzer [122] stated that the most important benefits of process management are:

- Higher transparency of the contribution of process' value added to the overall value added gets better measureable
- Better process efficiency due to reductions of interfaces and resulting material and information flows

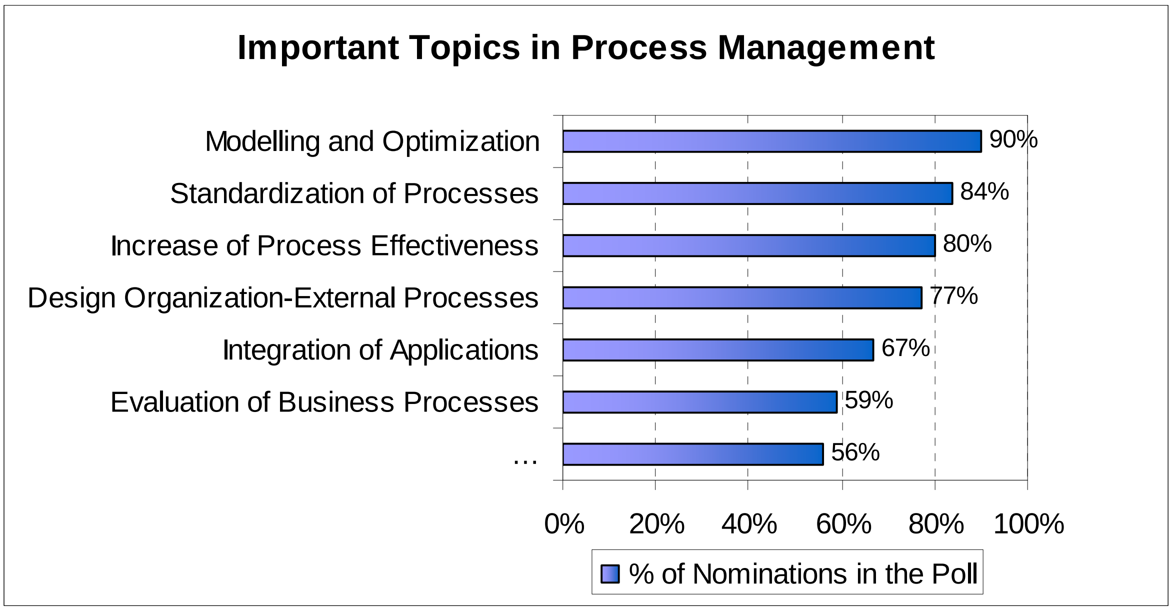


Figure 6: Important Topics in Process Management [69]

This is the summarized result from several empirical studies conducted by Fink [43], Bach and Biemann [7], Gadatsch [49], and IDS [69]. The latter brings up two essential aspects in terms of process management.

First, organizations recognize that process management contains important topics to be addressed in order to be successful in business. Figure 6 illustrates that organizations are aware of the importance of processes and process management. In this case, organizations might tend to spend significant effort for process definition and improvement. They apparently think that there is much potential for improvement to be realized along the value chain, especially towards productivity, cost reduction, and increase in effectiveness. Furthermore, organizations want to follow a more integrated process approach, which means that processes of customers, suppliers, and business partners also need to be included and integrated into the own process landscape.

Second, organizations realize significant benefits from processes defined and process management, which is shown in Figure 7. Besides the "Faster Processing of Orders" (63%) cost reduction (61%) and higher product quality (52%) are major benefits from organizations' perspective.

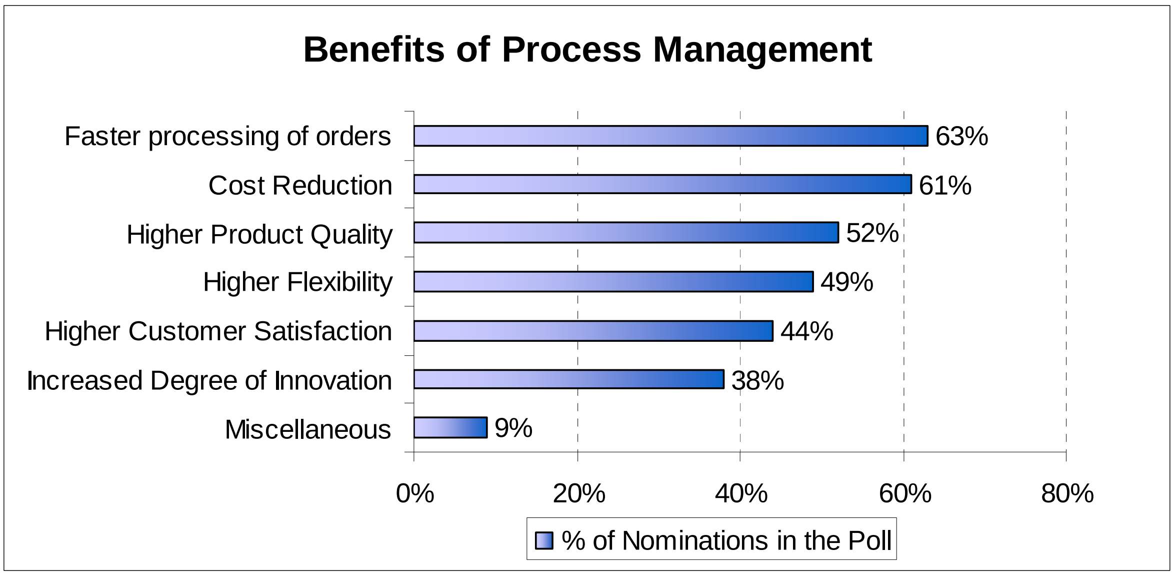


Figure 7: Benefits of Process Management [69]

But how does BPM contribute to development?

Successful companies typically measure their processes performance with Key Process Indicators (KPI). Based on these measurements, improvement potentials are derived and implemented in appropriated improvement projects that typically should be conducted with the same importance like other projects in an organization.

Business processes are the foundation for development processes, because BPM generates data and information that are valuable input for development processes. This is briefly explained with two examples.

Example: **Generating Product Requirements**

Requirement Engineering is a substantial sub-process within development. In order to source this process marketing department gets and evaluates data from the relevant market in such a way that requirements for future products can be derived. This information is taken as input for the development department.

Example: **Cost Reduction**

Management typically drives development cost reduction (Figure 3), i.e., the decision comes from business strategy. This decision is a subject for process optimization and executed by process management and controlling.

BPM allows for having the right data and information available when it is needed. If BPM is oriented towards product development, the concept is called Product Lifecycle Management (PLM). Using this approach makes development controllable and manageable [18]. The PLM concept is discussed in the following section.

2.2 Product Lifecycle Management (PLM)

All processes that are considered for problem solution purpose are covered by Product Lifecycle Management (PLM) throughout this entire thesis. Therefore, PLM is introduced and classified within business process environment.

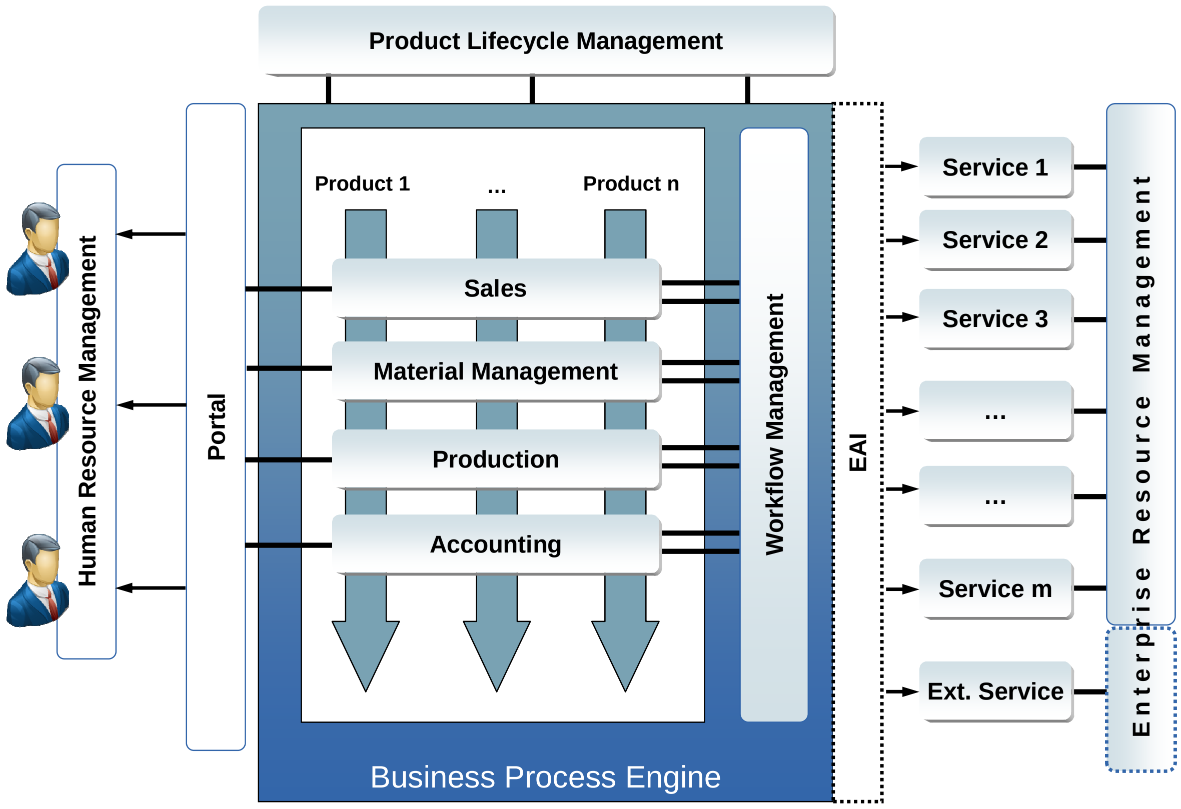


Figure 8: Product Lifecycle Management and Business Process Platforms

The *Product Lifecycle* in general spans from the first product idea, development and production, sales and maintenance, until phase- out and recycling of the product. This process model mechanism is illustrated using a matrix structure (Figure 8) involving several departments, which need different types of information [18]. These types are, e.g., Sales, Production, or Accounting on one hand (horizontal arrows); on the other hand, products 1...n to be developed (vertical arrows) take use of those process domains. Workflow management, which coordinates interfaces between products and process domains, and Enterprise-Application Integration (EAI) are controlled by the Business Process Engine – as the core of

PLM. Sourcing is done by Human Resources department, which make sure that appropriate people are available for activities to be done.

PLM is consequently a comprehensive, systematic, and controlled concept for managing and developing products and their related information of the whole *Product Lifecycle* [123]. Following the definition of Sääksvuori the concepts' intention is to "control and steer the process of creating, handling, distributing, and recording product- relevant information."[116]. This relevant information is basically a compilation of business rules, methods, processes, guidelines, and instructions. The concept provides an overview of all business relevant processes, their interrelationships, and furthermore gives instructions how to comprehensively fulfill the requirements of the desired product for a successful contribution to the desired market. Although it is a significant success factor to the PLM under IT control, PLM does not refer to any individual software application or method, but it is a wide totality [116].

A product lifecycle is highly individualistic and typically tailor- made to each organization. This means that every product lifecycle encompasses similar process steps or phases, but the concrete process instance differs from each organization. The creation and respective implementation of a PLM framework can fail if insufficient structures of the framework are in place, which makes processes hardly manageable and controllable. The following section shows the PLM design and argues the benefits of a PLM framework [133].

2.2.1 PLM Design

A typical PLM landscape is divided in three different areas that contain different types of processes. Figure 9 depicts such a framework and concentrates on functions, not on products as shown in Figure 8. Three main process types are distinguished:

- Core processes that are directly interrelated with products and its development
- Management processes, which control the core processes
- Support processes, that assist the core processes during execution

Assignment of process types depends heavily on value added to the entire business. Core processes encompass value add processes, which have an end-to-end character. This means that those processes range from stakeholders, which give input for generating the first product idea, to those customers the final product is sold to.

Management process control those core processes towards the defined strategy, e.g., Strategic Planning, if other markets with other product requirements need to be penetrated.

The third type is support processes, which supplement core processes during execution. Human Resources (HR) department, for instance, supports development departments with identification of demands and hiring of additional employees with special knowledge for the development of future products [18].

The processes per process type in Figure 9 are not in an appropriate order required by the product lifecycle. This would not make sense on this level, because interaction paths are only reasonable if they are associated with concrete activities or action and corresponding artifacts or output respectively. Moreover, a comprehensive overview of all defined (sub-) processes should have been given.

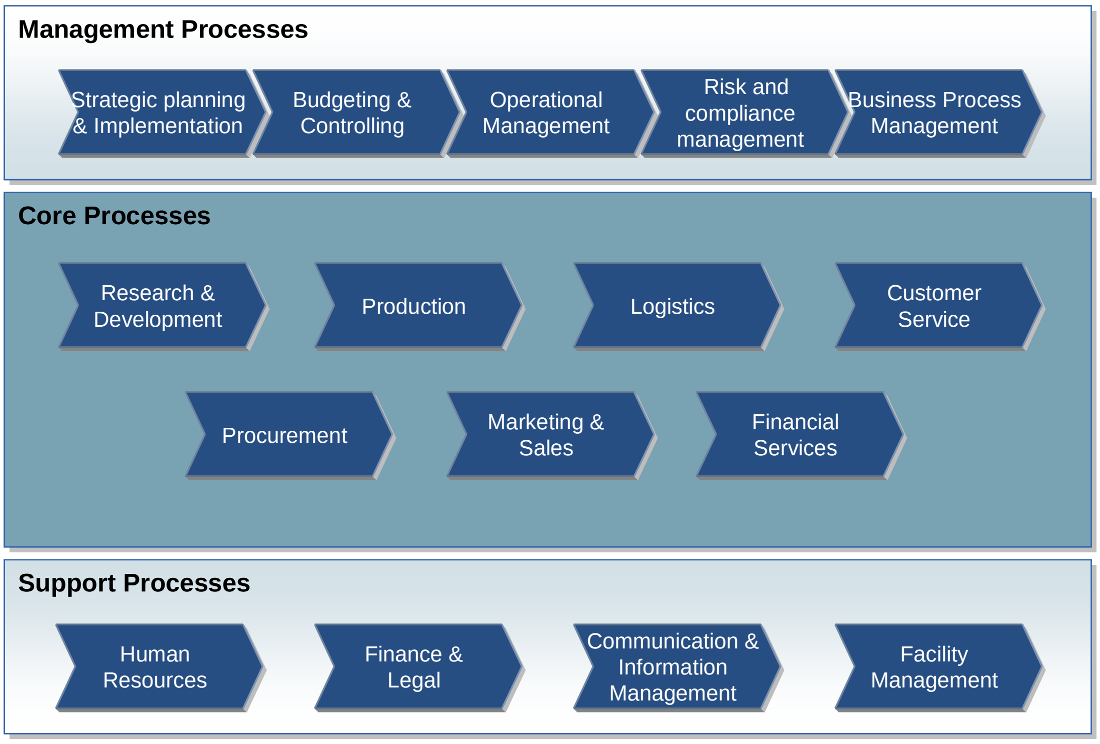


Figure 9: Product Lifecycle Management (PLM) Framework

Sääksvuori and Immonen [116] illustrate their product lifecycle in Figure 10. This graphic does also not consider any order of processes and activities. Moreover, it illustrates the wide totality of varying functions to support all process around the product lifecycle. Additionally, indicated by blues arrows, it is shown that the PLM is not a closed and isolated system, but it interacts with suppliers, service partners, and especially with customers.

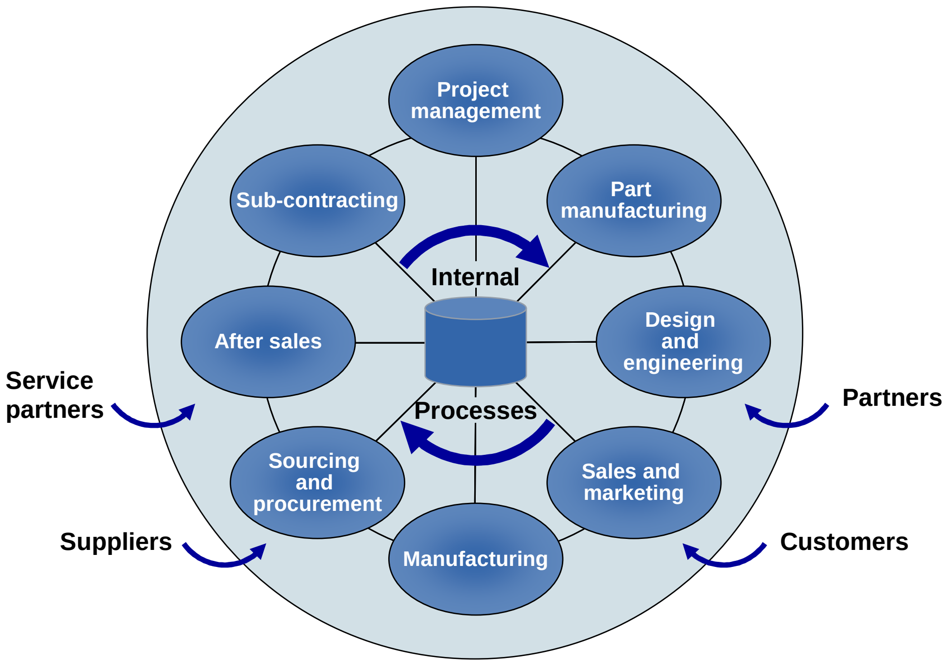


Figure 10: Product Lifecycle including interfaces

2.2.2 Benefits of PLM

Such a PLM framework gains many advantages if defined in a supportive way for development organizations.

Generally, managers think that a structured approach using PLM comes with significant business benefits, which makes PLM as a key lever to meet strategic goals. Figure 11 illustrates that 71% of the interviewed managers said to use PLM as a key lever for the reduction of “Time to Market”, which is prior to reduction of development cost (69%), the increase in product quality (59%), and the improvement of innovation (47%).

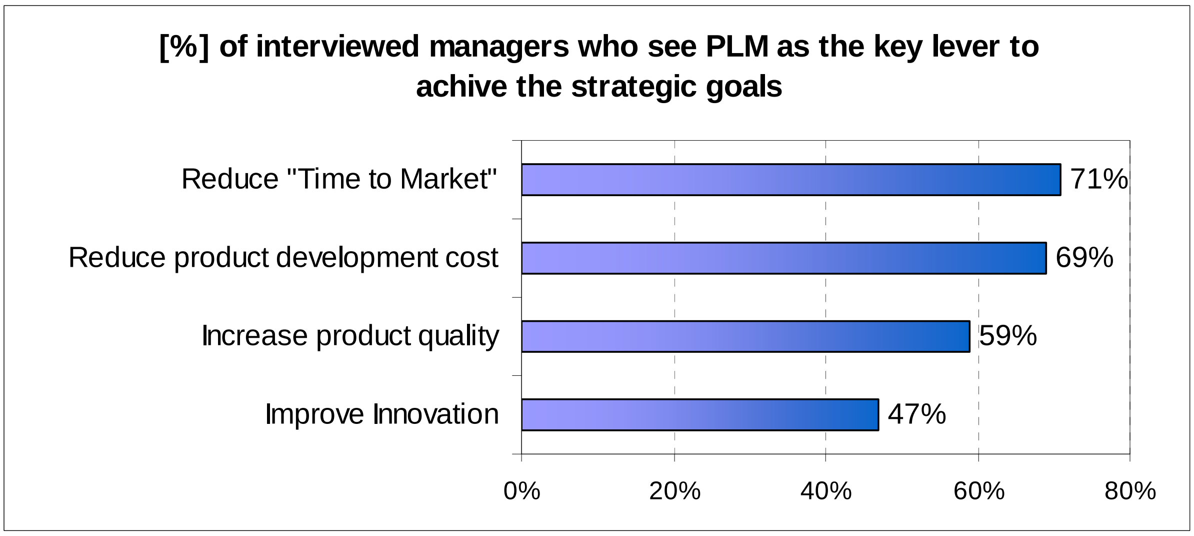


Figure 11: Strategic Goals of Managers interviewed

The dependencies of BPM and PLM get visible by comparing Figure 11 to Figure 7 in terms of results within the selection of criteria as well as the voting for each criterion.

However, the quantitative calculation of a PLM concept in terms of return of investment (ROI) is still a challenging topic, which is heavily discussed among experts. Actually, there is no quantified standard answer to this question, because every PLM is differently defined and executed in organizations. Following Arnold et al. [5] a rule of thumb is that by having a structured PLM in place one (1) Euro effort in bug fixing save more than 1.000 EURO in subsequent development phases. This is only a very rough estimation from industry experience that might not always be applicable. Generally, the benefits of course increase if more effort is taken in early development phases. However, this is no absolute linear relationship, which means that effect of benefits in “later phases” decreases the more budget any organization is spending in early development phases. Taking budget constraints into consideration a development environment has to find a compromise on how to distribute the budget between early and late development phases to get an overall benefit.

However, quantified benefits can only be measured if there is defined quantified baseline before a process improvement. This makes it merely possible to compare process indicators, e.g., “lead time” before and after improvements.

This means, in turn, that it is rather reasonable to provide some aspects that are typically nominated when it comes to estimating the ROI of PLM. These aspects are either estimations or results from improvement projects conducted in those organizations that have PLM already in place and have optimized their specific PLM. Arnold et al. state that the following success factors, which influence each other, are improved by introducing PLM [5]: Lead time, Cost, and Quality.

However, the improvement of these success factors results in several additional business benefits. Table 1 shows some of these business criteria that are positively influenced by improving the ‘influencing factors’ [38]. For instance, if lead time and cost are reduced, the business benefit “time to market” is also positively influenced, and, therefore, also reduced.

If processes are able to contribute to influencing factors, the business success factors are also improved.

Table 1: How Success Factors are influenced

		Success Factors		
		Lead Time (Time to Market)	Cost	Quality
Influencing Factors	Product and process complexity better controllable	✓	✓	✓
	Higher competitiveness	✓	✓	✓
	Reduction of prototyping cost		✓	
	Provision of framework for further product and process optimization	✓	✓	✓
	Speed of information exchange	✓		
	Reduction of wastages		✓	
	Savings through re-use of original data	✓	✓	
	Complete integration of engineering workflows	✓	✓	

2.3 Product Development Process (PDP)

A major goal of PLM is to support consistently the *Product Development Process* (PDP) using methods, models, and tools [123]. The resulting challenges and benefits are described in the subsequent sections.

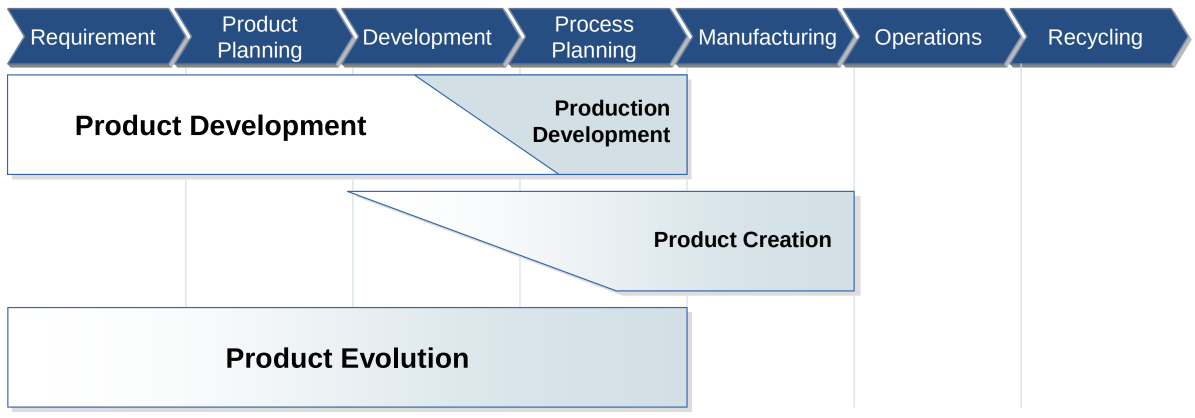


Figure 12: Relation between 'Product Development’, ‘Product Creation’, ‘Production development’ [42]

The PDP is a part of the PLM und encompasses those activities and processes that are directly connected with development and product creation. This affected processes span from requirements until process planning for production. In turn this means that processes like ‘Production’, ‘Commissioning and Operations’, and ‘Phase out’ / ‘Recycling’ are not considered. This is also depicted in Figure 12.

PDP results in a product, which consists of all production- and product documentation, feasibility studies, product specifications, and models and draft documentation needed to specify production means [42], [76].
Figure 13 shows a concrete product development cycle, which describes a manufacturing-oriented PDP, since process planning is included. Due to its nature, a purely software-oriented PDP would exclude process planning, which is, in this context, typically needed for manufacturing. Software development does not incorporate a classical production process like an assembly line. However, if, in turn, organizations consider institutionalizing processes for after- sales support to delivery software upgrade/update packages, planning and definition of a respective distribution process are essential.

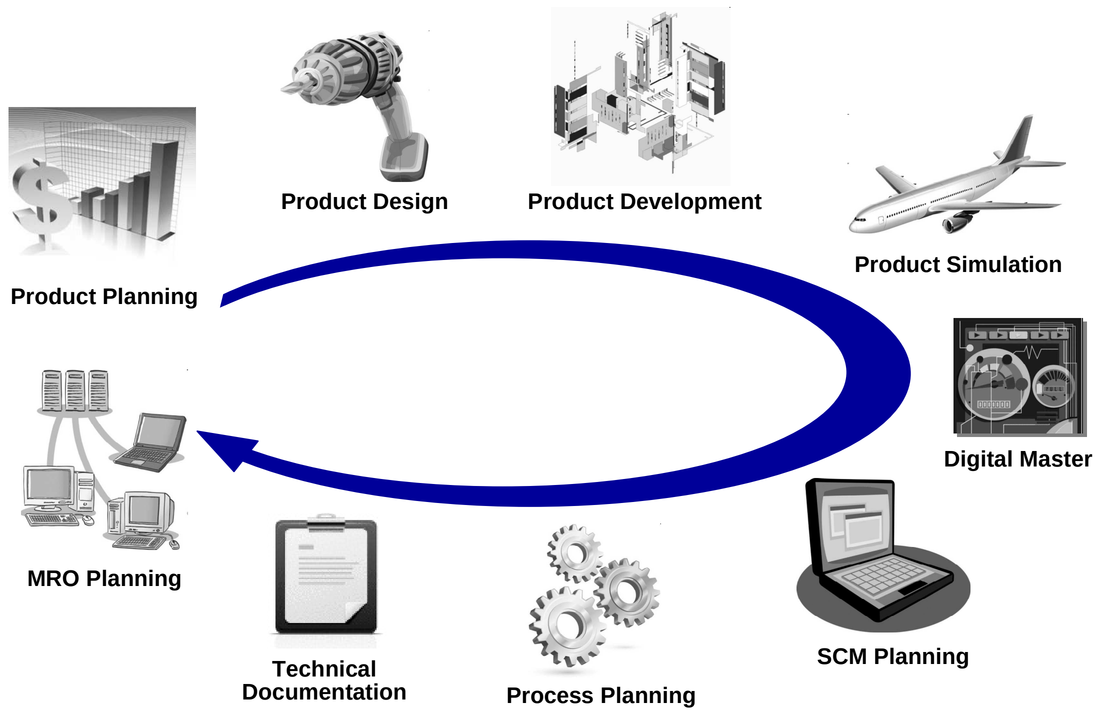


Figure 13: Phases of Product Development Process (PDP) [42]

2.3.1 Characteristics of PDP

Due to its nature, PDPs have some special characteristics that make them somewhat different to other organizational processes such as Sales or Procurement process [76].

Non- deterministic approach
Development processes are, due to their nature, not deterministic, i.e., product development always comes along with a specific portion of innovation and creativity that might change the original planned direction to be taken. The reason for this phenomenon is that at the beginning of product development, the specific knowledge for the final product is not yet available. This makes reliable planning of what the final result will look like very challenging,

which means, that a typical development project, e.g., needs some more project plan updates during its lifetime than other project plans, e.g., installation projects.

Iterative development
Iterations and “jumping back” to former development steps are also unique characteristics of a development process. This is often necessary, because adequate quality requirements are not fulfilled after the first iteration.

Significant creative portion
Development processes deal with the fact that something has to be created. This makes a significant portion of creativity during process execution necessary in order to meet any innovative requirement. Realization of attractive and innovative products is crucial for organizations to stay competitive in their business. Additionally, creativity depends on the capabilities and knowledge of individuals that underlines the non- deterministic approach.

Standardization
Highly innovative products require adequate processes that give developers the freedom to drive product development creatively, e.g., software or hardware products. This, in turn, makes a general standardization almost impossible. This means that development processes cannot be standardized and, therefore, need a significant portion or customizing and adaptation to a specific organization. Best example is the Rational Unified Process (RUP) from IBM. This iterative development process model actually comes with all aspects necessary for development. Nevertheless, the process model needs to be customized (“Tailoring” → See chapter 1.4) to be usable for a concrete organization.

Distribution of processes
The increasing dispersion is also a typical characteristic of development processes. Other processes like accounting are, in the meantime, also subject to outsourcing and dispersion, but not to that degree development processes are. The reason for this trend towards distribution is manifold, e.g., increasing cost pressure, specific development knowledge, enforcement to penetrate additional market etc. This special topic is discussed later in detail in chapter 2.5.

2.3.2 Challenges of PDP

PDP faced enormous challenges during the last decade. Among others, several reasons are responsible for this trend [116]:

- Growing competition and tighter budgets
- Globalization of business
- Shortening of delivery times
- Shortening of product development cycles
- Tightening of quality requirements / legislation

On one hand, product lifecycle gets shorter; on the other hand, product complexity increases exponentially. Figure 14 illustrates this phenomenon using the trend from the automobile industry. The number of vehicle derivates sold decreases at an increasing variety of derivates produced. One reason for this trend is that automobile manufacturers address country-specific and culture-specific features and customer preferences. Handling of higher product variety results in a much higher (internal) process complexity.

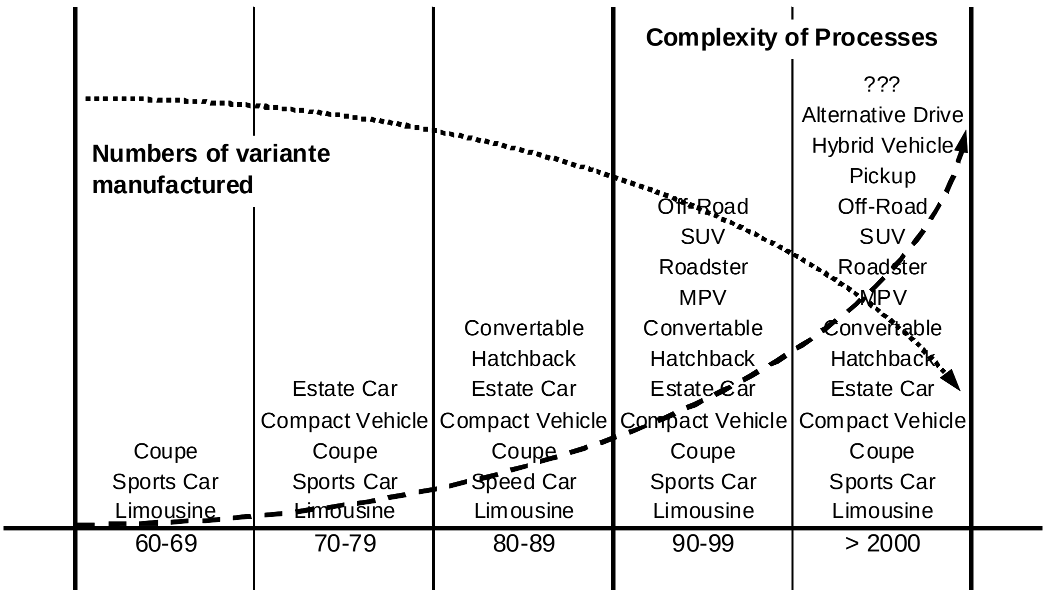


Figure 14: Trends of product strategic in the automobile industry [42]

Furthermore, global business environment makes it increasingly difficult to survive in the development business. In order to address the customer's need with increasing accuracy, organizations are enforced to provide the market with more product variants. This causes additional pressure on processes, especially towards process platforms and product line management [106]. The defined processes are forced to be optimized in order to work more efficiently.

Moreover, organizations have to handle the fact that indeed products get continuously more complex, but customers have been also given more and more opportunities to influence products' features and configuration, which is necessary to be still attractive for customers.

Organizations are challenged by strategic adaptations that are necessary to successfully compete in a local, national, and even an international market. A survey conducted by Accenture [36] in 2008 asked renowned European IT organizations about their upcoming changes and challenges within in the next five years. The result is depicted in Figure 15. Amazingly, about 90% of the interviewed organizations need to align their R&D strategy globally. 85% want to improve their innovation capabilities, and 72% are challenged by establishing engineering collaborations. This reflects a clear trend towards globalizations and collaborations.

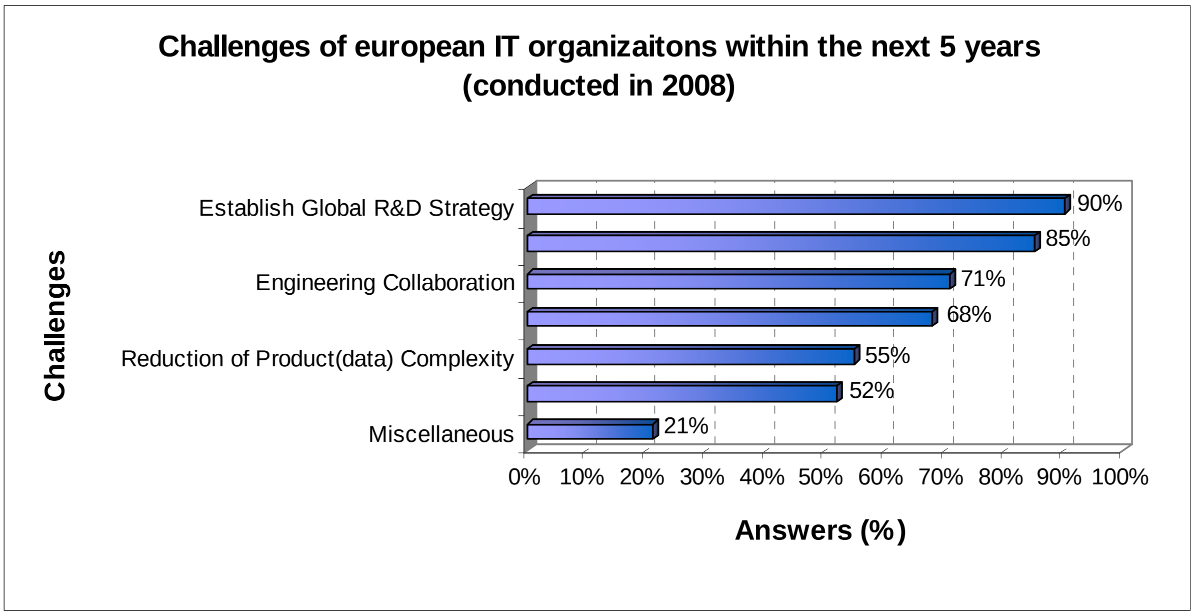


Figure 15: Challenges of European IT organizations within the next five years

2.4 Software Development Models and Modeling

2.4.1 Diagrams and Connectors

Software Models are often illustrated by using diagrams. In general, a diagram is a two/three dimensional representation that shows information in a geometric way. There is an enormous amount of different diagram types available. Basic diagram types encompass graph-based diagrams (e.g. tree diagram, network diagram, Venn diagram, or flow chart), chart diagrams (e.g. histogram, bar chart), and others (e.g. three dimensional diagram). For an overview of diagram types please refer to [146].

State diagram is often used in conjunction with flow charts (graph-based diagrams). Figure 16 gives a differentiation of these terms. Whereas the state machine is the performed respond action to an explicit event, the flow chart does not need explicit events, but rather transitions from one node to the subsequent node [117].

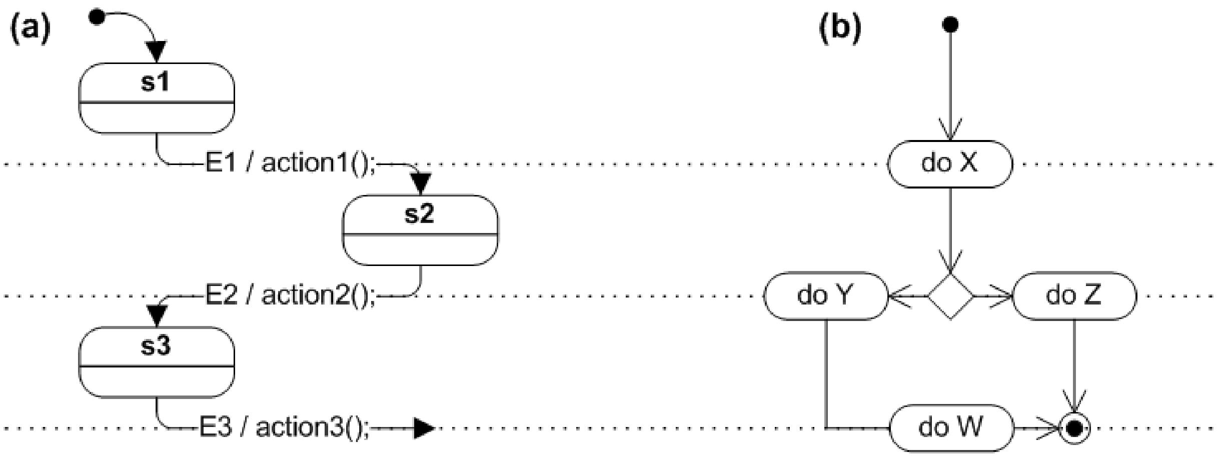


Figure 16: Comparison Sate Diagram (a) and Flow Chart (b)

State diagrams in UML are the basis for UML activity diagrams, which are primarily used for this dissertation.

Connectors are a basic element for modeling of diagrams or charts. Referring to Figure 16, the connectors are ‘edges’ between various states (=a) or nodes (=b). In computer science, a connector is the connection, i.e., pointer or a link between any data structure. Referring to Figure 21, the connector is represented by the directed arc of a petri net.

In informatics, especially processor programming, the definition of connector is a so-called (bit-wise) *composition operator*. The major operators that are relevant for this dissertation are the bitwise operators XOR, OR, and AND, which are well known as primitive actions that can be directly used by a computer processor [112]. This work takes advantage from these operators by accurately assigning responsibilities to specific tasks within the role model provided. The basic definitions encompass [17]:

- **XOR** - takes two bit patterns of equal length and performs the logical XOR operation on each pair of corresponding bits. The result in each position is 1 if the two bits are different, and 0 if they are the same.
- **OR** - takes two bit patterns of equal length, and produces another one of the same length by matching up corresponding bits and performing the logical inclusive OR operation on each pair of corresponding bits.
- **AND** - takes two binary representations of equal length and performs the logical AND operation on each pair of corresponding bits. In each pair, the result is 1 if the first bit is 1 AND the second bit is 1. Otherwise, the result is 0.

2.4.2 Process (Meta-) Modeling

Process modeling within development organizations (also known as Business Process Modeling) is the activity of illustrating and representing processes of an organization. This visualization follows the purpose to get better process support in terms of definition, control, and adherence by users. Additionally, processes are in the right “shape” to be analyzed and improved, due to process structure already given. Process modeling is mainly challenged by connecting several processes of the same nature together to one abstract process, which can be applied in as many cases as possible in reality. Thereby, a process Meta model may consist of several abstraction levels like illustrated in Figure 17 [32], [100].

Process Meta-Meta Models (M0-level)
The Object Management Group (OMG) has defined a four layer modeling architecture called Meta Object Facility (MOF^{TM}), which originated from UML [24]. The MOF^{TM} provides a Meta-Meta model on the top level, also called M0-level in Figure 17. From this level Meta models are derived [100].

Process Meta Models (M1-level)
Process Meta modeling is one type of Meta-Meta modeling, which is especially used in software and system engineering. There are several well-known Process Meta models assigned to M1-level in Figure 17. OMG has defined the UML Meta model.

Furthermore, the OMG has also defined the Software Process Engineering Metamodel (SPEM). SPEM defines and models software development processes and its components [79].

The basis of SPEM is a subset of UML Meta model resulting in a process description that was provided especially to software development industry.

Another Meta model that has been derived from UML Meta model is British Ministry of Defense Architecture Framework (MODAF). The MODAF Meta Model extended the UML Meta model 2.1 by creating a UML profile resulting in an architecture framework, which defines a standardized way of conducting Enterprise Architecture, originally developed by the UK Ministry of Defence [96].

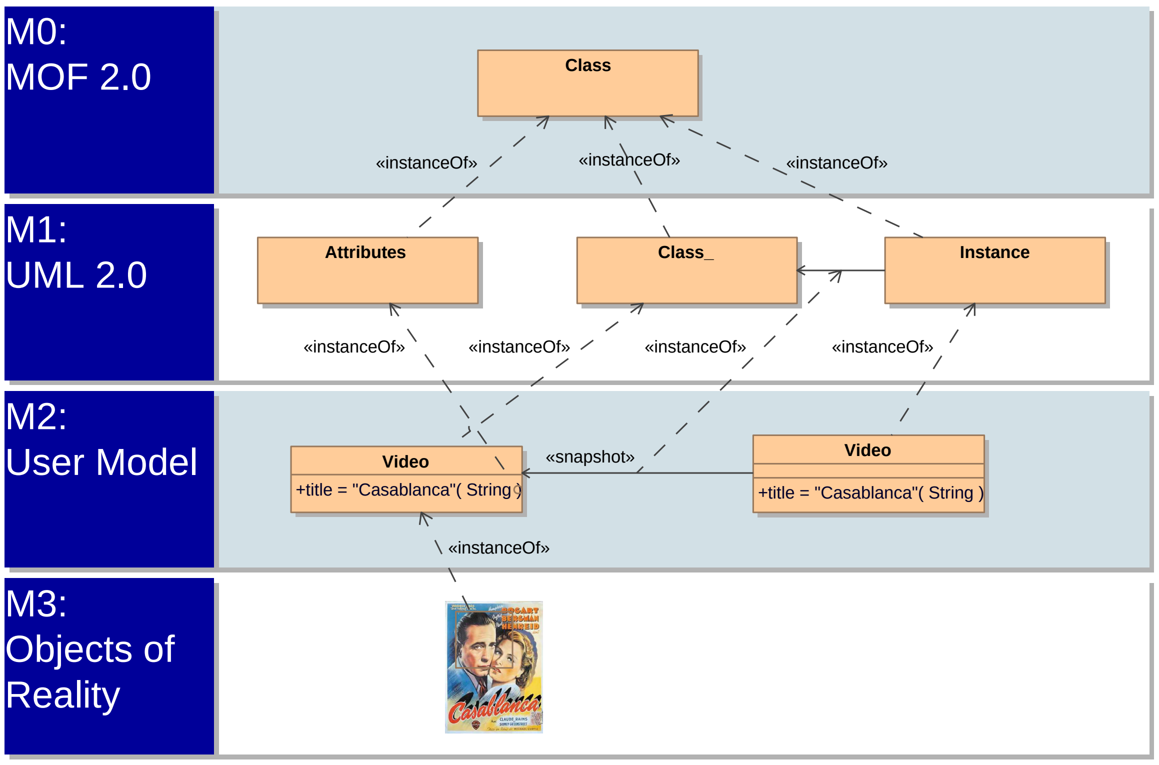


Figure 17: Four Layer Architecture of the Meta-Object Facility (MOF)

Process Models (M2-level)
Process models in (software) engineering are a networked sequence of activities that express an organizational development strategy and allow them to conduct evolution [133]. These models are derived from Meta models, which would be M1-level in Figure 17 referring to the MOF framework.

Process Instances (M3-level)
The M3-level, which is the lowest level in OMG's MOF^{TM} provides all data and objects of reality for the layers above.

2.4.3 Software Process Modeling Techniques

Techniques to model business process such as the flow chart, functional flow block diagram, control flow diagram, Gantt chart, or PERT diagram have emerged since the beginning of the 20th century. The most often used business modeling methods are Event-driven Process Chains (EPC) and Business Process Modeling Notation (BPMN). Software engineering takes advantages from Unified Modeling Language (UML). Although there are many other modeling techniques, e.g., Cognition enhanced Natural language Information Analysis Method (CogNIAM), Extended Business Modeling Language (xBML), ICAM DEFinition (IDEF0) [140], these are commonly used standards when it comes to process or software modeling. The three methods are shortly introduced in the following.

BPMN's objective is to support business process modeling resulting in a business process diagram that is intuitive to users even in a complex process environment. Figure 18 depicts an example of a chart using BPMN.

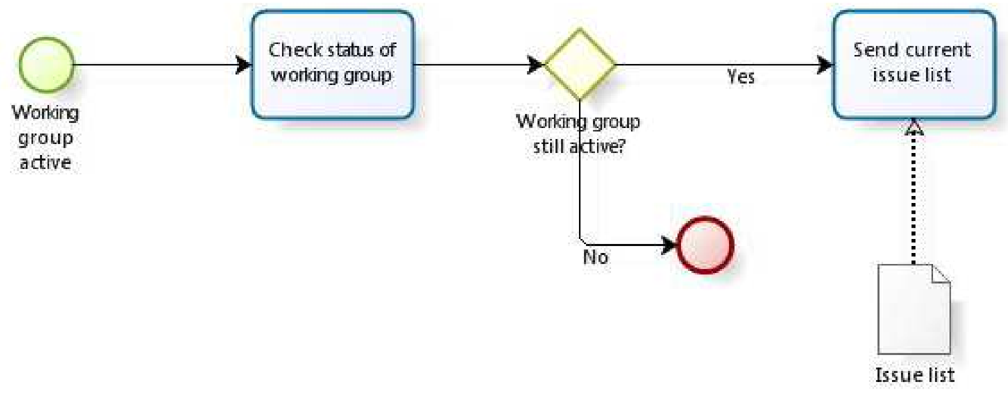


Figure 18: Example of a Business Process Modeling Notation chart

UML is a general-purpose modeling language and is the pre-dominant language for visualization (object-oriented) software engineering and modeling. This technique supports specification, construction, and documentation of especially software intensive systems. However, the variety of graphic notation techniques also allows for software process modeling (e.g. by using UML activity diagrams), because it combines techniques of data modeling (e.g. entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. The variety of UML diagram is illustrated in a collage in Figure 19 [100].

These are divided in

- Structured Diagrams, e.g. class diagrams, component diagrams
- Behavior Diagrams, e.g. activity diagram, use case diagram, UML state machine
- Interaction Diagrams, e.g. sequence diagram, timing diagram

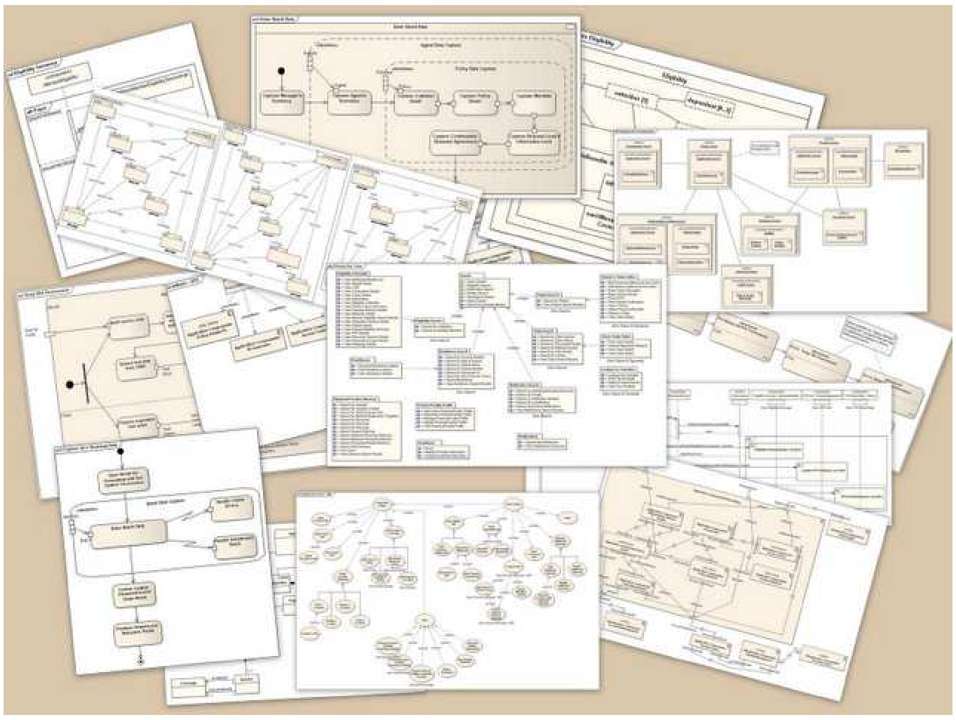


Figure 19: A collage of UML diagrams

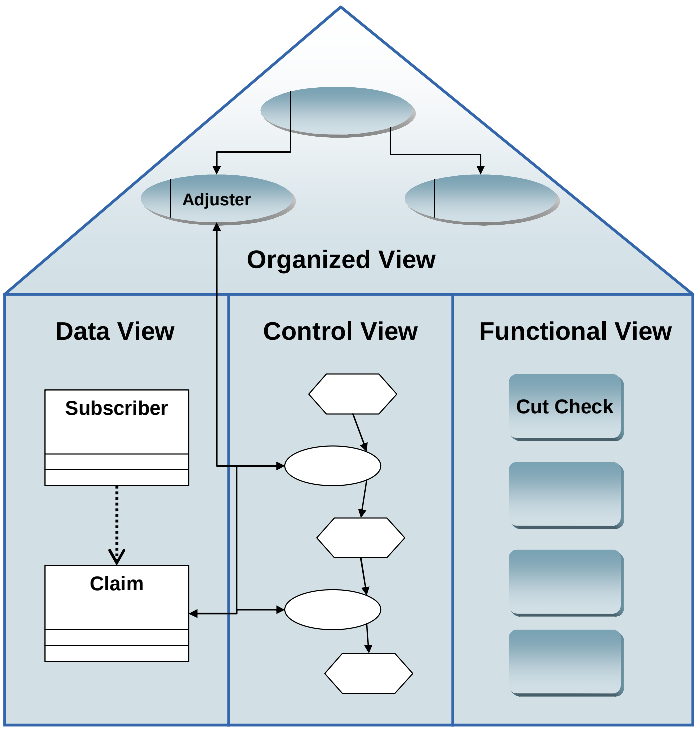


Figure 20: The four views of an Event-driven Process Chain (EPC)

Event-driven Process Chain (EPC) is a widely used type of flowchart used for business process modeling and process improvement. It is a multiple-view approach that is able to link the control view to roles and to entities in the corporate data view and functional view, which is depicted in Figure 20. EPCs are the preferred modeling technique in SAP/R3[6] and ARIS [133], [58].

Petri Nets are a mathematical modeling language for describing distributed systems that were invented in August 1939 by Carl Adam for the purpose of describing chemical processes [53]. A petri net also offers a graphical notation, which consists of **places** (state elements or p-elements), **transitions** (transition elements, t-elements) and **directed arcs**. The arcs connect places and transitions in a way that they run from places to transitions and vice versa. Places may contain a natural number of tokens, which are distributed over all places of the net. This so-called **marking** enables the transitions to fire or transform all input tokens (from input places) to the output places through the directed arcs. Figure 21 shows a petri net with basic t- and p-elements, directed arcs, and tokens [40].

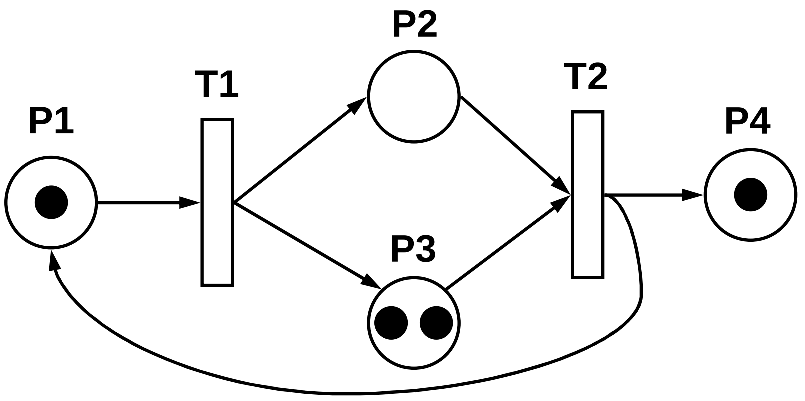


Figure 21: Example of a Petri Net

The principle of directed arcs or directed graph is applied for the classic form of a state diagram, which is a finite state machine. A state diagram is the graphical representation of a finite state machine. This tool is used especially in computer science to describe system behaviors. Thereby, the system needs to be set up with a finite number of states. There are numerous types of state machines, e.g., deterministic finite state machine (DFA), non-deterministic finite state machine (NFA), generalized non- deterministic finite state machine (GNFA), or Moore machine. For more information on state machines, please refer to [148]. Alternatively, state machines can be represented by State Transition Tables [117].

2.4.4 Software Development Models

Each process model consists of several basic elements that are necessary to be able to describe any desired process. These core elements are:

- Activities, sub-processes, and activity chains
- Input and Output parameter
- Actors for activities
- Objects (data, artifacts)
- Events and messages
- Branching and merging

[6] System Analysis and Program Development (SAP) developing an Entrprise Resource Planning System

- Checks and decisions
- Interfaces to other/external processes

Activities are connected via relationships to an activity structure or flow. These activities are performed by roles and responsibilities, meaning the employees of an organization. They are supported by additional resources like machines or computers (IT-infrastructure). Each activity is executed by applying a defined method, which takes information or knowledge and precedes it respectively (artifacts). Branching and merging are methods for parallelization in terms of activity execution [76].

One of the first process models used for software development was the Waterfall model (Figure 22). This sequential design process was originally a process model for hardware development and has been adapted for software engineering due to the lack of model at the beginning of software engineering. The model consists of numerous development steps starting from requirement engineering down to operation and maintenance phase, respectively [112].

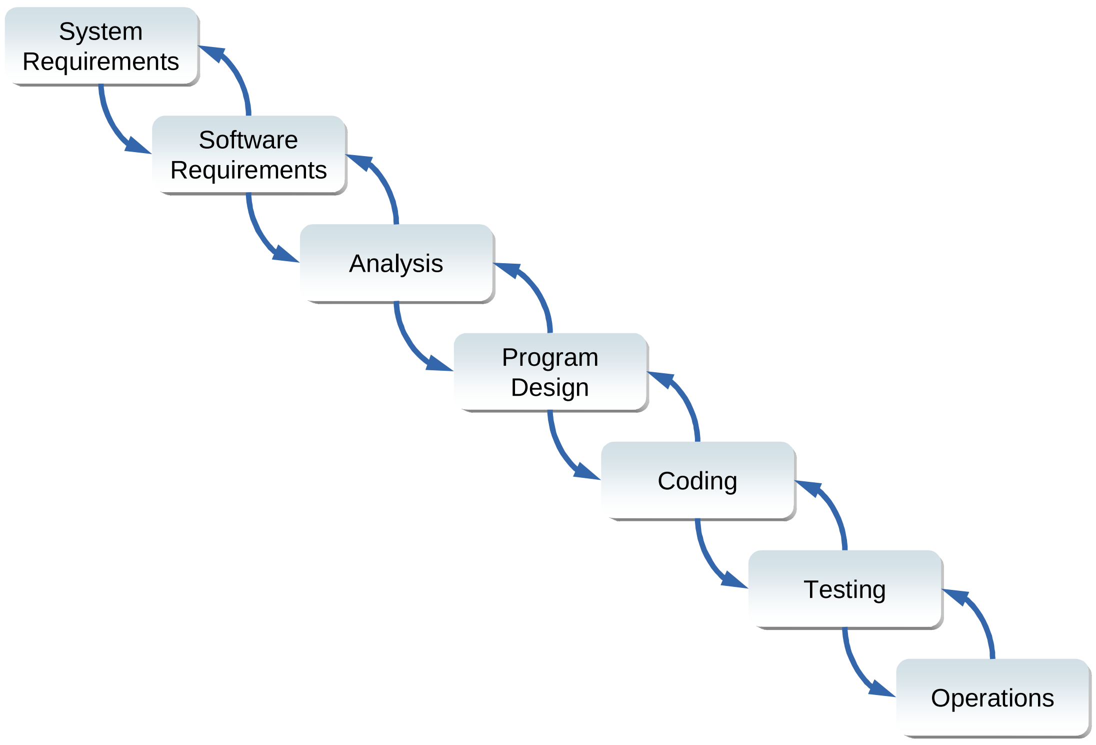


Figure 22: Waterfall process model (incl. fallback loop)

A further development of the Waterfall model is the V-model [21]. A major characteristic is that each development step on the left arm of the "V" comes along with an appropriate counterpart test activity on the right arm. The major difference to the waterfall model is that test phases are defined as succession of development activities on the right V-arm (Figure 23).

The V-model XT® - as the latest revision - is one example of a comprehensive process model providing as hands-on support for product development. The V-model XT® is the international acknowledged standard development model for conducting IT projects [79]. This model is actually a so-called procedural model, since it provides very supportive hands-on descriptions on how to run a software project. In order to define those non-mandatory process model elements necessary for a concrete project, Tailoring (→ See chapter 1.4) is applied.

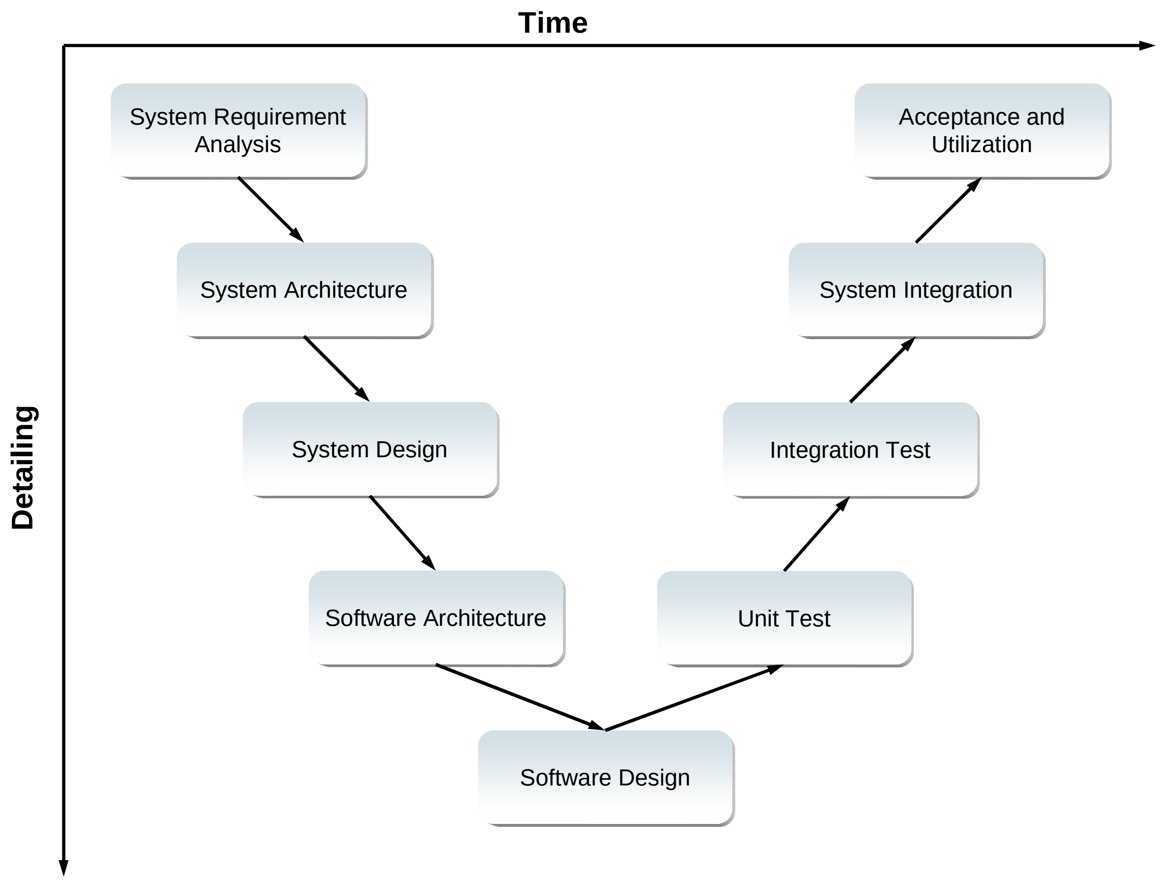


Figure 23: V-Model

The V-model has been also the basis for the W-model as a further extension [134]. Compared to the V-model, W-model defines the test activities already in parallel to development activities Figure 24.

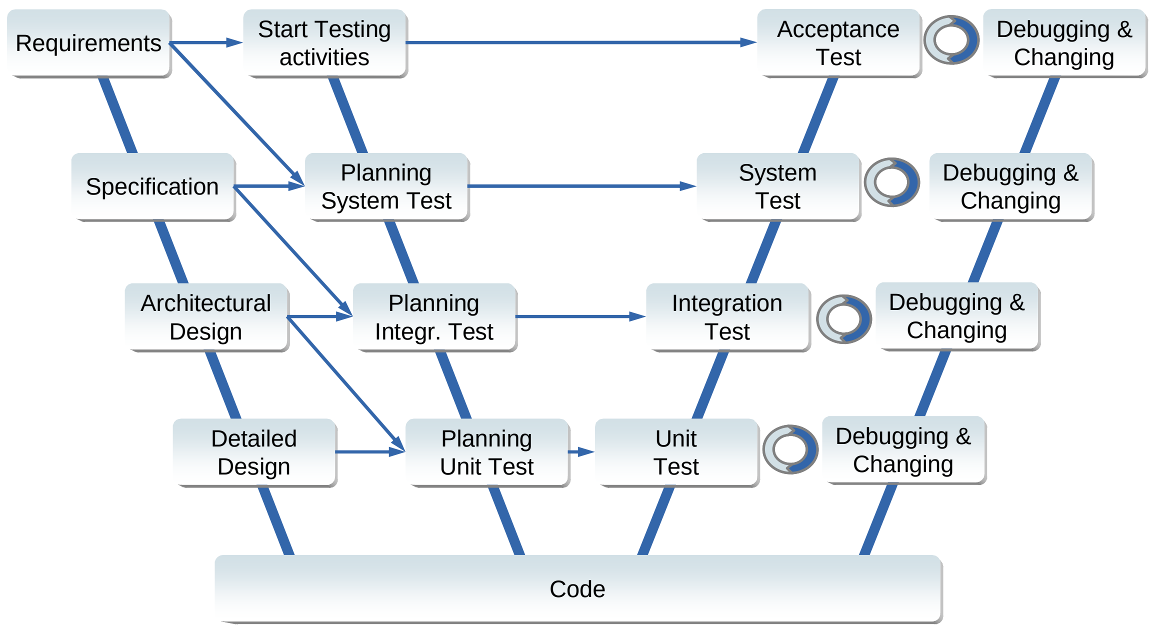


Figure 24: W-model

A further type of process model is the Spiral model. The Spiral model combines the idea of iterative development (prototyping) with the systematic, controlled aspects of the Waterfall model. The Spiral model also explicitly includes risk management within development. As depicted in Figure 25 the product to be developed is refined every iteration of the spiral. This allows for incremental releases of the product each time around the spiral [20].

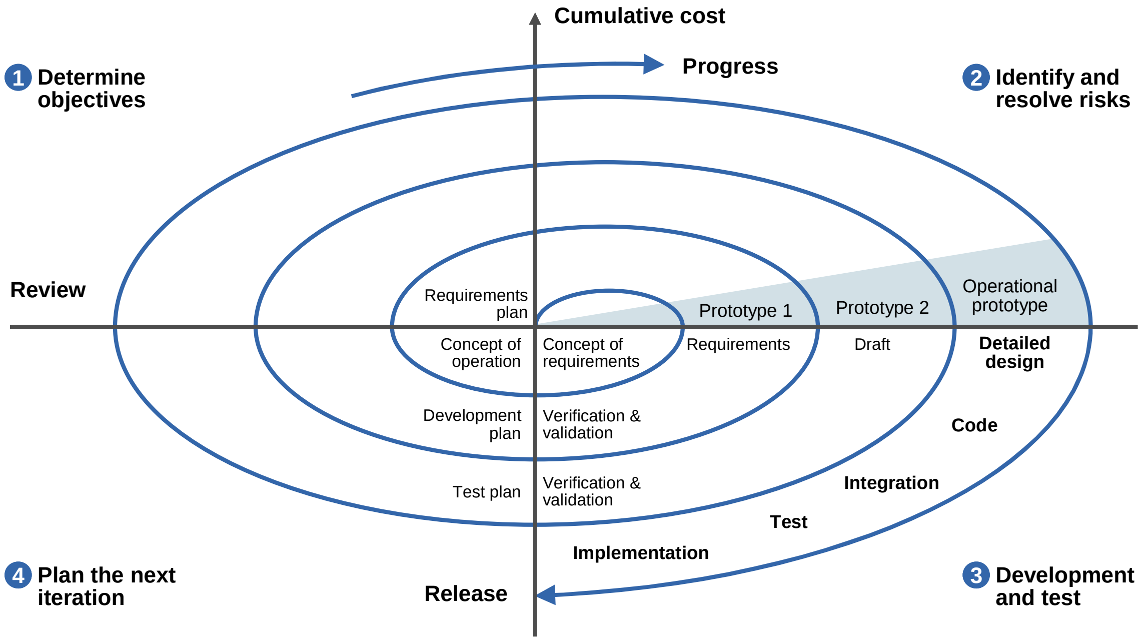


Figure 25: Spiral model

The Rational Unified Process (RUP) is another iterative and incremental software development process model [53]. RUP is a specific implementation of the Unified Process that uses the UML as description language (Figure 26).

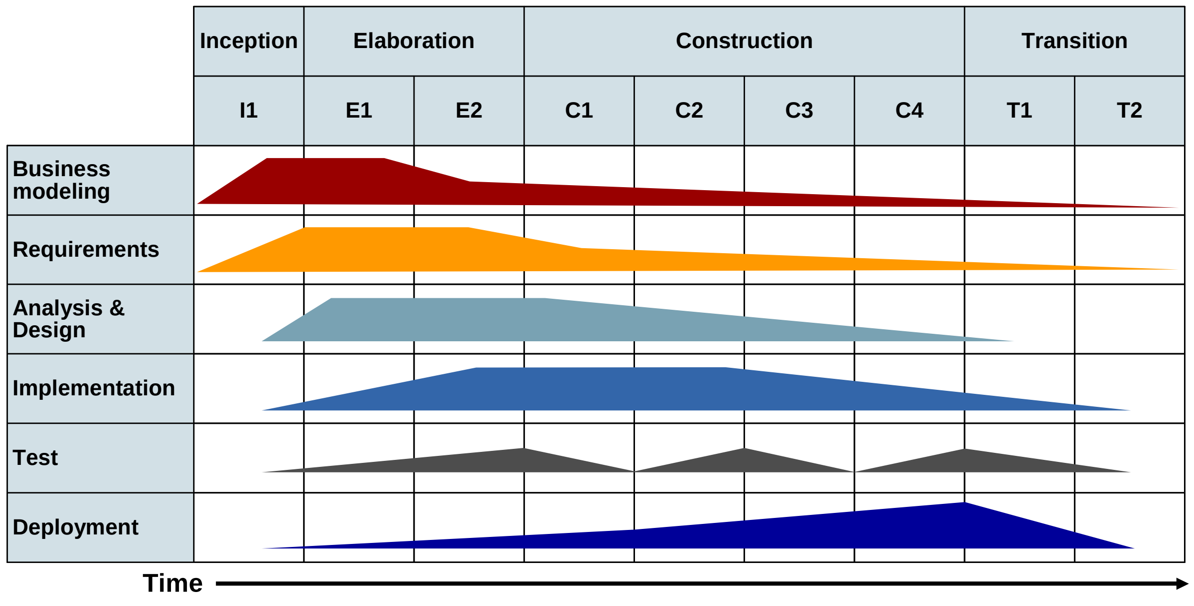


Figure 26: Rational Unified Process (RUP)

Specific characteristic encompasses not a single concrete prescriptive process, but rather an adaptable process framework, also intended to be tailored (→ See chapter 1.4) by the development organizations and software project teams. Business value is delivered incrementally in time-boxed cross-discipline iterations [79]. Figure 26 gives a short sketch on efforts that typically occur in various iteration phases.

Agile Software Development basically consists of a couple of software development methodologies origin form iterative and incremental software development. It is based on the Agile Manifesto[7] in which this term was first introduced in 2001 [15].

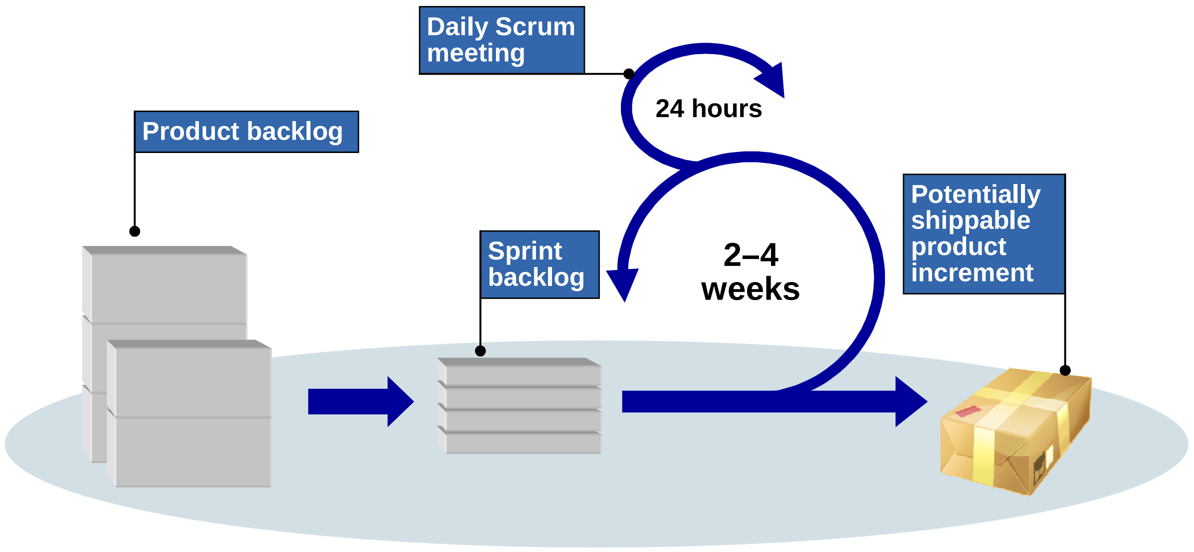


Figure 27: Overview of SCRUM in agile development

Software development methods encompass besides others Scrum [125], Crystal Clear, Extreme Programming, Adaptive Software Development, Feature Driven Development, and Dynamic Systems Development Method. Thereby, SCRUM is the central project management method used in Agile Development (Figure 17).

The Capability Maturity Model Integration (CMMI®) also counts to process models as a quasi-standard [133]. CMMI is a collection of best practices that has been derived from industry, government, and scientists. Currently there are three constellations available:

1. Product and service development — CMMI for Development (CMMI-DEV),
2. Service establishment, management, and delivery — CMMI for Services (CMMI-SVC), and
3. Product and service acquisition — CMMI for Acquisition (CMMI-ACQ).

CMMI may guide development organization on one hand through process development and improvement. On the other hand, it supports organizations appraising its processes and developing process maturity. This is done by means of maturity levels (Figure 28).

[7] http://agilemanifesto.org/

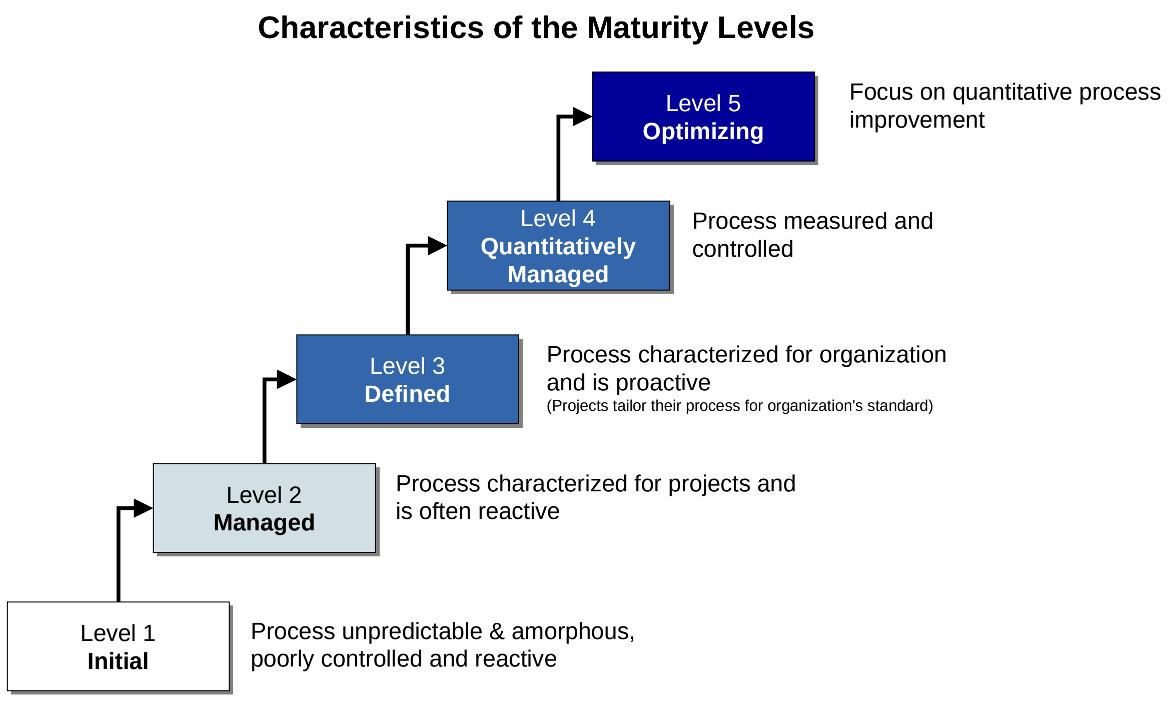

Characteristics of the Maturity Levels
Level 5
Optimizing
Focus on quantitative process improvement
Level 4
Quantitatively Managed
Process measured and controlled
Level 3
Defined
Process characterized for organization and is proactive
(Projects tailor their process for organization's standard)
Level 2
Managed
Process characterized for projects and is often reactive
Level 1
Initial
Process unpredictable & amorphous, poorly controlled and reactive

Figure 28: Maturity levels of CMMI

Based on defined process models, concrete instances of processes are derived that can be executed during project run. This level of a process model is defined in the M2 layer in the MOF framework as introduced in Figure 17. The concrete process instantiation on this level is the consolidated description of actions and activities that result in artifacts created by using defined methods (Objects of Reality, M3 layer in Figure 17).

Further process improvement modes are described and defined in standards ISO 9000 [72] and ISO/IEC 15504 Software Process Improvement Capability Determination (SPICE) [71].

2.5 Global Product Development and Engineering

The phenomenon of globalization goes back to the seventeenth century when so-called “multi-national companies” appeared, which operated in multiple countries. These global acting organizations play an important role in terms of trading, like the first multinational organization "The Dutch East Indies Company”, founded in 1602.

A major trend towards globalization started with a first wave in the nineteenth century. In this time, laissez-faire economic theory occurred and enforced nations to reduce or remove tariffs that limited the movement of goods. Additionally, the acceptance of the gold standard in the second half of the nineteenth century led to more global- oriented activities.

The second wave of globalization began near the end of the Second World War with a meeting in New Hampshire in 1944 that led to the foundation of The World Bank, The International Monetary Fund.

The computer industry became also an essential part of multinational business, since the computer itself attracted many industry organizations, not only from business machines, electronics, defense industry, but also included important entrepreneurial start-ups. These were among others: General Electric (formed in 1895 and entered the computer industry in the 1950s), IBM (consolidated in the tabulating business in 1911), Hewlett Packard (formed in 1939 as an instrument maker and entered the computing industry in the 1960s), EDS (formed in 1962 to serve large users of computers), Microsoft (formed in 1975 to provide products in the microcomputer software industry), and Dell (formed in 1984 to provide microcomputer hardware).

The Offshoring business model had been started in the 1960s in the semiconductor manufacturing. U.S. companies, also begun in the 1970s, began to move to move manufacturing activities such as labor-intensive chip assembly to low-wage countries in East Asia, including Singapore or Hong Kong. European companies followed later. Amazingly, by end of the 1980s, East Asia had the capacity to provide circuit boards and electronics products to the entire world. Software and IT services sector started the Offshoring wave in the early 1980s.

In the 1990s India, which was among the early entrants in the Offshoring business, began with so-call “body-shopping”. This was the process of sending trained programmers to work for a few months in another country on the client firm’s premises. This was followed by a blended strategy in which some of the work was done on the client’s site and some at the vendor’s site in India.

In the last five to ten years, facilities even began to carry out IT-enabled business processes such as accounting. More recently, Indian firms have begun to move up the value chain to do IT-enabled knowledge processing such as reading X-rays, conducting patient analyses, and carrying out IT research and advanced development. (Sources: [1], [6], [141]).

Although it is been thirty years that companies are trying to take development advantages for this “networking trend”. This means that development of desired (sub-) products takes place in those sites that fit best into overall organizational strategy, e.g., low- cost countries. Therefore, it does not matter where people with respective key knowledge come from as long as they are able to get “online”. Serious development process activities on a global basis, however, started within the last decade. For general aspects and research, please consider [63], [86], [94], [29].

These days Research and Development (R&D) experiences a tremendous trend to distributed and global development. Going “global” is a popular business model and still accelerates in many development organizations around the globe. Buzz words like ‘Globalization’ or ‘Internationalization’ are often used by top managers to emphasize current and future business trends and strategy.

But why are globalization and cooperation with foreign organizations so interesting for development organizations?

In the subsequent section, globalization is discussed from various points of views.

2.5.1 Companies going abroad

As already highlighted in the previous section, globalizations of especially software development started in the 1980s. This new trend was characterized by prior technology-oriented companies that tried to boost their innovative capabilities by setting up globally dispersed team and building globally R&D networks. The wave of globalization lasts until today.

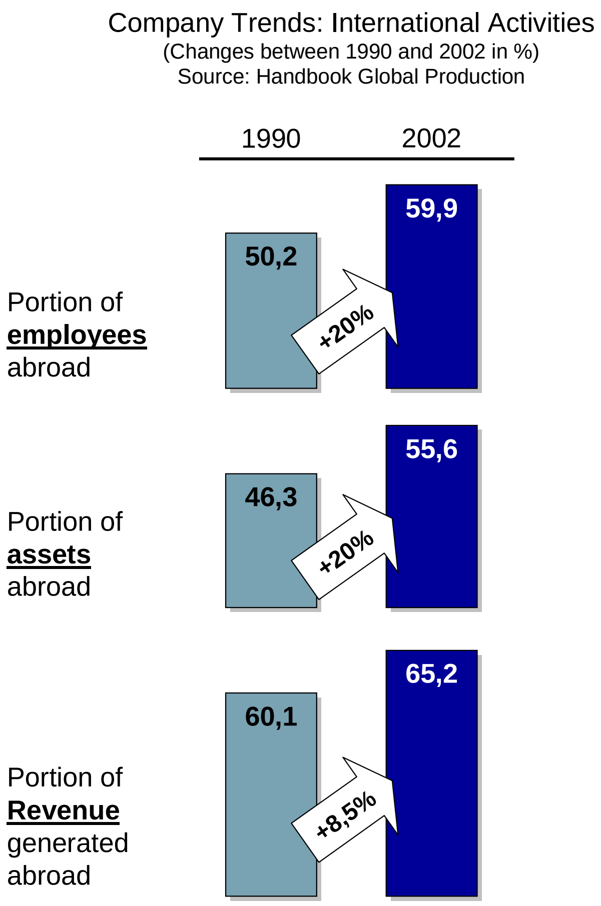


Figure 29: International Activities of Companies [136]

This means that companies located, e.g., in Germany move their business activities more and more abroad or at least include foreign workforce into their specific business processes. This is shown in Figure 29, which compares number of employees, assets, and foreign revenue in 1990 and 2002. As illustrated, the workforce and assets abroad grew by 20%. The portion of revenue organizations achieved abroad increased by 8.5%. These numbers make clear that international business is apparently important for companies in Germany.

Internationalization concerns especially outsourcing activities of several business domains. Figure 30 shows offshore penetration of five different sectors, namely R&D, Engineering, IT, Finance, and Human resources. As one can see, IT has by far the highest off-shoring penetration. However, the major source of IT, which is R&D and Engineering have the steepest growth rate of 38% and 50%. Therefore, the steadily remaining assumption that only IT service and help desks are subject to off-shoring cannot be confirmed anymore.

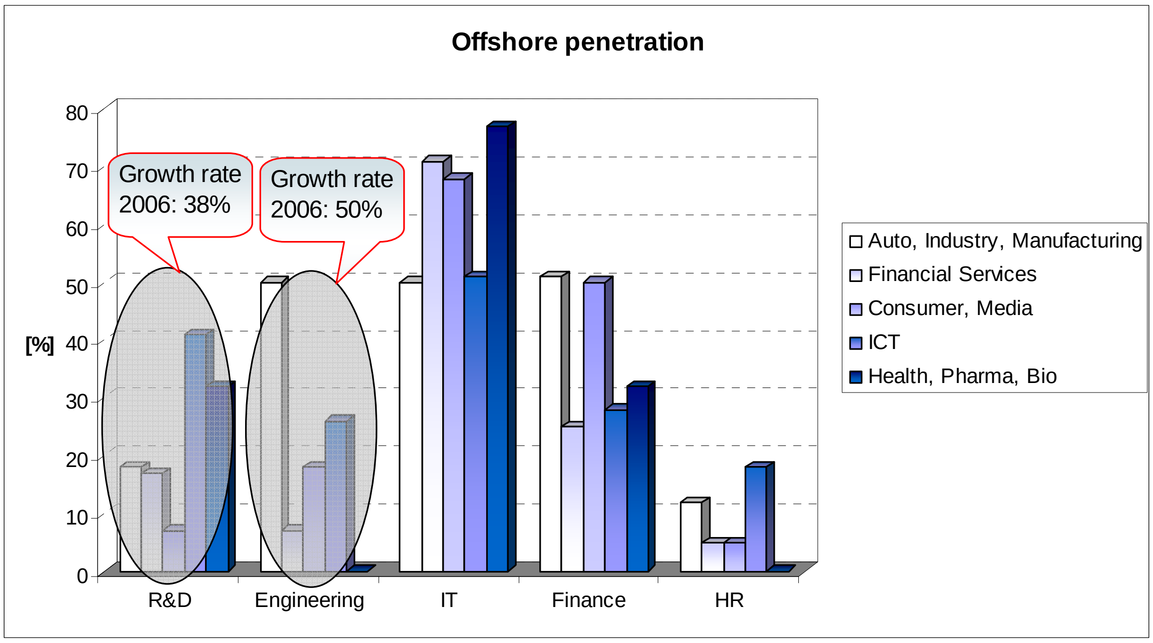


Figure 30: Offshoring and globalization across industries

Using a concrete example from a company, the trend to globalization is once again depicted in Figure 31, which describes the job shifting from a Siemens perspective[128], [129], [130], [131], [153]. Siemens is a German technology company that experiences a shift of employees working in Germany towards abroad. In 1993, about 60% (238.000) of the overall 391.000 employees were working in Germany. Ten years later (2003), about 60% (247.000) of the workforce is employed abroad. Today in 2011, only one third (127.000) Siemens employees work in Germany alone. This trend signals a movement of Siemens activities to other countries than Germany.

Globalization is furthermore indicated by turning around the point of view, i.e., looking at globalization from a low-cost country's position, e.g., India. This country has generated a significant increase in revenues within software engineering exports in the last ten years. Figure 32 demonstrates that India’s exports in IT and software engineering have been increased almost exponentially since 1998. China also plays a considerable role in terms of software and IT services exports. Its Software and IT Service exports have not been boosted like those of India, but China’s revenue increase is also remarkable. This means, in turn, that organizations in high-cost countries are heading for India to get software implemented at lower cost rates.

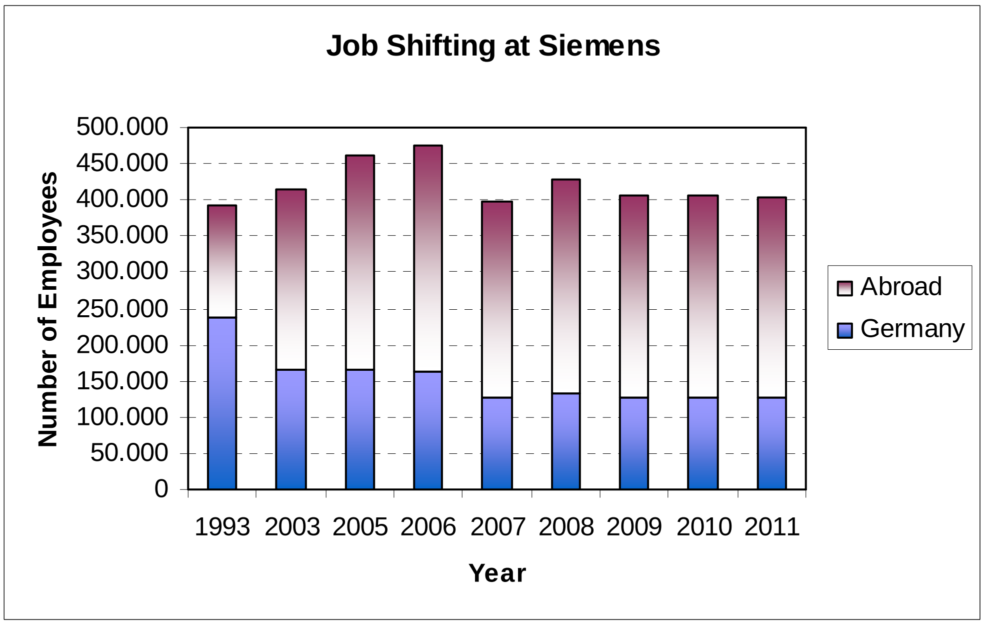


Figure 31: Job Shifting at Siemens AG

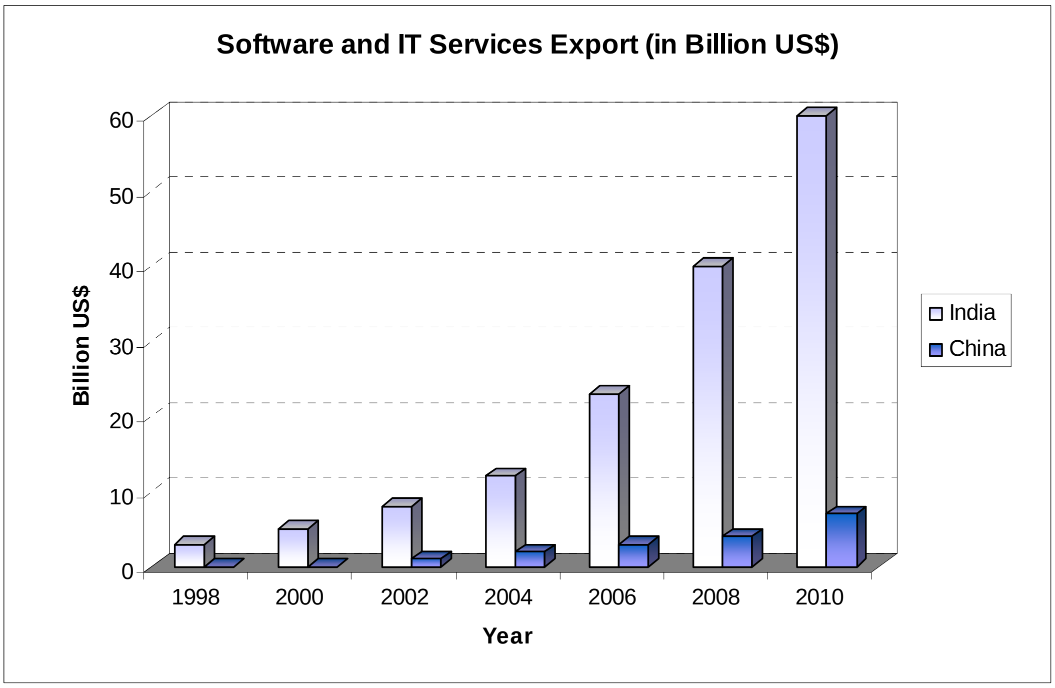


Figure 32: Software Engineering export revenues from India and China [79]

2.5.2 Terminology

The term "Global (Product) Development" is used if team members from different and multiple geographic locations participating in one development project 0[118]. Those team members could be within one country, but they might also be dispersed all over the globe. Thereby, collaborations lead to a valuable network of cooperation between customers, suppliers, and partners [18]. This section first provides some different terminology and definition on Global Development.

In general there are some major terms related to off-shoring, which are depicted in Figure 33 [98]. The explanation of this terminology requires a defined perspective. For definition purposes, it is assumed that the following definitions are seen and shortly explained from a German company's perspective, which is "Onshore/In-house" [79], [92].

- Onsite/In-house
  Processes and development of products are not to be given to any other organizations (in Germany and abroad). Therefore, they stay "In-house" or "Onsite" respectively.
- Onshore
  Processes or development of products are given to a domestic supplier. If the master company is German, the onshore supplier is also a German organization.
- Nearshore
  Processes or development of products are given to a supplier on the same continent. From the German perspective "Nearshore" would concern, e.g., Poland.
- Offshore
  Processes or development of products are given to a supplier, which is not on the same continent. From the German perspective "Offshore" would concern, e.g., India.

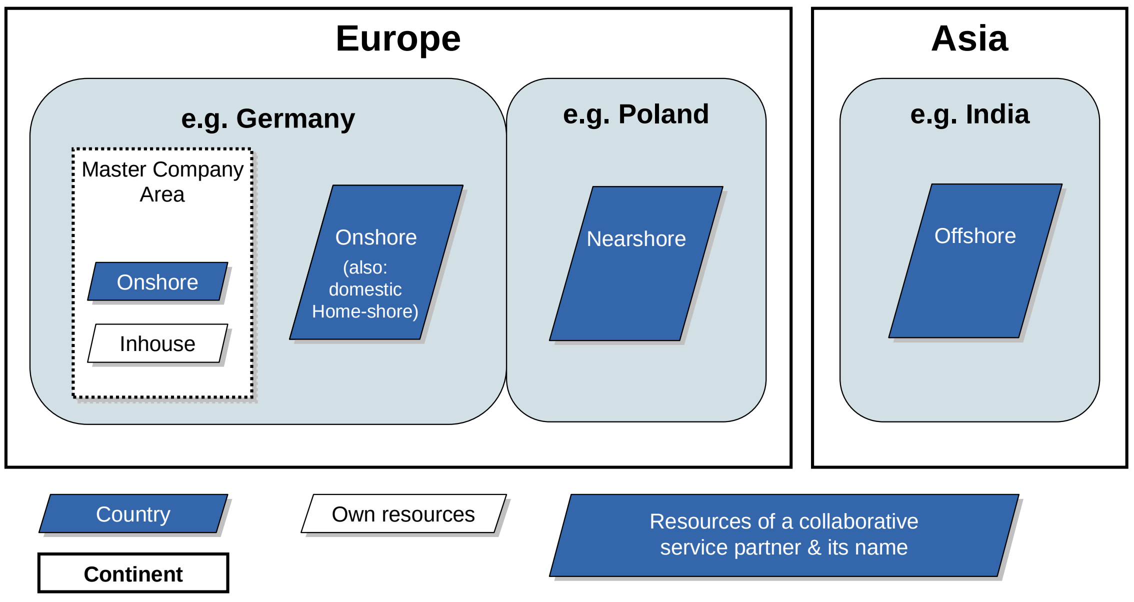


Figure 33: Terms and relations concerning off- and on-shoring

The term 'Offshore' is the one most often used when talking about business movements abroad. Typically, all three defined process types (operational processes, support/service processes, and core processes according Figure 9) of PLM processes are potentially subject to off-shoring.

Interestingly, management processes, e.g., 'Strategic Planning & Implementation', are normally not off-shored. The point is, that by giving these processes away would mean that any organization gives the whole business away, which is typically not intended.

However, the 'Off-shore'- models are rather useful for business improvement in terms of cost reduction etc. The (software) development discipline itself is rather less positively affected.

This was also stated by Dieter Rombach during his Keynote Speech at the International Conference on Global Software Engineering (ICGSE)[8] in Limerick July, 2009: "A software development project with two teams in one location is hard enough to conduct successfully. Why does anyone think it will be easier by putting 5.000 miles in-between?"

This means that geographical dispersed development projects face enormous challenges concerning set-up, management, and controlling. Therefore, the cost reduction effect has to over-compensate the additional coordination and communication effort that a project takes on by going off-shore.

Furthermore, several authors defined terms to describe cross-border development and engineering activities.

Eigner and Stelzer [42] have defined Cross Enterprise Engineering (CEE) that expresses the collaboration beyond any organizational borders. The original term comes from James Champy, who describes the term X-Engineering ("Cross-Engineering") as the following:

> "The walls between a company, its customers, and its suppliers – even between competitors – are coming down. In a world of free-flowing information and Products, X-ENGINEERING the cooperation reveals a radical new vision of the Cooperation." [31]

Champy talks about a multi-dimensional cooperation and collaboration by either organization internally or between customers and suppliers [31]. Thereby, organizational borders are not relevant anymore and all disciplines, e.g., software - or hardware development, etc. are supported throughout the whole product development process (PDP). This innovative engineering approach is depicted in Figure 34.

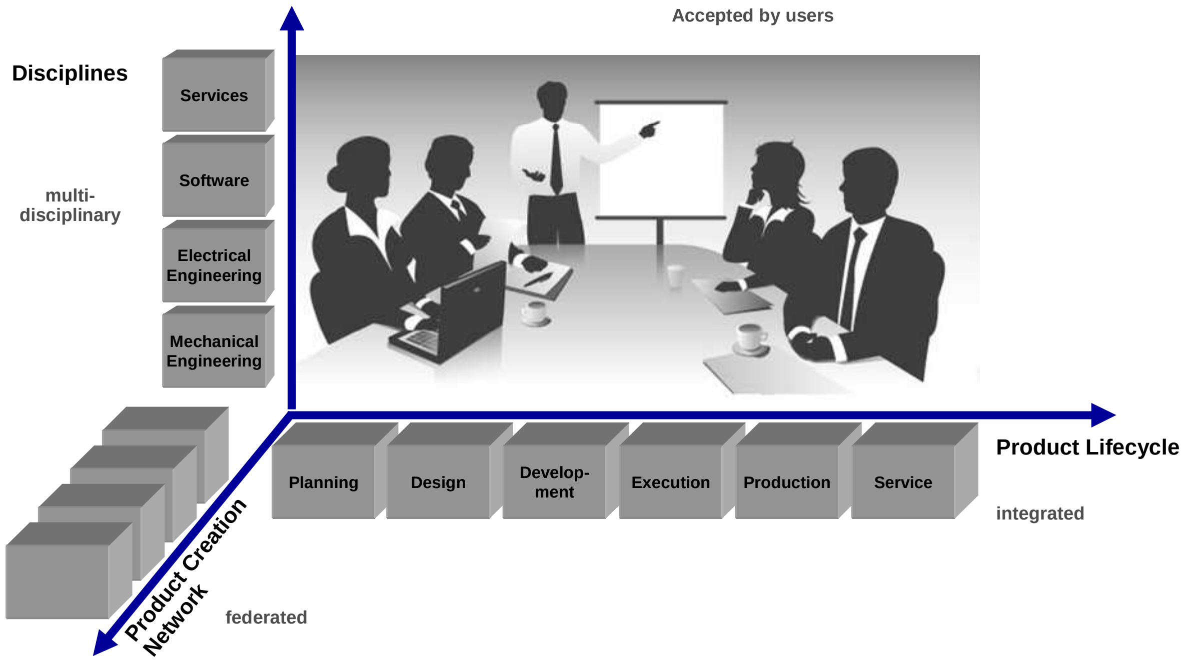


Figure 34: Multi-dimensional collaboration based on Cross Enterprise Engineering

[8] International Conference on Global Software Engineering

Scheer et al. [18] define the term “Collaborative Engineering” and emphasize the importance of collaboration during development phase. Following their statements this approach gets more and more necessary, since suppliers are not only delivering single spare parts any longer, but rather entire system components to the Original Equipment Manufacturer (OEM). But development of whole system components requires very close cooperation with the customer, which is the OEM. This, in turn, makes adequate processes along the value chain indispensable. Figure 35 illustrates the general idea of collaborative engineering.

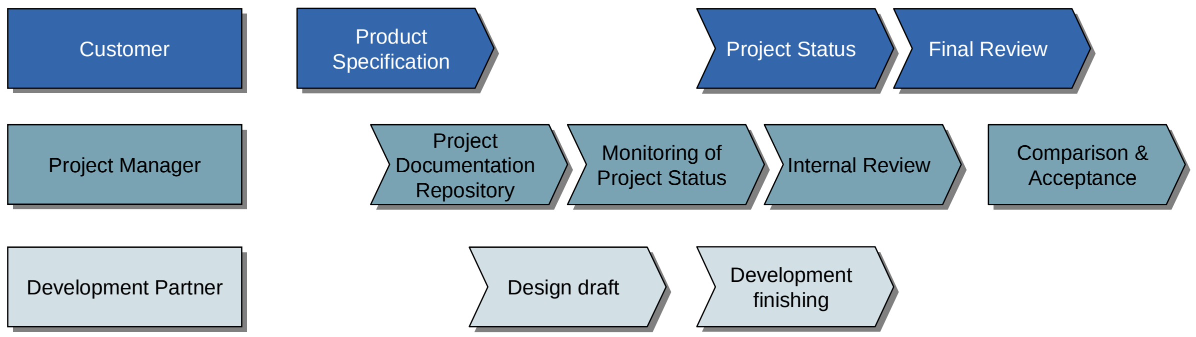


Figure 35: Collaborative Engineering and Project Management

2.5.3 Global Development Processes

Global development is not an approach that happened overnight. This trend emerged in the course of time with the necessity to reduce cost and development time. The basis for changes of the PDP depicted in Figure 36 is the PDP in Figure 12.

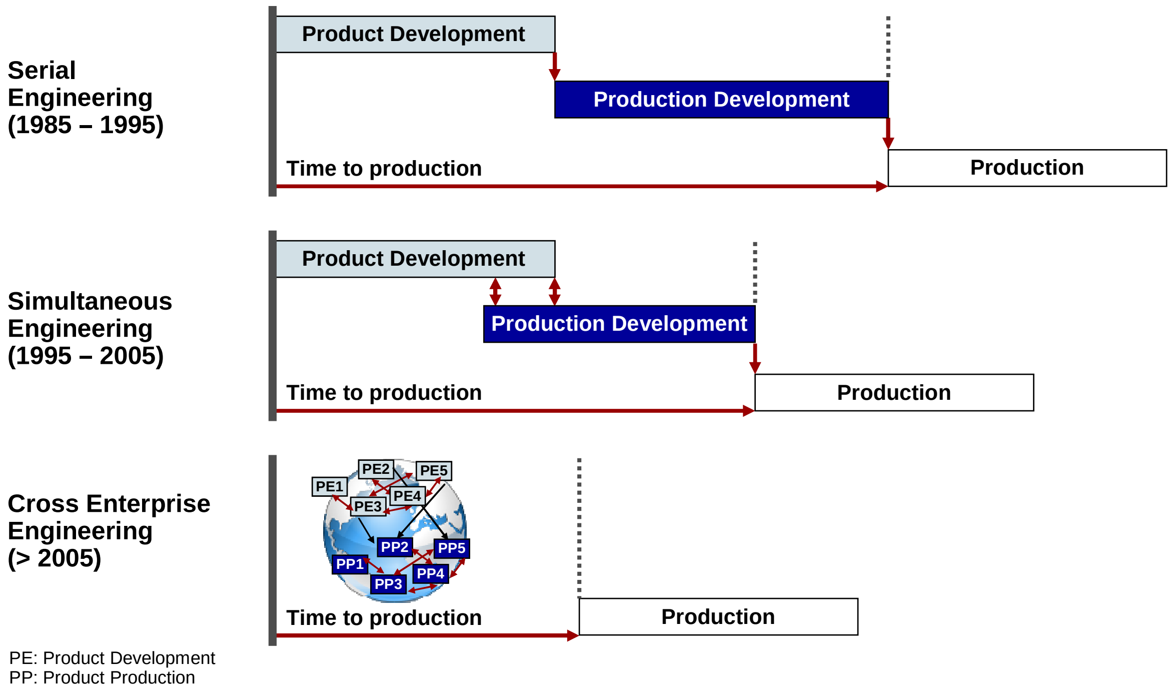


Figure 36: Changes in Product Development Processes [42]

Following the original Serial Engineering approach PDP and process planning for production is executed sequentially. In order to resist the pressure to reduce ‘time-to-market’, the ‘time-to-production' has been shortened over the last decades. Simultaneous Engineering had a little overlap of PDP with production planning; the so-called Cross Enterprise Engineering approach of [42] muddles the two phases almost completely and leads to parallelism of engineering. The parallelism decreases lead time for the price of more complexity, which additionally comes along with globalism.

2.5.3.1 Motivation for Global Development

Although it is very challenging to setup and lead global development projects, many development organizations following this path towards global developments (See Chapter 2.5.1). One major reason for expanding development business globally is cost. Wages and salaries are typically way lower in low-cost countries, e.g., India or China compared to the United States of America or Europe. An empirical study of Lamersdorf et al. [86] shows that cost and access to people are prioritized criteria to go global before other factors (See Table 2). For this survey, twelve practitioners from eleven companies with essential experiences in distributed software development in middle or senior management positions have been interviewed. Participating companies came from many different domains, such as satellite development, educational software, and software services. All of them were medium-sized or large with the smallest company having about 900 employees. Eight out of ten were headquartered in the U.S., one was based in Europe, and one is located in India.

Table 2: Reasons for Initiating Distributed Development

Criterion	Prioritizing Weight
Cost	**9**
Access to people	**9**
Knowledge of Markets	**1**
Required by Customer	**1**
Risk Reduction	**1**
Mergers and Acquisition	**1**

Furthermore, the access to a large pool of trained people motivates especially development organizations to setting up distributed projects. The success of a research and development department (R&D) is mainly driven by the innovative potential of the personnel. However, the desired competency is usually not always available in this country, where it is needed, which makes it beneficial to set-up distributed projects for collaboration [45].

Further motivating factors for conducting projects in distributed environments are [94], [63].

- Increased productivity: This is possible by doing development "following the sun" or to take advantage from hiring highly-educated and trained personnel
- Market proximity: Organizations are able to 'exploit' the proximity to the potential customer, e.g., more accurate derivation of product requirements. The look and feel of a future product is also better adapted and harmonized with local markets.

- Governmental policies and incentives: some governments give incentive to foreign investors if investments are made in their country; those countries are even more attractive for foreign investors.
- Shorter Time-to-market: In some cases Time-to-market and delivery time is shorter since products are not needed to be transported to the country where they are sold.

2.5.3.2 Critical Success Factors of global development projects

Each development projects' success depends on critical factors that are discussed in the following section. Due to its special character, the necessity of fulfilling these success factors is much higher in distributed and global projects than in a collocated development environment. This should not be confused with success factors of PLM that have been discussed in chapter 2.2.2.

Reduce Ambiguity
Ambiguity is still a big issue during development projects in conjunction with partners that are not collocated. Requirement and design specifications are just put over like "over the fence" wondering that some parts have been misunderstood or misinterpreted by other parties. Consequently, developers came up with a running product that typically meets the full requirements, but does not necessarily follow the "spirit" of the specification. This is mainly due to the fact that organizational processes, management and engineering practices and methods, different level of experience and know-how, and in general, the culture differs significantly within the development partner. In projects that are set-up in a distributed environment, it is hardly possible to clarify these ambiguities by just talking to a colleague at the next door. The partners are very often time shifted by eight or more hours, which means that clarifications of problems and questions are only possible by indirect ways of communication, such as e-mail. Direct ways of communications, such as video or telephone conference, are only feasible, if at all, by exceeding the working limit of one partner. For example, for conducting a video conference meeting with colleagues from Beijing, China (time shift to Germany + 8:00 hours), either the Germans need to rise very early or the Chinese need to stay longer in the office.

Maximize Stability
Stabilization in engineering projects is a big issue due to frequent alterations. Requirement often change during project run, project roles are exchanged, and in the worst case even the processes are modified. To address these problems methods like "Agile Development" are introduced to organizations, which work very well if the teams are collocated. Therefore, stabilization is an essential factor within an engineering project that is set-up globally with dispersed teams.

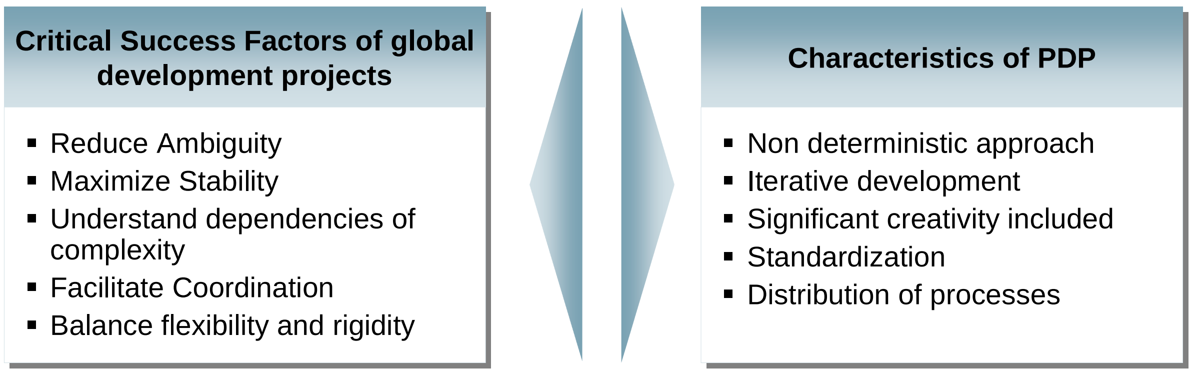
Understand Dependencies of Complexity
The increasing complexity of products results in higher complexity and volume of project tasks. This fact correlates with the dependencies of these project tasks. It is very important to determine the sequence of project tasks and their interdependencies before project starts. This reduces potentially idle time of development teams and increases the chance to develop the right product the first time. Therefore, the communication and information sharing need to be established very accurately and carefully. Therefore, a meeting structure must be defined in a way that all developers get the necessary information by attending only those meetings, which are relevant for their work items assigned.

Facilitate Coordination
Globally distributed projects need facilitation of coordination to a much higher degree than projects conducted in one location. Communication is very often associated with coordination, but there are many more possibilities to coordinate a project. These are besides others processes, management practices or product line architectures. The problem to be solved is to find a balance between overhead and risk. This means that an enormous process framework reduces risk, but increases the overhead tremendously resulting in high internal process costs. On the other hand, fully orienting the coordination on management practices like management by objectives makes the overhead cost of the project going down, but the risk of failure goes up since the project structure loses traceability, controllability, and repeatability.

Balance Flexibility and Rigidity
Finding an adequate balance between flexibility and rigidity is a major challenge for globally distributed development projects. Flexibility of all participating teams is necessary, because each team has different processes, culture, or background with special domain knowledge and organizational practices. Therefore, the process framework should be set-up in a way that allows for smooth adaptation of all these differences in-between the development teams. It is also advantageous if the processes in a global set-up encompass a little of the "agile spirit" instead of insisting on absolutely adherence to processes. However, projects should not follow processes that are ad hoc defined and their fulfillment is hardly controlled. Since we have those differences, in cultures and background, a consequently defined structure is very important for a successful development project, which means on one hand that the customer gets what he really needs. On the other hand, the project should not run out of cost, but rather achieve economic success.

Having the characteristics of a PDP and the critical success factors of projects defined (Table 3) a comparison of these categories bears an area of conflict.

Table 3: Conflict Areas between PDP Characteristics & Projects' Critical Success Factors


Critical Success Factors of global development projects
Reduce Ambiguity
Maximize Stability
Understand dependencies of complexity
Facilitate Coordination
Balance flexibility and rigidity
Characteristics of PDP
Non deterministic approach
Iterative development
Significant creativity included
Standardization
Distribution of processes

This means, e.g., "Maximize Stability" is in contradiction to "Non-deterministic approach". Considering the benefits of PLM in chapter 2.2.2, it turns out that an effective solution to overcome this conflict is the definition of organizational processes and the introduction of PLM. For instance, if PLM reduces Time-to-market, it is implied that that development procedures and process need to have a certain kind of stability, which is, in turn, critical success factors of development projects.

2.5.3.3 Challenges to Global Development Processes

Global oriented projects either exceed challenges or/and suffer from additional or other challenges. Studies have shown that a global project set-up takes about 2.5 times longer to coordinate than a local one. The major reasons are the following ones [76], [18], [106], [41]:

Complexity of Project
Due to its nature, globally defined projects are more complex than local ones. Since those projects have many different organizations with various participants that might never may have met before. Therefore, the number of interfaces, especially the first-time relationships of project members increases exponentially. This makes a smart and structured approach absolutely necessary to be able to coordinate a cross-organizational teaming set-up, manage evolution, and monitor the progress of development.

Different Cultures
Different countries have different cultures, which also leads to problems in global development projects. For instance, in specific countries team members are not used to take part in telephone conferences and rather prefer to email the questions and issues. In case organizations are not aware of that problem, this practice waste much time until every participant has exchanged this information the way he is used to communicating.

Another example concerns the small talk at the beginning of a call. Some cultures find it rude directly coming to the point without having a little talk about something complete different, e.g., some ‘private’ topics. In turn, it might be fairly annoying or even frustrating for the other party, if their counterparts never come to the point directly.

Different development process
Globally dispersed teams that have been just “assembled” from different organizations have typically a different culture in place of how the product develops. Each of them has internalized a certain type of process that is common for them. A global project set-up has to handle that issue by incorporating all those different types of processes and approaches towards an integrated PDP. This integrated PDP is necessary to have all resources working in the same directions with on major goal, e.g., common understanding of product requirements, usage of methods that are compatible to each other, commitment to time schedules of the project.

Furthermore, as shown in section 2.1 ‘Organizational Business Processes’, an organization can realize cost reductions by process optimizations, which, in turn, increase efficiency significantly. If organizations move to low-cost countries for further cost reductions, it is crucial to also optimize the global processes. This prevents the risk that efficiency enhancements through, e.g., lower salaries are undone by just using inefficient global processes.

Varying Knowledge and Infrastructure
One reason why global development projects are set-up is the lack of knowledge needed for development of future products. However, this advantage grows to a tremendous problem when organizations try to plug in special knowledge from specialized resources from anywhere in the world into their own organizations. The reason for this issue is that other project members also get in touch with the new knowledge and vice versa, which leads to integration issues immediately. Furthermore, other resources of different experience and knowledge in terms of, e.g., methods, tool, or models, which also coin developed products.

This might influence final integration of sub-products. The alignment of knowledge means to spend big effort on training, delegations, or to extend travelling to bring the experts together.

Logistic Problems
Daily project communication is mainly dependent on infrastructure with dispersed teams. This means that network resources and infrastructure needs to be planned and set up. If this is not done very thoroughly, the daily work in global projects runs inefficiently. This is caused by a very slow network connectivity, which is consequently overloaded and breaks down after a while, which automatically leads to unplanned downtime of the whole project. This might also concern telephones since telephone nowadays also works with a data line over the internet (“Voice over IP”).

Communication Issues
It is also quite obvious that in a globally dispersed project set up, communication from management to the workforce is anticipated to go rather slowly and with a high portion of fuzziness. If one imagines that the manager is located 5.000 miles away from him, a sound communication strategy is crucial in order to reach every employee and team member, no matter where in the world he is actually working. Only this approach guarantees high transparency in decision-making processes, which in turn makes management again reliable and accepted in the whole project and company.

2.5.4 Cooperative Development Models

The core of this dissertation is based on the studies of Xitong Li et al. [86] who developed patterns to document and protocol web service composition. This work basically deals with the management and control of messages that are transferred between entities (i.e. ‘sender’ and ‘receiver’) to realize protocol mediation in the field of Service-oriented Architecture (SoA). The idea of mediator patterns has been modified for this work insofar that the mediator realizes the connection points of diverse organizational processes. More about process frameworks and patterns can be found in [27] and [144].

Meyer B. defined a development model that differs significantly from conventional models [93]. This model creates coarse software architecture of the entire software system which is then divided in sub-projects representing components of a system. These components are developed by small development teams. Figure 37 shows that Meyer also aims for a sequential development process, however, not for the whole system, but only for sub-systems or components as mentioned before. Those projects might run simultaneous, i.e., various project goals are achieved at various points of times. After development of a component (sub-project) a review takes place that identifies those components that are potentially subject to re-use. These components are provided to the overall development process. The approach generates a parallel process or at least one with little overlaps, which, in turn, requires an intensive amount of coordination and communication.

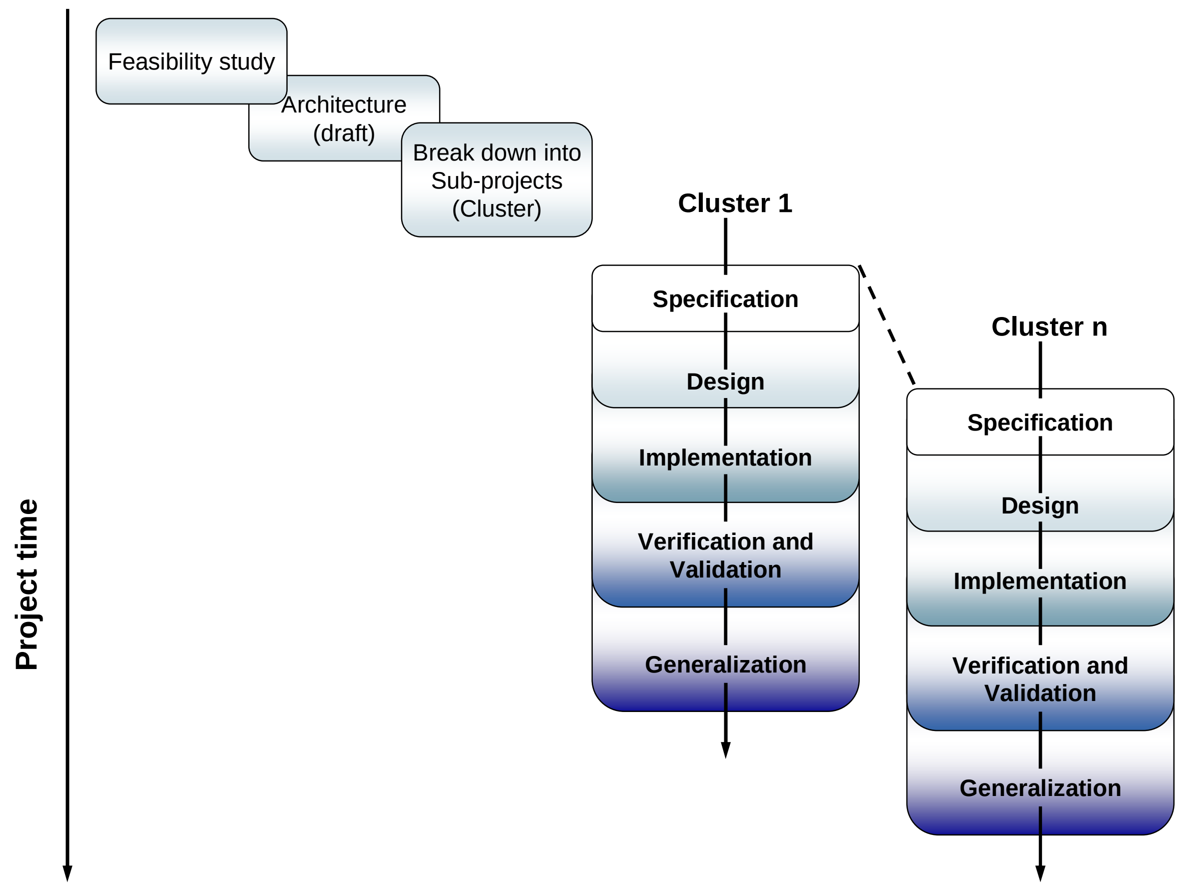


Figure 37: Cluster model of Meyer B. [93]

Cooperative development in software engineering with a major focus on processes is an essential topic of Altmann and Pomberger [1]. In their work, they emphasize the importance of processes and challenges of global project set-ups with dispersed team. Those challenges of software projects encompass:

- Increasing complexity
- Non- formalized processes that cannot be automated
- High risk potential
- High documentation effort
- High communication necessity

Due to the importance of processes, they have developed a model for cooperative software development processes. This model, which is actually a Meta model, consists of two parts: the product view and the predominant process view (Figure 38). The definition of these views is based on the studies of Floyd/Züllighoven who also distinguish product and process-related activities [44].

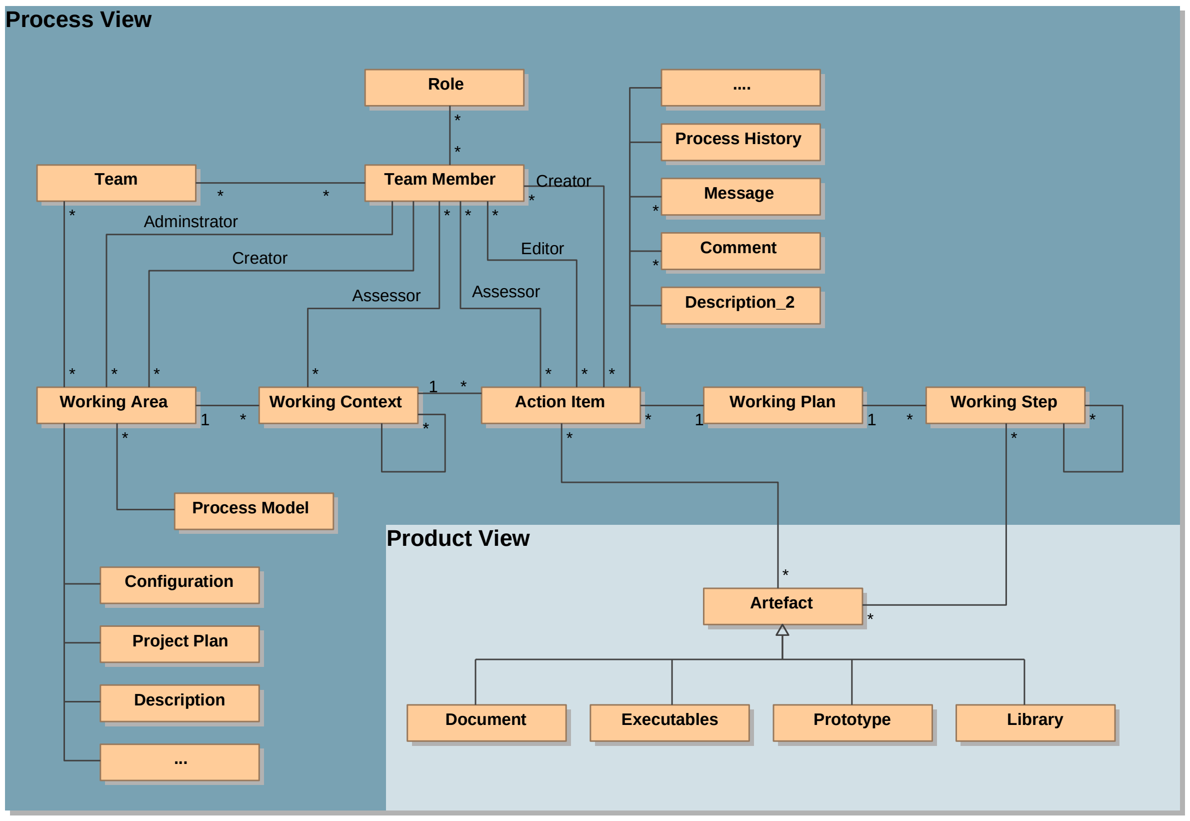


Figure 38: Model for cooperative software development processes

Product-related activities focus on requirement engineering and system development resulting in the final software product. This software product consists of prototypes, executable code, and documentation. In their approach, the product development process can be divided in different phases that require pre-defined work results or artifacts respectively.

Process-related activities concern coordination and cooperation of the product development process and encompass product administration, quality assurance, and project coordination, which is basically project management. For the purpose of product administration baselines for all artifacts are defined. These baselines are references for product status in a collaborative project. The status are synchronized and used for the next development iteration.

V-Model XT® is another development model and *the* standard model of the Federal Republic of Germany and guideline for planning and conduction of development projects [85]. It is related to this work insofar that it brings in a sub-order relationship as a special characteristic [79]. This relationship enables every customer in major project to conclude one or more sub-contracts with any contractor necessary. This allows for more flexibility of the customer to get special features that require special knowledge implemented even on short notice.

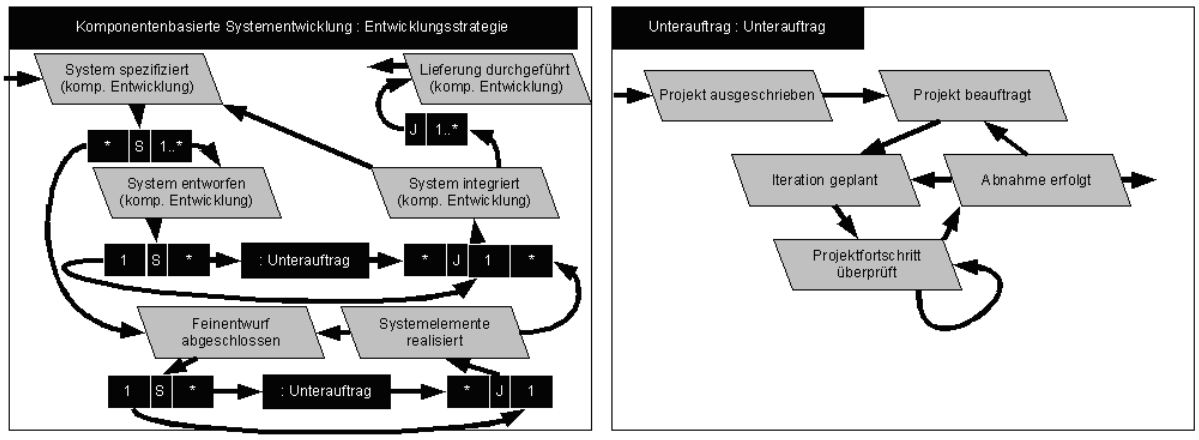

Komponentenbasierte Systementwicklung : Entwicklungsstrategie
System spezifiziert (komp. Entwicklung)
Lieferung durchgeführt (komp. Entwicklung)
System entworfen (komp. Entwicklung)
System integriert (komp. Entwicklung)
: Unterauftrag
Feinentwurf abgeschlossen
Systemelemente realisiert
: Unterauftrag
Unterauftrag : Unterauftrag
Projekt ausgeschrieben
Projekt beauftragt
Iteration geplant
Abnahme erfolgt
Projektfortschritt überprüft

Figure 39: Illustration of a sub-order relationship ("Unterauftrag") in V-Model XT

Furthermore, model aspects in global development projects has been intensively discussed and evaluated by Prikladnicki. He has defined a capability model in a global collaborative context. This model captures patterns of evolution in the practice of distributed software development in internal offshoring projects [106].

In [106] Prikladnicki and Audy have defined a set of criteria to define geographically distributed environments or scenarios, respectively. Based on these scenarios, a general model is defined, which includes several contribution level of an organization in a distributed software environment.

Furthermore, Prikladnicki comes up with a reference model for global software engineering with a detail discussion about factors that enable multinationals corporations to operate successfully across geographic and cultural boundaries [106].

In [86] a multi-criteria based Development Distribution Model for making decision on global development projects is illustrated. This model does not only consider cost as a decision criteria for setting up global projects but takes into account parameter like workforce capabilities, innovation potential of different regions, or cultural factors.

Sooraj and Pratap consider coordination aspects and problem in [133]. They use an inter-sited coordination index for overcoming coordination and communication problems using simulation.

2.6 General Empirical Studies

A major empirical study has been conducted in this field of research by A. Avritzer et al. in the Global Studio Project (GSP) at Siemens Corporate Research [7]. This experimental research project has been set up at six universities with about 30 developers in five countries and four continents. The core consists of one central team responsible for important upfront work, e.g., architecture definition and several remote teams that were filling in the gaps. The GSP encompass the following principles:

- Hybrid centralized/distributed management, i.e. different places for execution of various development domain, e.g., software processes are developed by the central team; testing is done by a remote team (compare Figure 40)
- Iterative development using a two week iteration cycle

- Minimization of cross team communication; this was managed by the central team
- Formalism of documentation: The project followed the principle that the higher the distance of sites the more formalism is necessary to execute process domains. For instance, the Requirement Specification has been done with less formalism due to the availability of domain experts in the central team. This minimizes communication for requirement clarification. The Test Specification, however, was created by a remote team and, therefore, needed more abundant documentation.

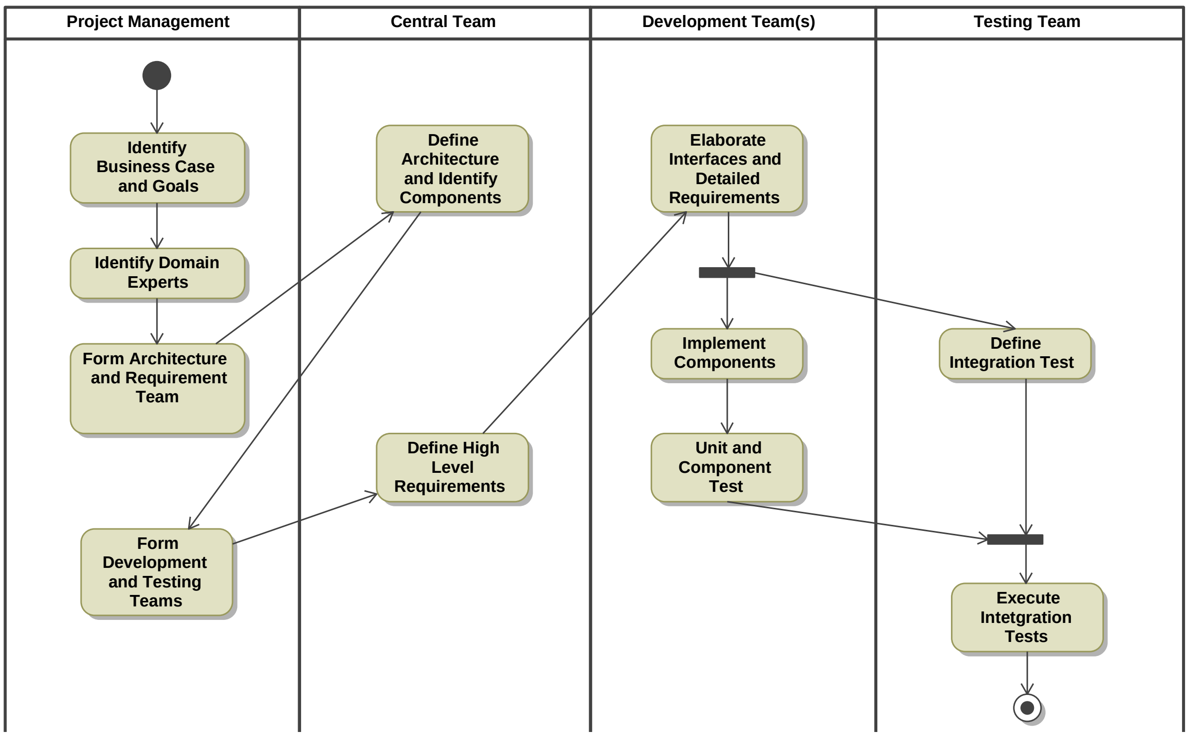
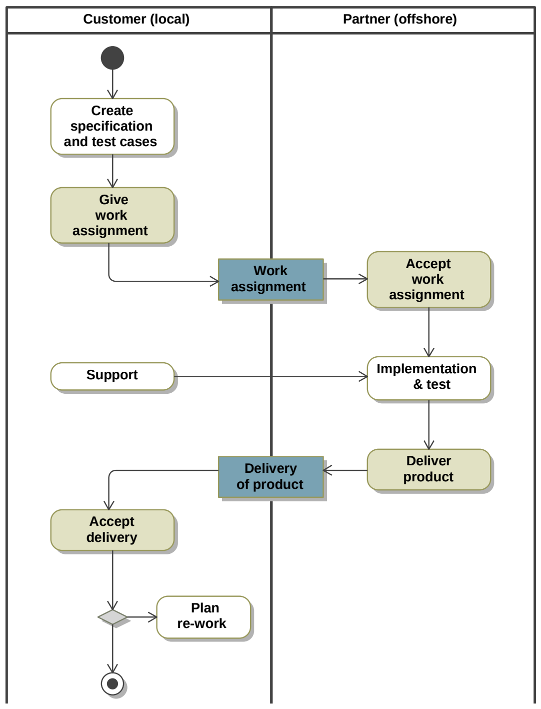

Project Management
Central Team
Development Team(s)
Testing Team
Identify Business Case and Goals
Identify Domain Experts
Form Architecture and Requirement Team
Form Development and Testing Teams
Define Architecture and Identify Components
Define High Level Requirements
Elaborate Interfaces and Detailed Requirements
Implement Components
Unit and Component Test
Define Integration Test
Execute Intetgration Tests

Figure 40: GSP Version 3.0 Process

Wichmann describes his personal experience about offshore collaboration in [145]. This report is from the mechanical engineering industry and related to software engineering. He provides an activity diagram using the UML that shows the responsibilities of the project partners at the local site and the offshore site (Figure 41). Wichmann does not describe typical challenges of a dispersed development project; moreover, he defines criteria that product sub-products have to meet to be selected for development offshore. Those criteria encompass, e.g., low communication effort with experts, defined requirements of the component, a very sound cost-benefit-relationship, offshore development of those components without core competencies requirements etc.

For more experience, reports and in- depth information, please consider [62], [94], [62], [19].



Discipline	Local Team	Offshore Team
Requirements	local	
Offshore specification	local	
Analysis and Design	local	
Implementation & Test *- core modules* *- related modules*	local	
Implementation & Test *- based on related modules*		offshore
System test		offshore
Deployment	local	
Configuration- and change management	local	offshore
Environment	local	

Figure 41: Offshore activities (coarse-grained)

2.7 Cultural Aspects

Collaboration in international context is not only driven by technical and process aspects. Often, the culture plays an essential role and decides whether or not a project is successful. Therefore, culture is an essential challenge in global development projects, which is absolutely necessary to be considered. The culture of organizations encompasses the way they act and behave in daily life and how they react to certain stimulus from outside. Organizational leaders are typically not comprehensively aware of the power of any culture around the world. Culture can be responsible for success or failure of a business.

In order to analyze this phenomenon in the context of business negotiation, Hofstede evaluated 116.000 questionnaires from a multinational data base at IBM. The data covered 64 countries and up to 53 different cultures worldwide. Hofstede defined five dimensions of culture and categorized the 53 cultures into these dimensions for validation purposes [62].

Another cultural model with even seven dimensions was developed by Trompenaar as a result of an empirical study. For more details on Trompenaar's work please refer to [139].

These cultural evaluations of Hofstede and Trompenaar describe tendencies and not characteristics or even categorizations of individuals! Consequently, a country's scores should not be interpreted as deterministic.

However, it is very interesting to know upfront if organizations are willing to collaborate with other organization from other countries. The leaders and decision makers should be aware of what dimensions of culture exist and what the tendency or collaborations of any country prefers. Referring to 'Power Distance' [9] it is, for instance, crucial to know, whether an

[9] Power distance is the extent to which the less powerful members of organizations and institutions (like the family) accept and expect that power is distributed unequally [62]

individual should talk to a superior or to an individual in order to effectuate anything. For further in-depth discussions on cultural behavior, please refer to [106], [62], [139].

Culture in the context of global engineering has often been analyzed by several and various authors. A great many case studies have been conducted in this field of research.

Boden et al. analyze culture and knowledge management and illustrate by means of some realistic cases that knowledge management is an important bridge between people and cultures [19].

Casey explores the difference of cultures in a multinational organization within a three year case study where people from Ireland and Malaysia were taking part in projects. He shows that “improvement of cultures” could have the opposite effect if done too rapidly [28].

Al-Ani and Redmiles were talking about trust in dispersed development team. They come to the conclusion that trust is rather an issue in large and diverse development teams. In such team constellations, trust is higher among authoritative team member, e.g., team leader [12].

Further detailed studies are documented in [53], [106], [40], [59], [94], [40], [67].

2.8 Communication in Collaborations

Communication is seen as one of the most important aspects in collaborations with worldwide distributed teams. This is documented in work from authors all over the world. The dependencies of communication and distance have been intensively discussed by Herbsleb in [62], [30]. He argues that communication of modification requests take about 2.5 times longer in a cross site environment than in a common location. There is also evidence that colleagues that are not located same-site feel less ‘teamness’ than others. This makes it very difficult to help and assist in times of high workload.

Prikladnicki’s work [106] also considers the ‘teamness’ factor by exploring the phenomenon of perceived proximity of team members in collaborative projects. This means that even if teams are physically very close to each other, they might be very far from an individual’s perception point of view. Therefore, the *perceived proximity* (Figure 42) of team members is crucial [148]. Quadrants 1 and 4 define a high perceived proximity. However, the perceived proximity can also be low, although collaborating teams are co-located (Number 3).

Serce F.C et al. provide an empirical study with 152 participating students from Panama, Turkey, United States (U.S.), and United Kingdom (UK) on communication behavior in a globally-conducted software development project [133]. For this purpose, types of communication behaviors that occur when student teams are engaged in a software development project are defined and evaluated. The study concludes with the result that communication correlates with task type, culture, and GPA[10]. Those teams with above average communication behavior were outperforming other teams. More aspect about communication in global collaborations can be found in [3].

In the last decade "Chat“ or “Instant Messaging” as a preferred communication medium became more and more important in development organizations. Following [62] each developer spends at least 75 minutes for communication through this “modern” media. This aspect is also discussed in [62].

[10] Grade Point Average (GPA)

Further experience on instant messaging in global software development projects are discussed in [99].

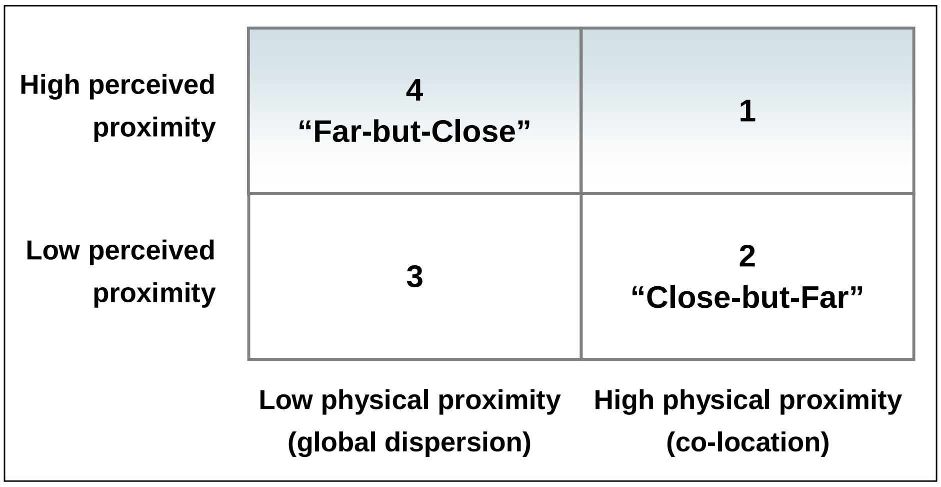


Figure 42: The paradox of perceived proximity

2.9 Process Models and Globalization – tying it all together

As stated in the previous sections, organizations have process models and approaches defined that support development departments during project execution. Processes are a necessary and helpful mean, especially for projects with high complexity, i.e., a high number of parties involved, variety of domains contributing to a project, or high number of software lines of code to be developed for the final product. Typically, these processes are defined locally including, at most, all domestic partners.

The trend towards globally distributed development arises with a couple of additional issues that organizations have to handle: teams that are dispersed all over the world hamper communication in projects and, moreover, miscommunication and misunderstandings due to differences in language and culture occur. Additionally, every organization has its specific defined processes in place, which are more or less comprehensively defined. This definition follows the culture of the respective organization, which also depends on the resident country of a collaborating organization. The more sub-processes of a project are distributed all over the world, the more communications difficulties.

Changing the process of organizations in order to better fit into a distributed collaborative project would require the respective organizational culture of the supplier to change. However, this, in turn, negatively influences organizational performance during the project, if resources have to follow an unaccustomed way of working.

In order to resolve this dilemma, processes need to be capable for global collaborations. This means that they require well defined and integrated process descriptions from each of the participating parties. These aspects contribute significantly to the problem definition and to the solution of this dissertation’s approach.

3 The Approach from a Bird's eye view

This section provides an overview of the actual problems and issues that make the subsequent solution necessary. In addition, the basic solution concept is introduced.

As stated before, new trends and desires from customers enforce organizations to drive innovation tremendously towards attractive products. This requires special knowledge, which is sometimes not available in development departments, which compels those organizations to investigate the world-wide resources know-how-pool and hire those competencies desired for the next development project. This openness to include employees or even organizations from all over the world into development projects enforces organizations to set up global and distributed processes, since not every needed resource is willing to move. This makes collaborative and distributed development projects special in a way that the personnel working on a distributed project cannot meet easily in person as it is possible in projects that are run in one location. For this reason, special process set-ups that adequately support such a distributed environment need to be defined.

3.1 Application of the Process Integration Approach

It is assumed that two or more distributed organizations want to run a project together. For the purpose of better illustration, we consider three organizations. These organizations have organizational processes in place as depicted in Figure 43. Thereby, Organizations A and B have processes defined on "System Development" – level; whereas, Organization C has its processes defined on "Software Development" – level.

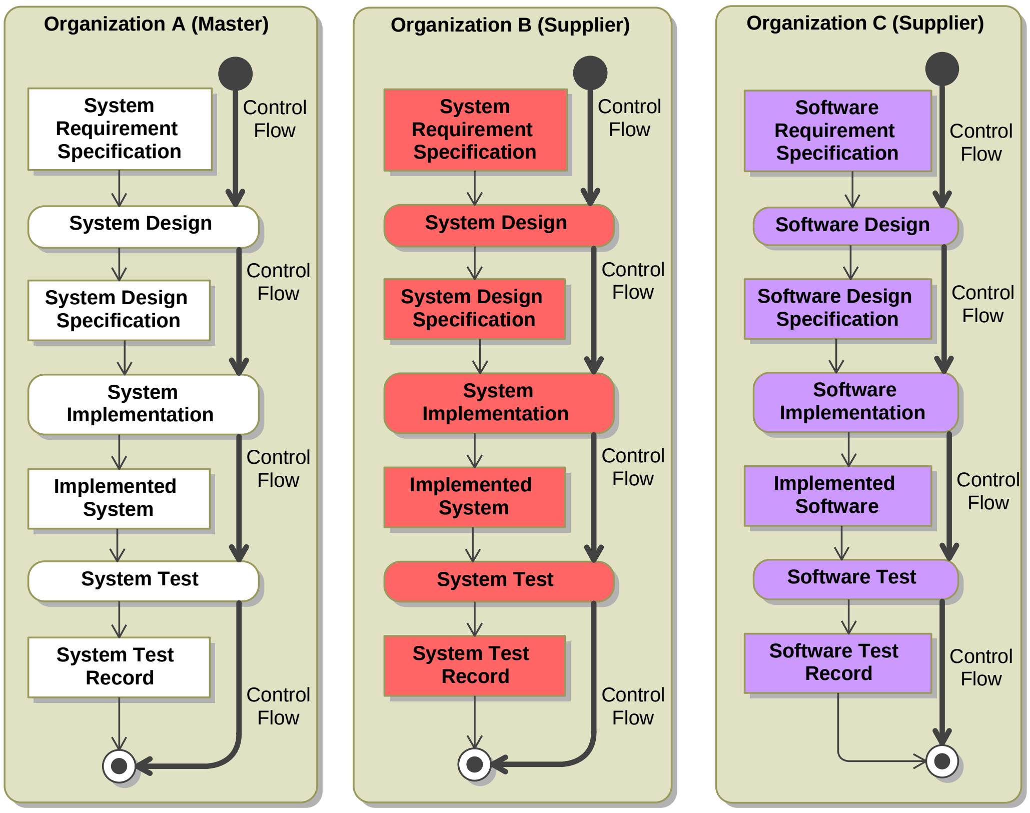


Figure 43: Initial Processes of Organizations A, B, and C

For the purpose of collaboration, respective process owners and responsible process engineers (See Chapter 1.4) have firstly to clarify who will be the lead organization of the entire collaborative project. This is necessary since the lead organization's process (in this case Organization A (Master) in Figure 43) is the one that integrates sub-processes or activities of other organizations. The depicted processes have that label on the very top of the activity.

3.2 Major Challenges on Process Integration

If there will be a collaboration project between two or more parties, the master organization – as the driver of the project – typically wants to integrate some activities from other organizations into its own process; thereby, some activities are done exclusively by one organization, others in parallel or conjointly. However, the integration or the change of processes always occurs with issues outlined in the following.

Proneness to processes
Development organizations typically want to keep their own processes as far as possible since the established processes are kind of customized to the organization. Ad hoc process changes, which might be unavoidable in process-based collaborations, would rather evoke uncertainty and confusedness within the workforce of any organization. This reduces the acceptance of the process and therefore the productivity of the development department. The role of the Master is detailly described in chapter 3.3.3.1.

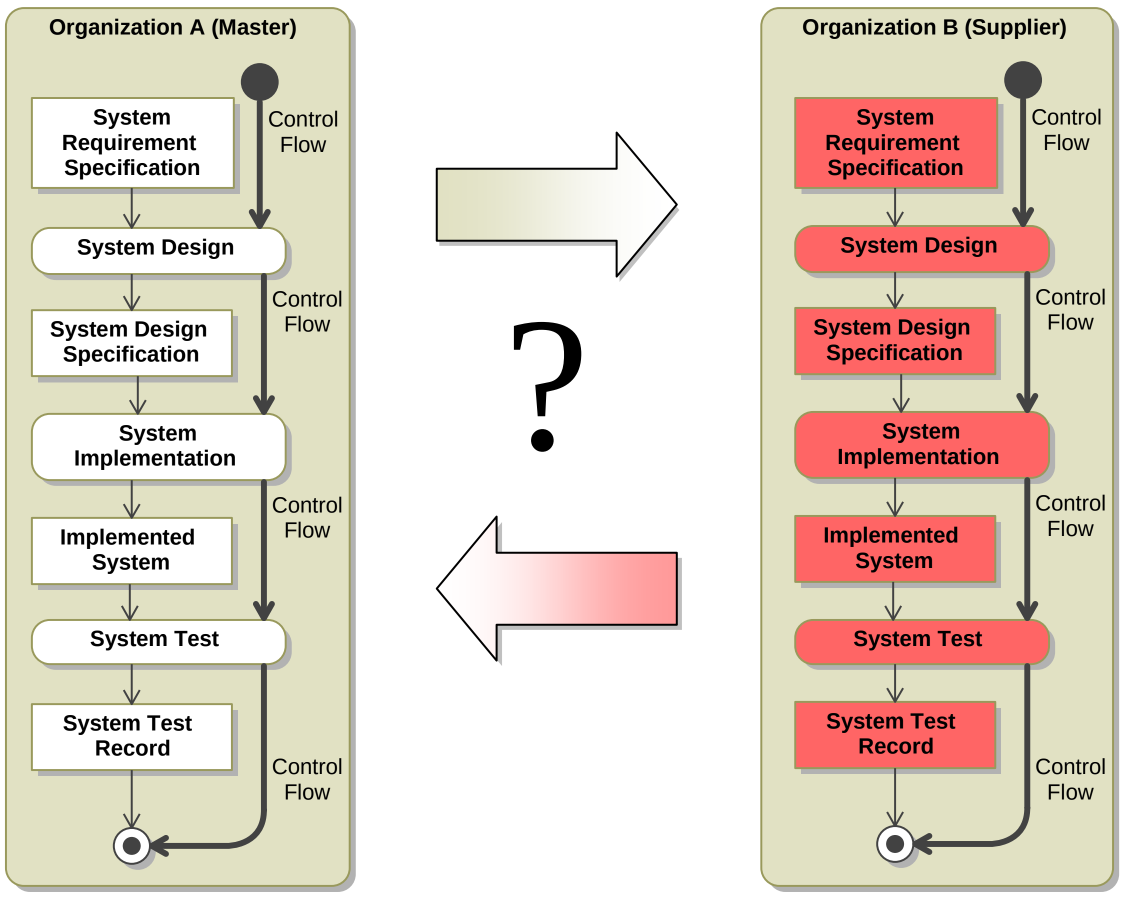


Figure 44: Integration challenge of semantically different processes

Semantic difference of processes
Each organization has usually their proprietary processes defined containing own process descriptions, artifacts, roles etc. Even if activities have the same label, e.g., *System Implementation* as shown in Figure 44 of both Organization A and B, they are typically semantically different. This means that *System Implementation* of Organization A is not the same as the one of Organization B differing, for instance, in artifacts, methods, or roles used during execution of this activity.

Role concept
Project constellations with more than one project partner often lack of clear definition of roles and responsibilities. This results in work delays since action items are rather pushed away in order to minimize the individual workload.

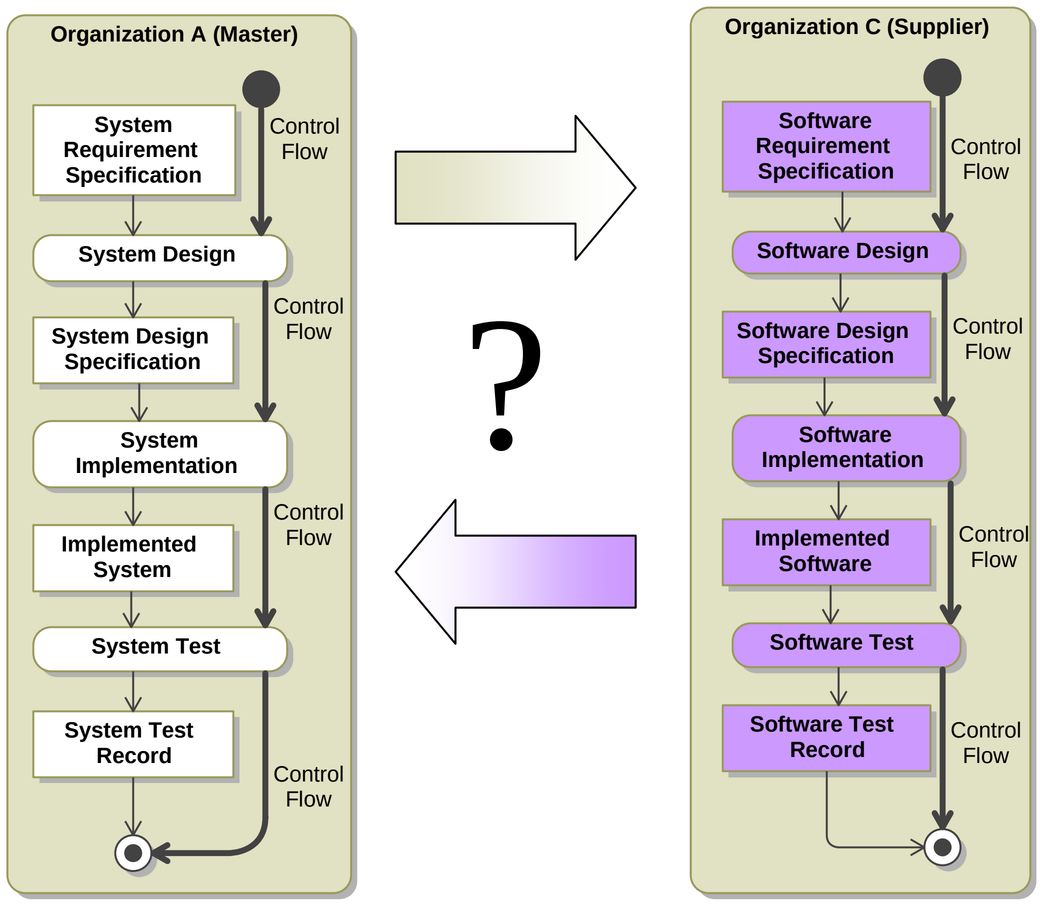


Figure 45: Integration challenge of processes granularity levels

Granularity of processes
Further problems originate from the granularity of processes defined by each organization. As depicted in Figure 44, Organization A and B have processes defined on a system implementation level, which might be adequate for its product development and business. In contrast to Organization A, Organization C has only its software development process in place (Figure 45). Compared to Organization A and B, the process of Organization C (for a collaboration scenario) is documented on a finer grained level (software level vs. system level).

In order to solve all these issues, the challenge for process management group is the clarification of what exactly happens during integration of other organization's activities or processes. This problem is symbolized with a question mark in Figure 44 and Figure 45.

3.3 Solution Concept for Process Integration

The following chapter briefly sketches the solution idea of the process integration approach. This is necessary since a global development project encounters many challenges that need to be handled, e.g. project complexity, communication issues, logistic problems, different cultures etc. (See chapter 2.5.3.3).

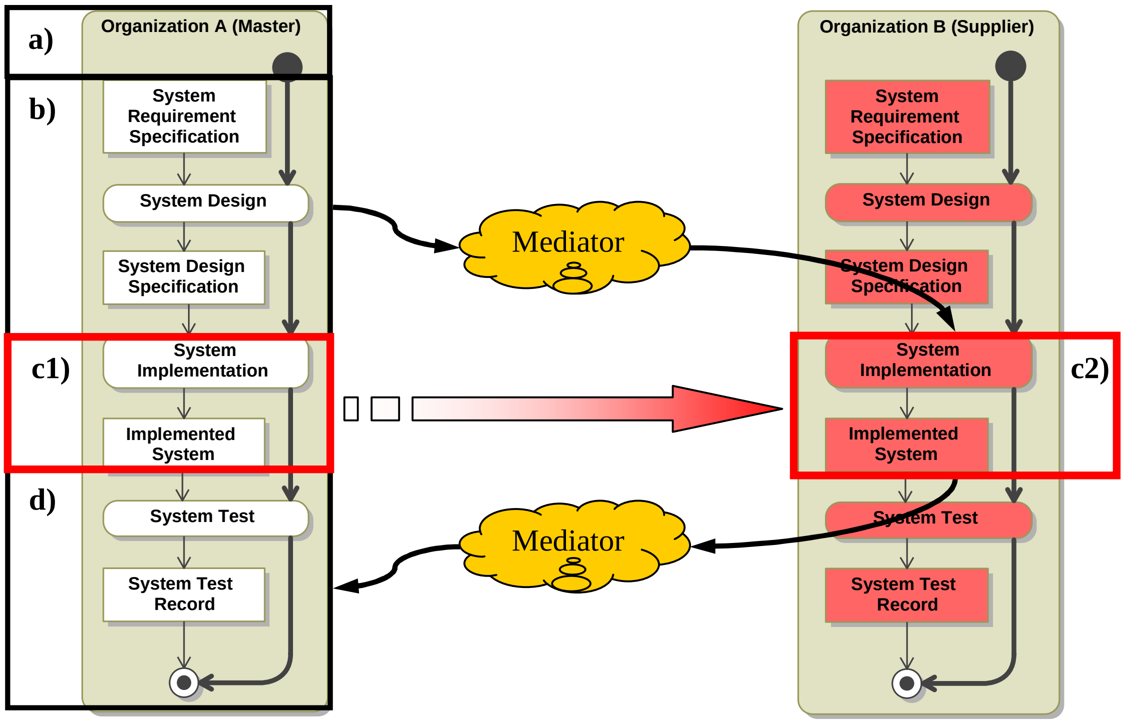


Figure 46: Mediator Introduction for Process Integration

First, all collaborating parties have to identify those sub-processes are relevant for collaboration, which is important to avoid double or even redundant work. Figure 46 marks those sub-processes (red rectangles) that will be integrated into the resulting collaborative process. Organization A (Master) decides to integrate System Implementation (=c2) and the resulting artifact of Organization B (Supplier). It is also crucial to define clear responsibilities prior to process and/or project execution to deliver specified artifacts, e.g., in Figure 46 the Implemented S*ystem* is provided by Organization B. However, is Organization B also exclusively responsible for that artifact within the collaborative process? Therefore, a role concept to be defined needs to clarify such questions.

Second, the identified sub-processes and activities need to be connected. The connecting arrows are depicted in Figure 46. The process flow in this case would then be: "a – b – c2 – d". As already mentioned, processes are typically different from a semantics perspective, this means *System Implementation* of Organization A (=c1) and B (=c2) have the same label, but the meaning is different in terms of used methodologies, resulting artifacts, etc. This means that processes cannot be just connected. Therefore, an "interface" is necessary that handles semantic differences between processes by explicitly indicating interface issues and inconsistencies and, thereby, providing a resolving solution.

In order to handle this issue a connector is introduced, which is called '*Mediator*'. Generally, a mediator is an interface to connect cross-organizational processes. In Figure 46, Organization A wants to outsource *System Implementation* (=c1) to Organization B (=c2).

Towards a common understanding, the terms **Mediator** and **Scenario** are defined in chapter 1.4 (Definitions and Terms).

3.3.1 Mediator Pattern

As described, a "mediator" is actually a "mediator pattern" in terms of an interface that consists of several different actions. An exemplary mediator pattern is shown in Figure 51, which illustrates the auxiliary function showing how to bring together two or more semantically different processes.

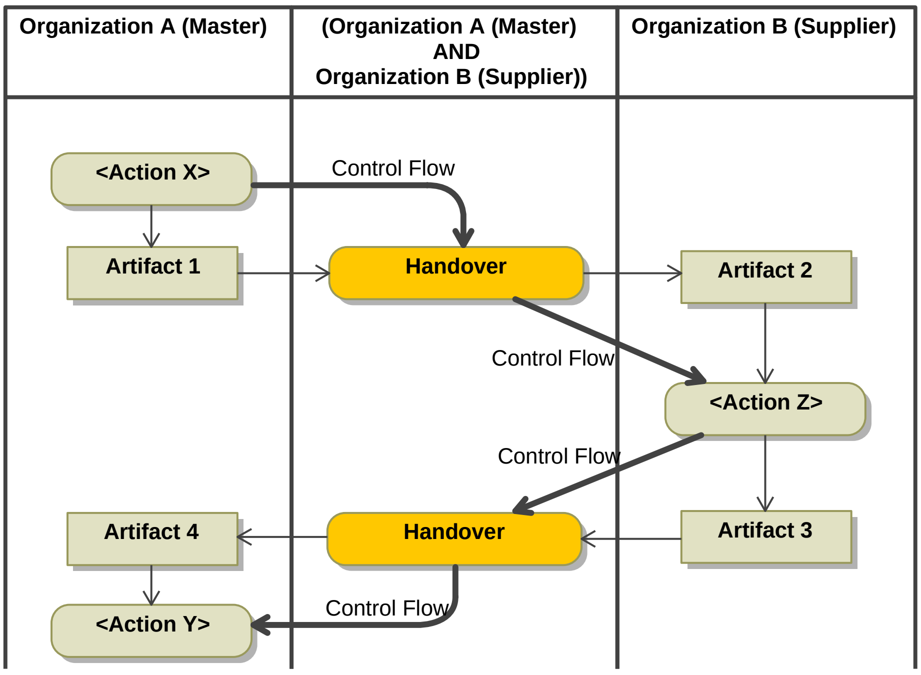


Figure 47: Exemplary mediator pattern for process integration

The mediator's purpose is to guide the procedure of how to connect processes and workflows before collaboration starts. Due to the help of these auxiliary functions it is possible to exchange either single actions/activities or even whole action chains from any process. Therefore, the mediators look differently, depending on the scenario applied.

These mediators are modeled with UML activity diagram terminology (swim lane diagram), but do not necessarily follow the exact UML modeling conventions [100], since they have only an auxiliary character (e.g. missing start and end nodes). A mediator typically consists of at least three swim lanes, one for each Organization A and B and another swim lane for interfacing organizations. Furthermore, it contains the notation for control flow and object flow. Mediators with only one swim lane will be explained as special case in chapter 4.2.5.

As shown in Figure 47, additional actions ("*Handover*") are necessary to get two or more processes connected. The *Handover* basically deals with the issue how to convert e.g. *Artifact 1* (in Figure 47) as output of <Action X> into *Artifact 2*, which is input for <Action Z>. Therefore, these activities are the crucial interface of Master and Supplier(s) to discuss and consecutively define details about the handover of artifacts from Master to Supplier(s) and backwards.

This addresses especially:

Documents:
There is clarification necessary whether or not the Master provides all relevant documents the Supplier needs to successfully conduct its relevant development portion. Furthermore, resulting documents/artifacts shall be handed over in a defined format, e.g. source code, specification etc. (The artifact handling will be described in detail in chapter 3.3.2).

Development method:
It needs to be clarified whether or not the Supplier is able to follow a desired development model (e.g. iterative or agile development). This might be necessary to fulfill quality restriction to get an admission or license for desired market, e.g., FDA [11] conform development of product like a computer tomography for the U.S. market.

Knowledge:
The Supplier has to state whether or not enough and adequate know-how is available in his organization to accomplish desired tasks successfully.

3.3.2 Artifact Synchronization and Handover Concept

During synchronization or hand-over activities organization has to compare and agree upon desired artifacts that need to be delivered from the master organization to supplier organization and vice versa. This is typically a very complex procedure organizations have to walk through, since considered processes are semantically not equivalent, which also concerns the corresponding artifacts. Moreover, this is the "crux of the matter" that decides on the successfulness of any collaboration. Therefore, it is necessary to give both process and project manager hand-on support on what artifacts are typically exchanged during handover or synchronization activities. However, this work will not provide a set of artifacts that must be provided at any process step during development.

3.3.2.1 Number and Types of Artifacts

In order to show the methodology and functionality of the handover process, Figure 48 provides a more detailed view into the handover activity. As depicted in the mediator pattern in Figure 47, one single artifact was given as "the" input for the *Handover*. This artifact represents a potential variety of artifacts, which is handed over in a software development process. Within the handover activity, respective parties decide on what artifacts are necessary for the next upcoming process step; there might be not only one single artifact, but a potential variety of outputs needed to proceed further development steps. This means, in turn, that the handover activity needs to check whether or not content of existing artifacts from the prior organization ("Organization A") does meet the minimum needs of the

[11] FDA = Food and Drug Administration of the United States of America

following organization ("Organization B"), which takes those artifacts as input. This resolution of differences between input and output artifacts is furthermore an essential part of the collaboration. The following section illustrates the handover scenarios by using one representative document for all potential artifacts.

1. Organization A provides exactly the input Organization B needs

As depicted in Figure 48, input and output artifacts are the same, which is symbolized by both the identical number of sections that are incorporated into the artifact and the identical color code that symbolizes identical content. In this case there is no interface problem since the inputs correspond with the outputs.

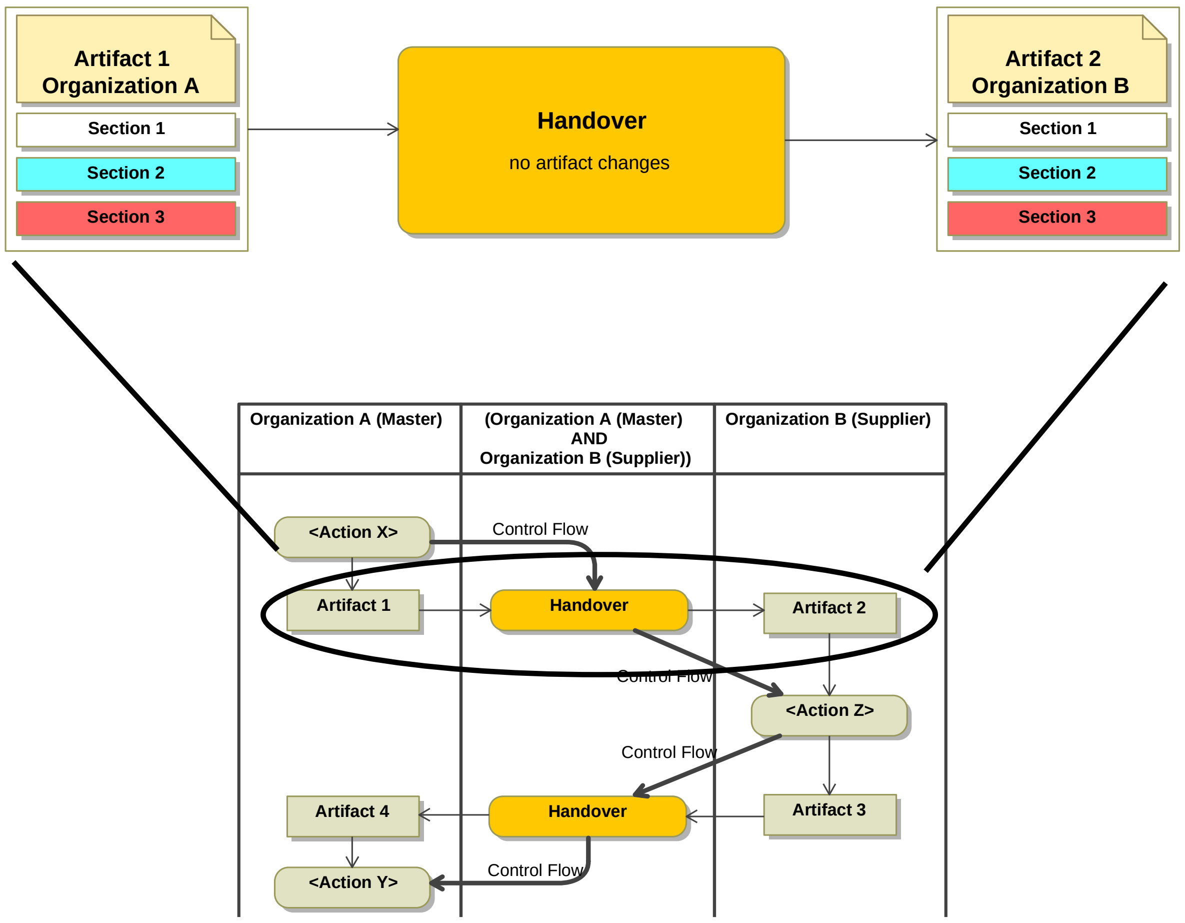


Figure 48: Artifact Handover in Detail: corresponding input and output artifacts

2. Organization A provides less content than Organization B needs

In this case the process to be integrated requires additional and different content ("sections") than the input artifacts are able to provide. As illustrated in Figure 49, Organizations B needs an additional "Section 4" in the artifact handed over. If these issues already pop up during process definition, process manager shall drive the process definition towards a solution. These insufficiencies are identified latest during the handover/ synchronization meeting when all relevant parties conduct the respective handover/synchronization activity.

In any case, it has to be decided whether

a. Organization A have to provide the missing content (the gap between output from A and necessary input for B)

This situation obviously occurs when Organization A, for instance, just misses any artifacts to create during development

b. Organization B has to provide the missing content (the gap between output from A and necessary input for B)

This often occurs if Organization B follows a higher process capability than Organization A. This means that Organization B requires, e.g., lots of statistics for statistical process control in order to address their process needs. If Organization A does not use any statistical process control for managing projects, it is by nature unable to provide any of these statistics and artifacts to adequately support the software provider's process requirements.

c. The parties accept the deficiency between out- and input and continue executing the collaborative process. In this case, no further action would be necessary. This means that Organization B omits its own process requirements in terms of statistical process control etc.

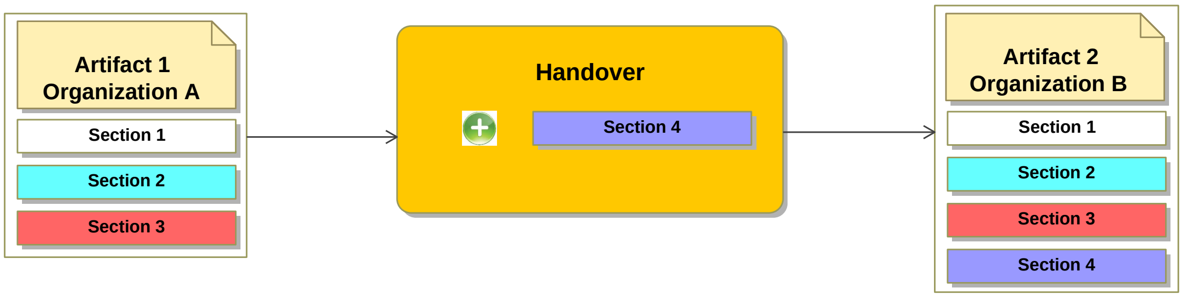


Figure 49: Missing Artifact during Handover

3. Organization A provides more content than Organization B needs

If too many artifacts are created by Organization A that exceed the needs of Organization B (Figure 50), there is no bottleneck or any other reason that would prevent the process from further being executed. Moreover, Organization A probably will think about optimizing its process for this specific collaboration by just omitting the creation of artifact's "Section 3". Exceeding process needs just for sake of the own process takes too much effort and time required in this case.

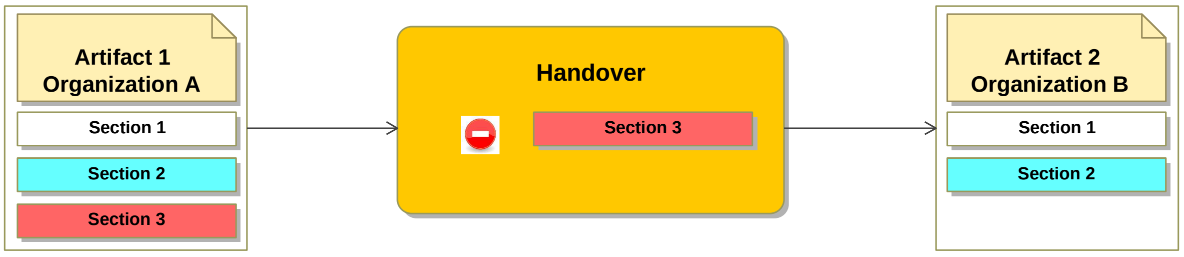


Figure 50: More Artifacts provided than necessary

As mentioned before, proper artifact exchange is an essential lever, which decides how successful software development collaborations are. For this reason, there is a list of artifact types with corresponding examples that might be crucial for collaborations. This list is derived by comparing and consolidating model outputs from e.g. CMMI® [133], V-model XT® [79], and IEEE 12207 [73].

Table 4: Crucial Artifact Types for Software Development Collaborations

Artifact Type	Corresponding Example
Plans	---
Specifications	▪ Requirement ▪ Design ▪ Architecture
Descriptions	▪ Manual ▪ Analysis ▪ Strategy ▪ Dependencies ▪ Constraints ▪ Guidelines
Code	---
Reports	---
Audits	---
Records	▪ Checklists ▪ List of decision criteria ▪ List of interfaces ▪ Actions ▪ Agreement ▪ Recommendation
Concepts/Proposals	---
Baselines	---
Change requests	---

3.3.2.2 Quality of Artifacts

As stated before, the number and types of artifacts that are transferred to the counterpart organization needs to be defined prior to process execution. Besides that, the quality of transferred artifacts is in many businesses very crucial (e.g. Healthcare business observed by FDA, security rules in nuclear power plant etc.) and therefore shall be defined. This means that certain mechanisms have to be institutionalized in collaborative business setup ups that allow for getting a consistent product quality throughout the entire collaboration.

Quality is inspected by mapping the existing output, e.g., source code or specification, to a defined set of criteria any artifact has to fulfill. This is typically done in milestone/quality gate review meetings in which all relevant parties participate, i.e., the suppliers and receivers.

Quality is jeopardized whenever artifacts' responsibility changes, i.e., if they are interchanged through interfaces between parties. Especially a collaborative process needs to address potential quality issues before they even occur by establishing quality review respectively. However, quality mechanisms are not particularly considered in this work.

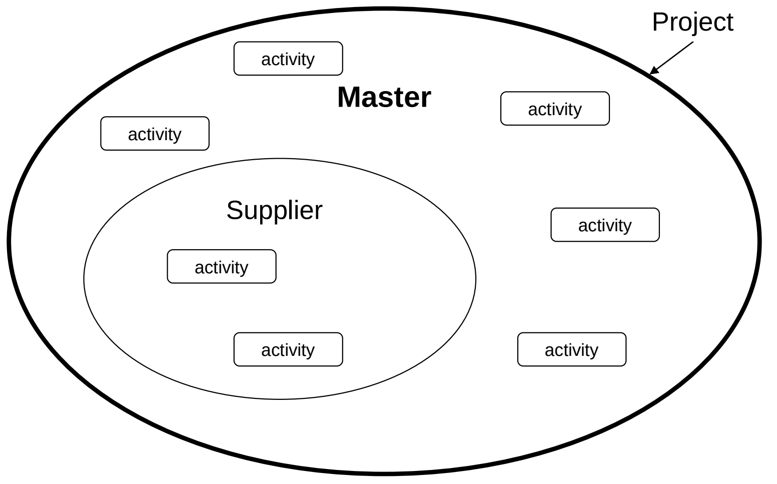
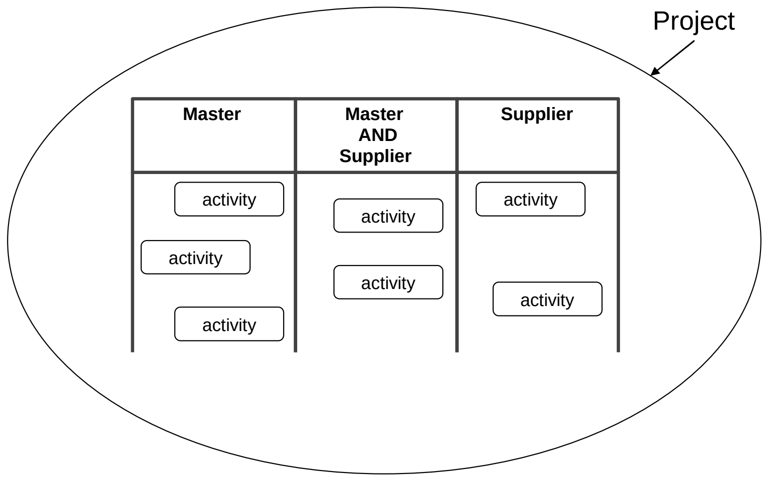
3.3.3 Role Model

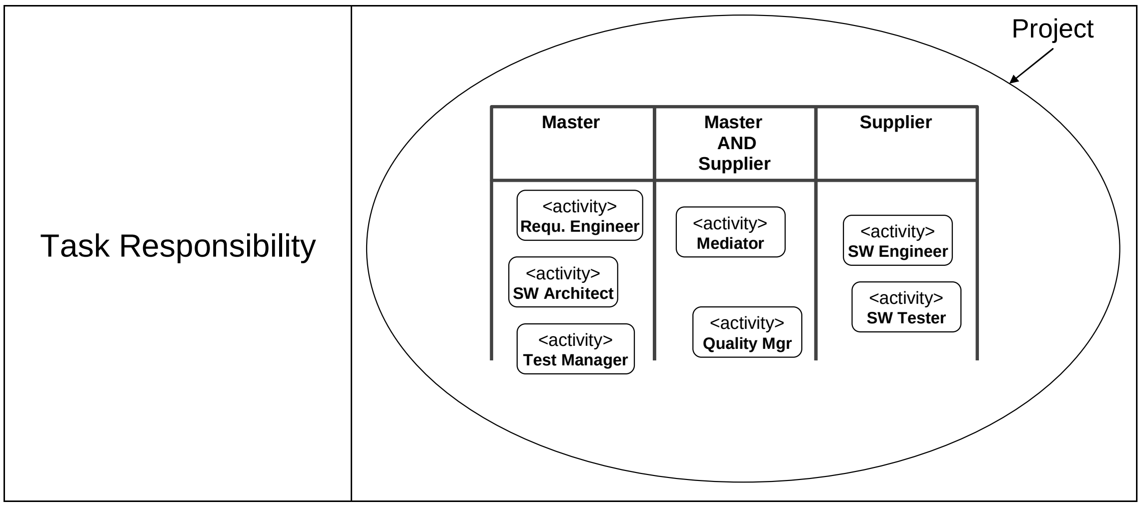
A role model becomes necessary within collaborations of two or more organizations. The purpose is to get responsibilities of specific actions and activities clarified prior to collaboration start. This reduces confusion during project run especially in global distributed projects.

This thesis provides a role model for accurately assigning responsibilities to appropriate tasks. Generally, different responsibility levels can be distinguished, whereupon three different levels of responsibilities are shown in Table 5 and explained in the following.

Although this work will not focus on on task responsibility level all three levels of responsibility (according to Table 5) are discussed to get a comprehensive feeling of potential responsibility domains.

Table 5: Role Model Definition Level

Project Responsibility	 Project activity Master activity activity Supplier activity activity activity activity
Activity Responsibility	 Project Master Master AND Supplier Supplier activity activity activity activity activity activity activity



3.3.3.1 Project Responsibility

If two or more organizations are working together, a leadership organization, a so-call "Master"-organization shall be defined. Consequently, all other collaborating organizations are Suppliers. This differentiation results from observations from practice, where in most of the cases, one organization takes on the Master role. This is similar to a consortium where a so-called "Sole mandated lead arranger" is defined that manages all consortium internal affairs [74]. Consortium is a Latin word, meaning 'partnership, association or society' and is derived from consors 'partner', itself from con- 'together' and sors 'fate', meaning owner of means or comrade.

The role of the Master organization depends on the strength of collaboration. In Figure 51, three different types of collaboration strengths are modeled. The master organization is only explicitly existent in case of weak and medium strength collaboration, whereas it is implicitly modeled in strong or tight collaborations (high strength). The definition of 'strength' follows the number of communication or synchronization (yellow diamond) points (white arrow).

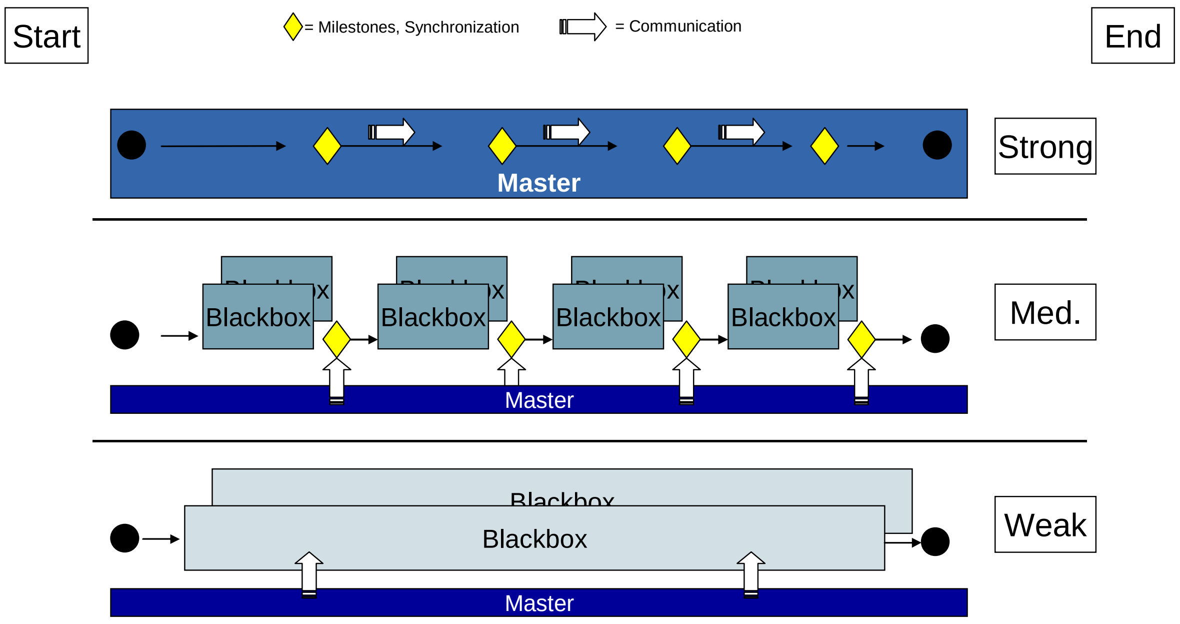


Figure 51: Project Responsibility: Strength of collaboration

In the ‘weak’ case, only a few synchronization points are taking place; in the ‘strong’ case, synchronization and communication are done throughout the entire process. This is also illustrated by having the Master directly included into the process.

Examples for the ‘weak’ case are processes that are well established and optimized, such as delivery of spare parts. In this case, less critical issues are to be discussed, e.g. the number of spare parts to be delivered.

The ‘medium’ case concerns, e.g., development of additional features based on an existing platform in software or hardware development.

The strong case typically takes place if new innovative features are developed, especially if more domains need to be included, e.g., “Global Positioning System (GPS)[12]” or the European pendant “Galileo”[13] navigation system. This requires both software and hardware development in conjunction with satellite and network connections. The “Toll Collect[14]” system in Germany is an example where several organizations are linked together by setting up a consortium. In this case, the Master organization would be the consortium leader responsible for project’s success and failure.

3.3.3.2 Activity Responsibility

In order to exactly define which organization is included and responsible for executing various activities and actions, three role connectors have been defined. Table 6 summarizes those connectors that are available on organizational level for collaborations.

Remarks: the term “Role” is seen here on organizational level.

Table 6: Definition of Role Connectors

<no connector>	Exclusively one organization is executing an action
$\wedge$ (“AND”)	Both organizations are executing an action
$\vee$ (“OR”)	Either Organization A or B or both executing an action
$\otimes$ (“XOR”)	Either A or B, but exactly one organization is executing an action

The role concept is illustratively applied in the scenario in Figure 52 (Remark: It is assumed that Organization A (“Master”) and Organization B (“Supplier”) have semantically equivalent processes defined). The swim lane diagram defines task responsibilities in the headline of each swim lane (marked with a green box in Figure 52). Therefore, the *Master* is in charge of *System Design* by his own. *System Implementation* is done by both *Master* and *Supplier* as the “AND” connection according to Table 6 in the headlines shows. Finally, the Supplier is solely responsible for S*ystem Test*. The termination node goes back to the *Master* since this organization initiates the entire collaboration.

[12] http://en.wikipedia.org/wiki/Global_Positioning_System; 2010, Nov-25
[13] http://en.wikipedia.org/wiki/Galileo_%28satellite_navigation%29; 2010, Nov-25
[14] http://www.toll-collect.de/

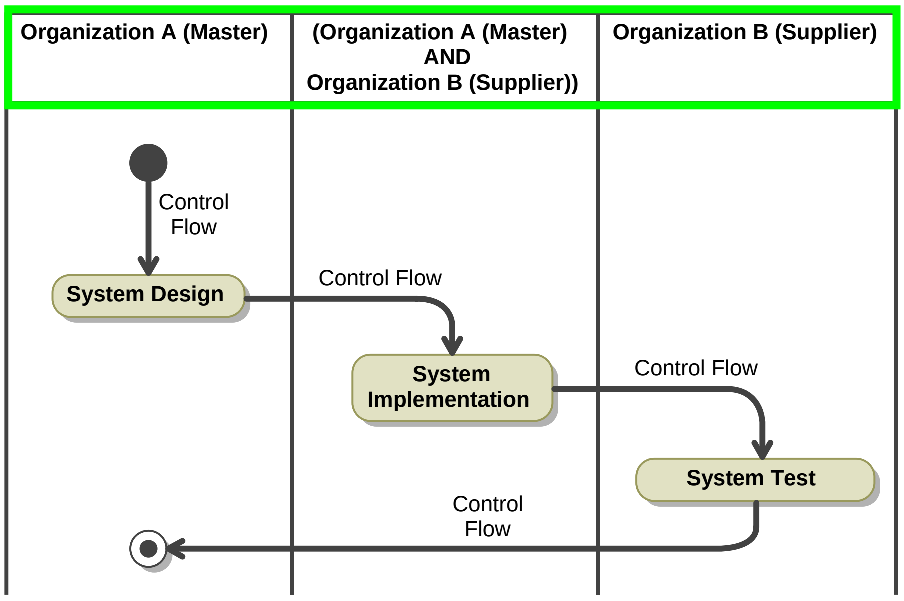


Figure 52: Activity Responsibility: Organizational Role Definition

3.3.3.3 Task Responsibility

Based on the activity assignment of relevant activities/actions content- specific roles have to be defined and assigned to those activities. Thereby, content- specific and collaboration specific roles are differentiated. Table 7 depicts a set of potential roles, which is not limited to those. All described roles might be defined, however, these are not mandatory.

Content- specific roles are typically defined anyway in organizations that contributed with specific artifacts, e.g., Requirement Specification to organizational processes.

In contrast, the role *Mediator* is typically defined in collaboration. This role would be responsible for early recognition and mediation of cross organizational conflicts and/or appropriate escalation to senior management.

Furthermore, there are also roles that can be allocated in both categories. Besides the Project Manager as a core role, Tool Administrator, Quality Manager, or Configuration Manager might be crucial for projects' success in a global and distributed development environment.

Table 7: Task Responsibility: Specific Roles

Content specific	Collaboration specific
Requirement Engineer	Mediator
System/Software Architect	…
Software Engineer	
Hardware Designer	
Hardware Engineer	
Test Manager	
…	
Project Manager	
Tool Administrator	
Quality Manager	
Configuration Manager	
…	

4 Solution Scenarios

“Deal with the difficult
while it is still easy.
Solve large problems
when they are still small.
Preventing large problems
by taking small steps
is easier than solving them.
By small actions
great things are accomplished.”

- **Lao Tzu** (604-531 BC)

As illustrated in previous sections global development project are enormously stipulated by development challenges. The following section provides a solution for these challenges containing a process- based methodology how two or more organizations can work together in a globally distributed development environment. The result is a new collaborative process, which contains the original sub-processes of the collaborating organizations.

These collaborative processes are described by a set of standard collaboration scenarios, which have been derived from industry experience and literature review. The scenarios incorporate the introduced *mediator* that helps development organizations setting up an integrated process environment.

4.1 Preliminaries

The following processes and all subsequent defined scenarios are modeled using activity diagrams according to UML [24], [114], [75]. All control flows in the diagrams are in **bold**. For better readability, the following action color codes (Table 8) are used throughout all activity diagrams.

Table 8: Legend and Color Codes for Activity Diagrams Usage

<Organization A> <Organization A>	Actions executed by Organization A (including alternative color, exclusively used)
<Organization B>	Actions executed by Organization B

<Organization C>	Actions executed by Organization C
<Organization D>	Actions executed by Organization D (Case Study 1)
<Organization E>	Actions executed by Organization E (Case Study 1)
<New action>	Actions that are newly defined to connect original processes for setting up a collaborative process
<Evolutionary action>	Actions, that are newly **defined** or **derived** based on the input of existing actions from collaborative organizations ("evolutionary")
<hierarchical action>	Hierarchical Action (=trident) of any organization respectively Remarks: color code might vary due to respective needs

4.1.1 Integration Possibilities

Integration of activities into other activities is a basis for collaboration in general. Process integration enables companies to master the challenge of having people located around the world, working for one project. These processes can be implemented in different ways, depending on the strategy a development organization is willing to pursue. Three basic integration strategies are explained in the following:

1. First, the Master can enforce the supplier to completely accept and follow his (the master's) processes. In this case, there is no further integration approach necessary since the processes to be used are somewhat dictated by the master.
2. Second, the master might include some selected activities from the supplier's processes. In this case, there is a need to define an approach on how to comprise other organizations' processes.
3. Last, the process is set up on a global basis with dispersed teams. This means that organizational processes are set up in a joint way by having several organizations or companies working together as equivalent partners with a defined lead.

This work focuses on the second and third integration possibilities, since the first approach does not leave any remaining unclear process issues to be clarified due to clear process guidelines provided by the Master organization.

4.1.2 Illustrating Example

In order to illustrate the concept ideas, this section provides a small concurrent example (Figure 53), which applies the explained theory.

Remarks: Due to the fact that the collaboration set-ups are different depending on the scenario shown, the root process of some scenarios shown later on will slightly differ from the initial root process in Figure 53. This is necessary to illustrate clearly the functionality of the appropriate functionality of the overall approach and the resulting collaborative process in each case.

Nevertheless, the basic root scenario consists of two organizations: Organization A, which functions as the 'Master' and Organization B/C as the 'Supplier'.

From an industry point of view, the Master is that organization, which initiates the entire collaboration and makes itself responsible for managing and controlling it [62]. This encompasses, e.g., the definition of desired (sub-) products and work packages, the identification of additional, special competence for implementation, hardware development, or testing to optimize overall collaborative effectiveness and efficiency. The 'Master' organization is typically the one that receives all work products, so-called artifacts that are produced or developed during project run. In turn, 'Master' does not mean that the complete workflow of the Master's organization is mandatory for all organizations to follow.

The other collaborating organizations are 'Suppliers'. The suppliers provide special competences that make them attractive for the Master, so that the Master is coerced from an economical/technological point of view to collaborate with the supplier(s). A special competence of the supplier is, e.g., special knowledge in any technical domain. Strategic advantages of a supplier from a Master's view are, e.g., the supplier's location to penetrate new markets or a low- cost development site.

As already implicitly mentioned, complex development projects might include more than one supplier. Nevertheless, the process integration approach of this work is scalable in a way that it fully supports participation of more than one supplier.

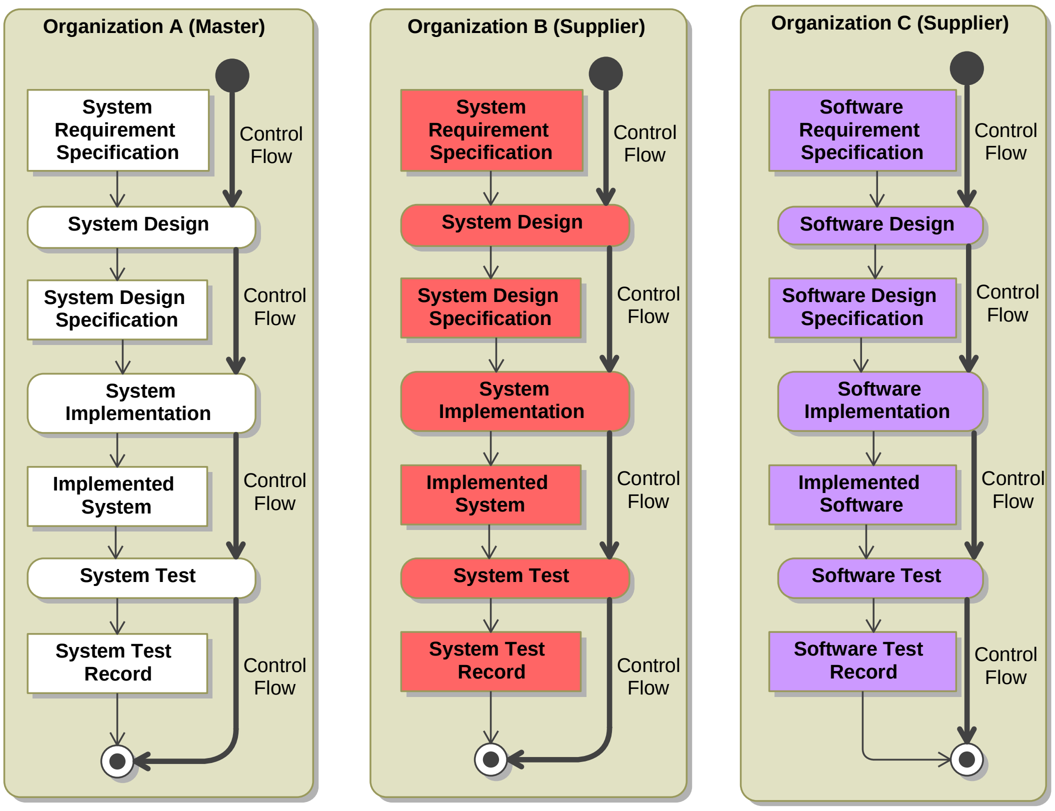


Figure 53: Root processes of Organization A, B and C

Based on the root processes of Organizations A, B and C in Figure 53, different types of collaborations and are explained in the following. This work does not define one single model, which is useable for all types of collaborations that might occur anytime. Moreover, a few typical collaboration scenarios from practice are generalized so that they can be applied to any defined process [83], [82].

The Master and each Supplier know their own specific development process and approach only from their point of view. As depicted in Figure 53, the process of Organization A and B consists of three actions *System Design*, *System Implementation*, and *System Test*. Organization C defines its processes on a finer grained level, i.e. *Software Design*, *Software Implementation*, and *Software Test*. Of course, this does not represent an entire product lifecycle; however, it is a sufficient part for illustrating this basic concept. As labeled in the diagrams (Figure 53) they contain both control flow (in bold) and object flow, which means that appropriate artifacts are also modeled as objects nodes, e.g., *System Specification*, *Implemented Software*. In order to correctly follow process semantics, two activity parameter nodes are added, that are *System/Software Requirement Specification* as input for System Design and *Software/System Test Record* as output of *Software/System Test*.

4.2 Scenarios for Process Integration

4.2.1 Semantically Equivalent Processes

The first scenario defines the collaboration process of Organization A and B using semantically equivalent processes. This significant assumption exists only in this scenario, which means for the two root processes in Figure 53 that not only the action names are equal; moreover, they mean exactly the same. This concerns the description of activities, e.g., identical steps, actions, used templates, programming languages etc., as well as the interpretation of the appropriate partner process (Organization B), which is identical to its own process. This means, in turn, that the Master (Organization A) expects any action or activity from the potential Supplier (Organization B) to be exactly the same as in his own organization. Therefore, the Master can replace any of his process steps by process steps from the supplier, without expecting any compatibility or interface issues.

If this scenario is applied to the concurrent example in Figure 53 (root processes), it is assumed that the Master wants to outsource the *System Test* to the selected Supplier, e.g., in order to reduce cost or to get more independent results out of the System Test. The collaborative process is depicted using swim lanes within UML activity diagrams. Due to the fact of having equivalent processes defined, the Master can just transfer *System Test* to Organization B, as illustrated in Figure 54. *System Test* is conducted by Organization B without any further interface in-between. After *System Test* the control flow points back to the Master, since – as stated before – this is the organization that controls and manages the collaboration.

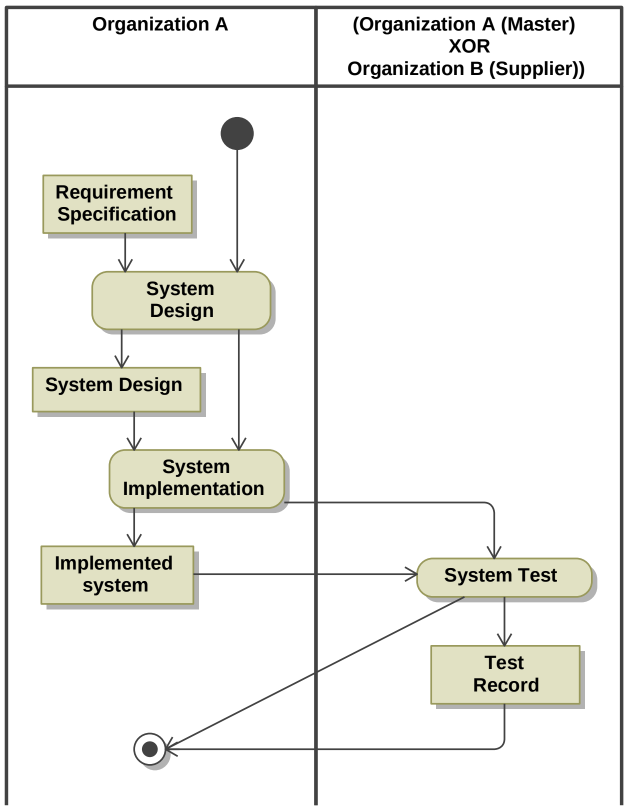


Figure 54: Collaboration with equivalent processes

Application to practice

Referring to practical experience, this development scenario is very rare in industry. Every company develops their specific products and has therefore specifically defined development processes for both software and hardware that optimally support the business. The probability that two development processes of two independent organizations are equivalent is very low. However, experimental public R&D environments like universities might have identical processes for conducting internships or even development projects [96].

4.2.2 Horizontal Integration

The scenario 'Horizontal Integration' is similar to the previous. However, this scenario gives up the assumption that processes of the organizations are semantically equivalent. This is due to the fact that in practice those processes to be connected are typically semantically not equivalent.

The root processes in Figure 53 are still the same, but they do not mean the same anymore. For instance, *System Implementation* of the Master is now differently done compared to *System Implementation* of the Supplier, e.g. Organization A uses methods from agile development, whereas Organization B follows the Rational Unified Process (RUP). Consequently, actions or activities cannot be easily just exchanged anymore, because *System Implementation* of the Supplier (Figure 53) may need other input artifacts than the Master creates as output from *System Design*.

In order to solve this issue the process integration approach provides a mediator (Figure 55) that has been introduced and described in chapter 3.3.1.

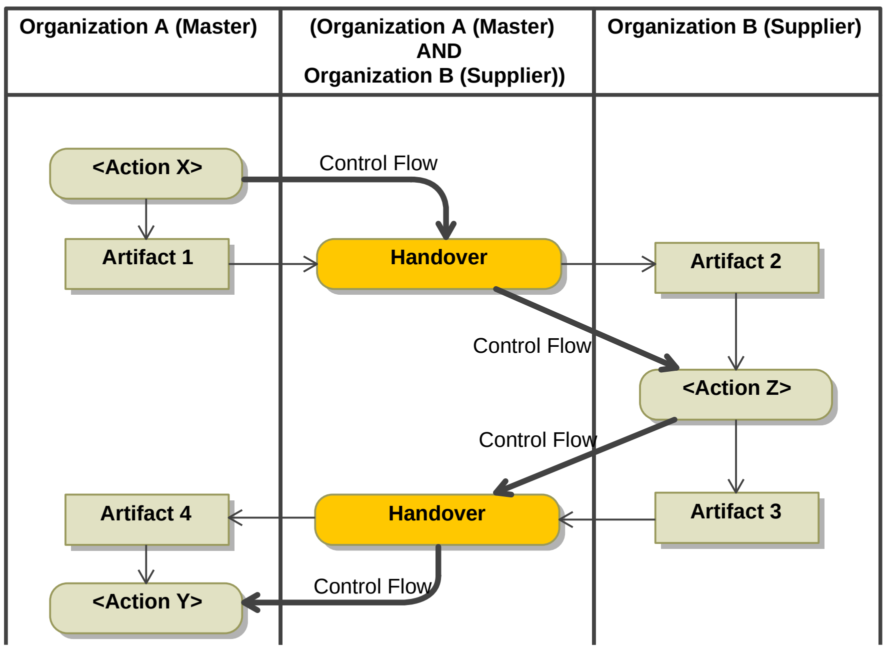


Figure 55: Mediator pattern for ‚Horizontal Integration'

Referring to the concurrent example, it is assumed that the Master wants to outsource the *System Implementation*. Since the processes are not semantically equivalent the usage of the mediator pattern is necessary to create a collaborative process as illustrated in Figure 56. The scenario is called 'Horizontal Integration', because the action to be outsourced is pushed horizontally into the root process.

System Design of Organization A is the connection point previous to the *Handover* action, which needs to be newly included by following the definition of the mediator in Figure 47. *System Design Specification* of the Master is a central input for the *Handover* action, since this is the basis for further development activities. The execution of the *Handover* also results in a *System Design Specification*, which is illustrated in Figure 56. The functionality of the *Handover* action in this case is the conversion of the *System Design Specification* as output from Master's *System Design* to a *System Design Specification* as input for the Supplier. Based on the *System Design Specification* of the Supplier, the *System Implementation* is executed by the Supplier resulting in the *Implemented System*. The corresponding action on the Master's site is not necessary anymore; therefore, it is deleted. The artifact *Implemented System* needs to be "*converted*" backwards to the Master's format as input for *System Test* (Master). For this purpose, another *Handover* action is included, according to the mediator definition in Figure 55, which exactly creates that output in the desired format for the Master. By having the second *Handover* added to the collaborative process, it is demonstrated that both parties (Master and Supplier(s)) have to conduct and mutually agree upon this artifact. Consequently, this is the basis for the *System Test* done by Organization A.

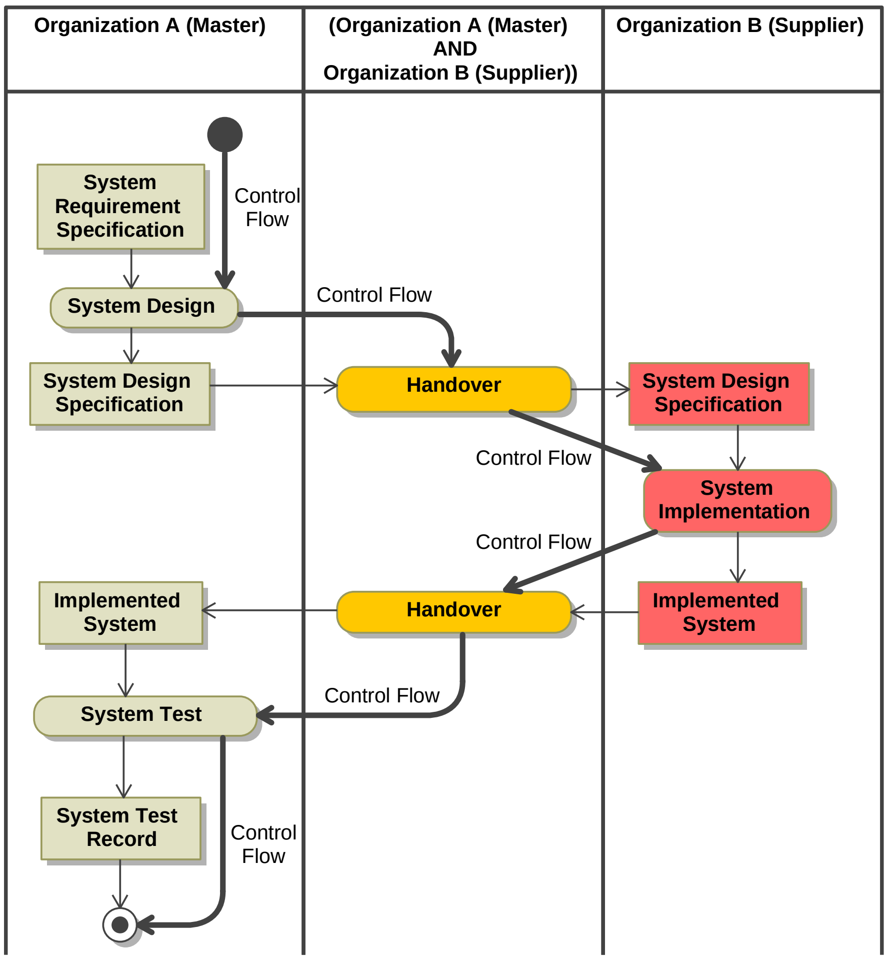


Figure 56: Collaborative Process for 'Horizontal Integration'

This scenario seems like a typically sub-order relationship as defined in V-Model XT® [79], as a standard development model of the Federal Republic of Germany and guideline for planning and conducting development projects [85]. However, this approach even extends such a sub-order relationship due to the fact that indeed the process models are not semantically equivalent, and process integration can be conducted at whatever point it is intended; in turn, the decision of task allocation is not modeled in this process as it is with the V-Model XT®.

Application to practice
The scenario very often applied in development businesses with short development cycles, e.g., in mobile phone software development. For this purpose, component specifications are handed over to a specific development site, for instance, India.

4.2.3 Additive Vertical Integration

Using the scenario 'Additive Vertical Integration' provides the possibility to include whole action chains into a process. This is relevant if organizations want to outsource entire development domains like software, hardware, or mechanical engineering.

In order to illustrate this scenario, the initial root process from Figure 53 has to be slightly modified as depicted in Figure 57. Organization A does now have a concrete software development process portion defined containing *Software Requirement Engineering*, *Software Design*, *Software Implementation*, and *Software Test*. The Supplier develops Hardware using a process that consists of *Hardware Design*, *Hardware Implementation*, and *Hardware Test*.

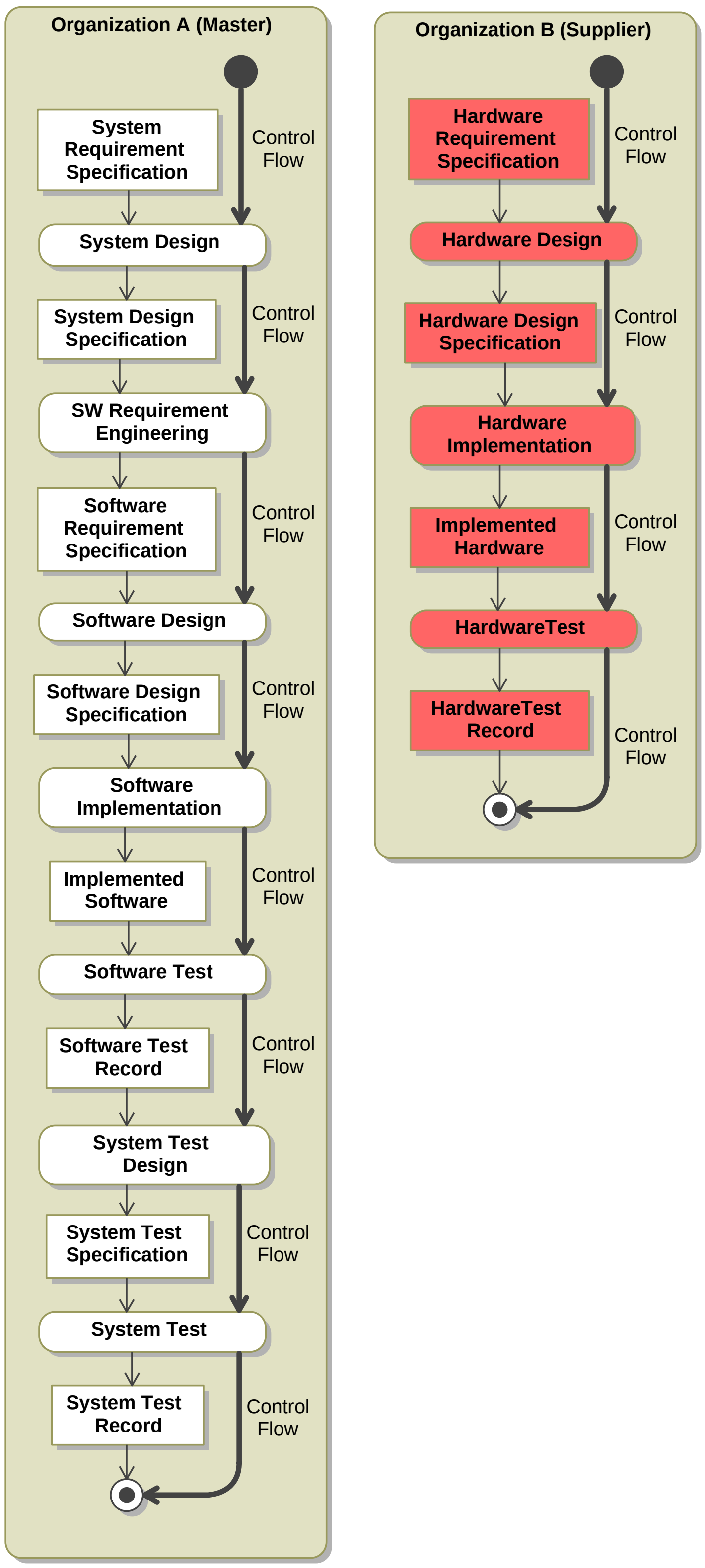


Figure 57: Root process 'Additive Vertical Integration'

The appropriate mediator for this scenario is illustrated in Figure 58. The Master is again the initiator of the collaborative process and <*Action X*> is the connector from the Master's development process, which is followed by *Decomposition*. This *Decomposition* is a newly defined action that is necessary to allocate features to the software or hardware development process. *Decomposition* is followed by two artifacts 2 and 5, which are created and defined in a way that they are usable input for <*sub-workflow A*> (white) and <*sub-workflow B*> (red). The output artifacts 3 and 6 of these sub-workflows are in the same type than the inputs. After the development part *Integration* of software and hardware products takes place, which essentially combines several features or sub-features that have been developed in different domains towards one system or sub-system. As illustrated, the *Integration* is also newly created and included as connection point. After *Integration* process flows back to the Master, which is symbolized with <*Action Y*>.

The unique characteristic of this 'Additive Vertical Integration' scenario is parallelization of actions and activities during product development.

Depending on the process granularity, there may exist several points of synchronization in-between. From a practical point of view, the number of newly defined synchronization points due to collaborative processes is limited, since hardware development does usually have other milestones defined anyway. Those milestones differ from those of software development, e.g., hardware prototype manufacturing, non-destructive testing.

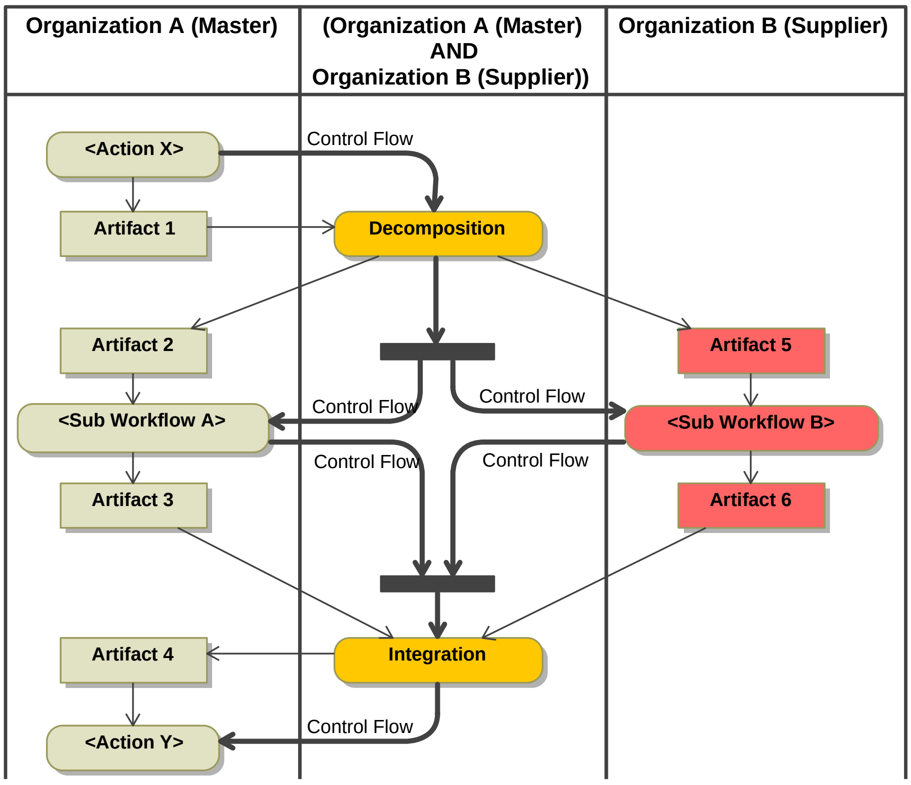


Figure 58: Mediator for 'Additive Vertical Integration'

Going back to the concurrent example, it is now assumed that the Master develops a system containing software and hardware. For this purpose the Master wants to outsource hardware development. Concretely, an entire action chain, i.e., the hardware development process of the Supplier, needs to be included to generate a collaborative process, which is shown in Figure 59.

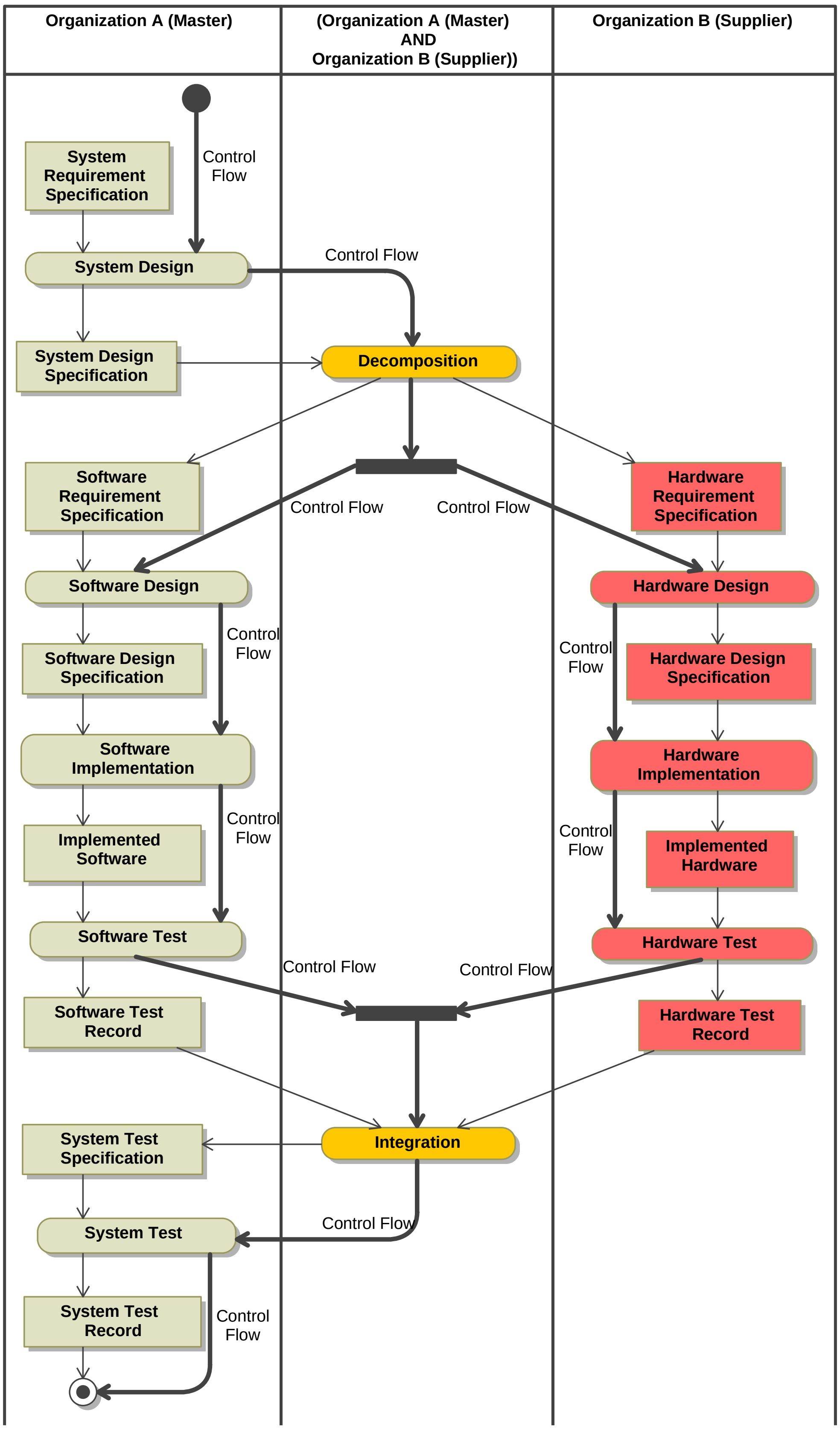


Figure 59: Collaborative Process 'Additive Vertical Integration'

System Design of Organization A is decomposed into a *Requirement Specification* for hardware and software. The decomposition is – as in 'Horizontal Integration' – not only an allocation of features, but also a mutual agreement of all parties participating in the collaboration.

This encompasses:

- Allocation of features (as mentioned before)
- Type of templates to be used
- Clarification how configuration management is done
- Definition of synchronization points or milestones respectively within development phase, incl. reports etc.

After development phase of software and hardware that ends with *Software* and *Hardware Test*, a further *Integration* needs to take place. Within this action the implemented hardware is brought together with the developed software. This includes on the one hand the mutual agreement of Master and Supplier that hardware was developed the way the Master wanted to have it. On the other hand software is technically flashed onto hardware. This includes also integration testing, which is not modeled in Figure 59 to keep the scenario clear and less confusing.

Application to practice

This scenario basically demonstrates 'process parallelization', which is nowadays applied in almost every development organization. Referring to Figure 59, this collaborative process could be used in any healthcare development department (e.g. computer tomograph, magnetic resonance tomograph) where one business unit develops hardware components; whereas, another business unit is responsible for software development.

Furthermore, mobile phone development does also apply for this scenario, e.g., if one organization develops the hard case, chipset and responders like Bluetooth, Infrared, or Wi-Fi, and another organization implements adequate software systems that supports all the hardware functionality for the phone.

4.2.4 Alternative Vertical Integration

Following 'Additive Vertical Integration' the scenario 'Alternative Vertical Integration' is derived. From a modeling perspective this scenario is very similar to the 'Additive Vertical Integration' with one essential difference: processes are not executed in parallel, but alternatively. The mediator for the scenario is depicted in Figure 60. It shows that only one sub-workflow is executed at each process run, either <*sub-workflow A*> or <*sub-workflow B*>. The decision which process path is taken is decided in the *Rational Analysis* that is an additional, newly defined action in that scenario. After development run an *Acceptance* of resulting work products or artifacts needs to be done. This is followed by a connection point towards the Master (<*Action Y*>).

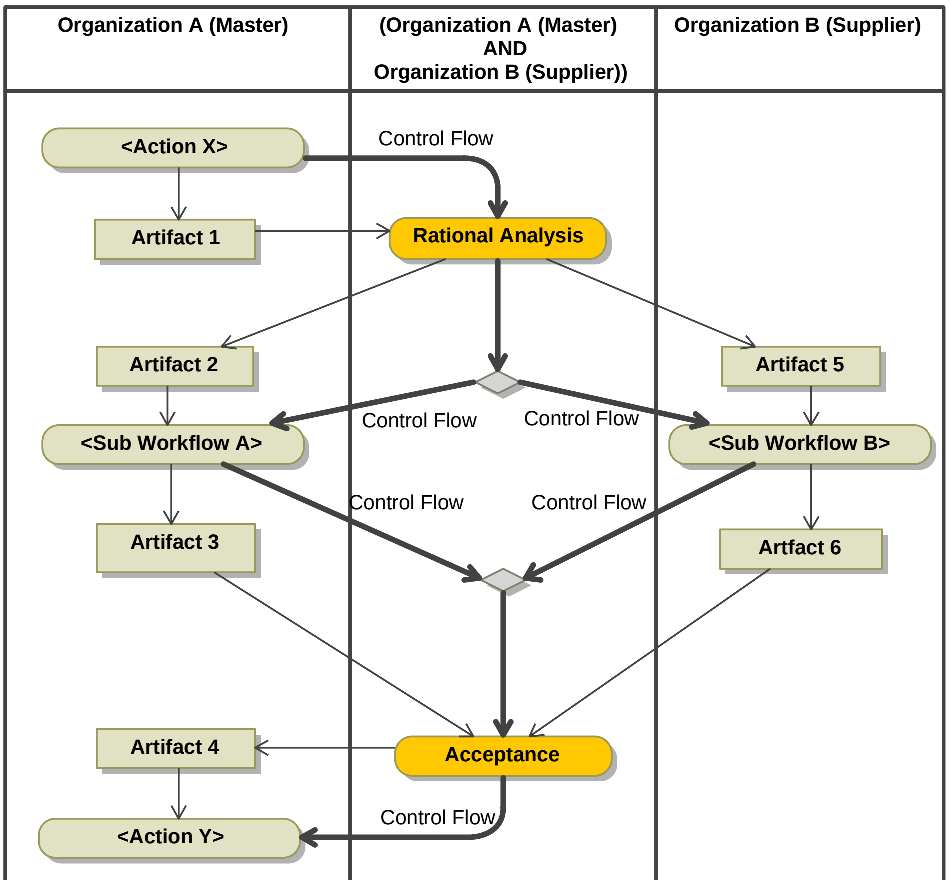


Figure 60: Mediator for 'Alternative Vertical Integration'

For illustration purposes, the current root process in Figure 57 is again slightly modified as depicted in Figure 61. The Supplier does not deliver hardware components, but safety-critical software, due to its specialization and resulting special knowledge in this domain. The collaborative process in the concurrent example is illustrated in Figure 62. After the Master's *System Design* a *Rational Analysis* starts to clarify which Organization implements dedicated components. If any component contains safety-critical parts to be implemented, the Supplier will be in charge of this component, otherwise the Master is responsible. After *Rational Analysis*, the Master or Supplier develops the desired component. If the Supplier has been in charge of a safety critical component Master and Supplier conduct the *Acceptance* meeting, where the appropriate components are checked and validated whether they comply with the definitions in the specification. After that, the workflow goes back to the Master's initial workflow (<*Action Y*>).

Application to practice

The scenario is mostly relevant in a longer lasting business relationship of two or more partners, since the character of the scenario is rather 'iterative'. This gets explicit by using the *Rational Analysis* to evaluate, e.g., the safety criticality of a component. Typically, this does make sense in case components need to be developed sporadically, instead of continuously in a project.

This scenario could also be a sub-process of the entire collaborative process landscape where iterative loops are defined and executed if necessary.

Safety-critical software concerns software that shall protects human’s life and should not be mixed up with security software, e.g. a “digital doorman” before entering a website. Safety-critical software is relevant, e.g., in trains that are used for public transportation, like the German “Intercity Express (ICE)”. This software has to follow standard, e.g., CENELC[15] in which a software development organization might be specialized. This organization can offer its know-how of CENELC conform programming to Federal Railway Authority or to another supplying third party.

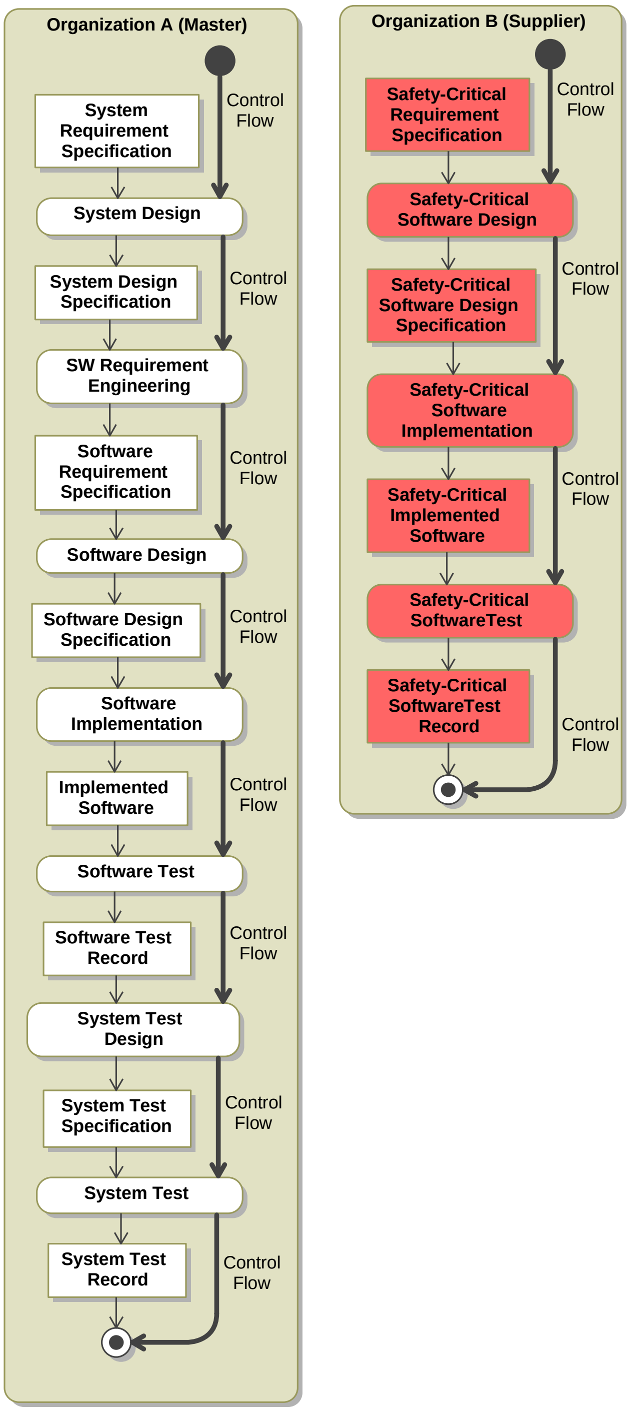

Organization A (Master)
Control Flow
System Requirement Specification
System Design
System Design Specification
SW Requirement Engineering
Software Requirement Specification
Software Design
Software Design Specification
Software Implementation
Implemented Software
Software Test
Software Test Record
System Test Design
System Test Specification
System Test
System Test Record
Organization B (Supplier)
Safety-Critical Requirement Specification
Safety-Critical Software Design
Safety-Critical Software Design Specification
Safety-Critical Software Implementation
Safety-Critical Implemented Software
Safety-Critical SoftwareTest
Safety-Critical SoftwareTest Record

Figure 61: Root process 'Alternative Vertical Integration’

[15] Comité Européen de Normalisation Électrotechnique (CENLEC)

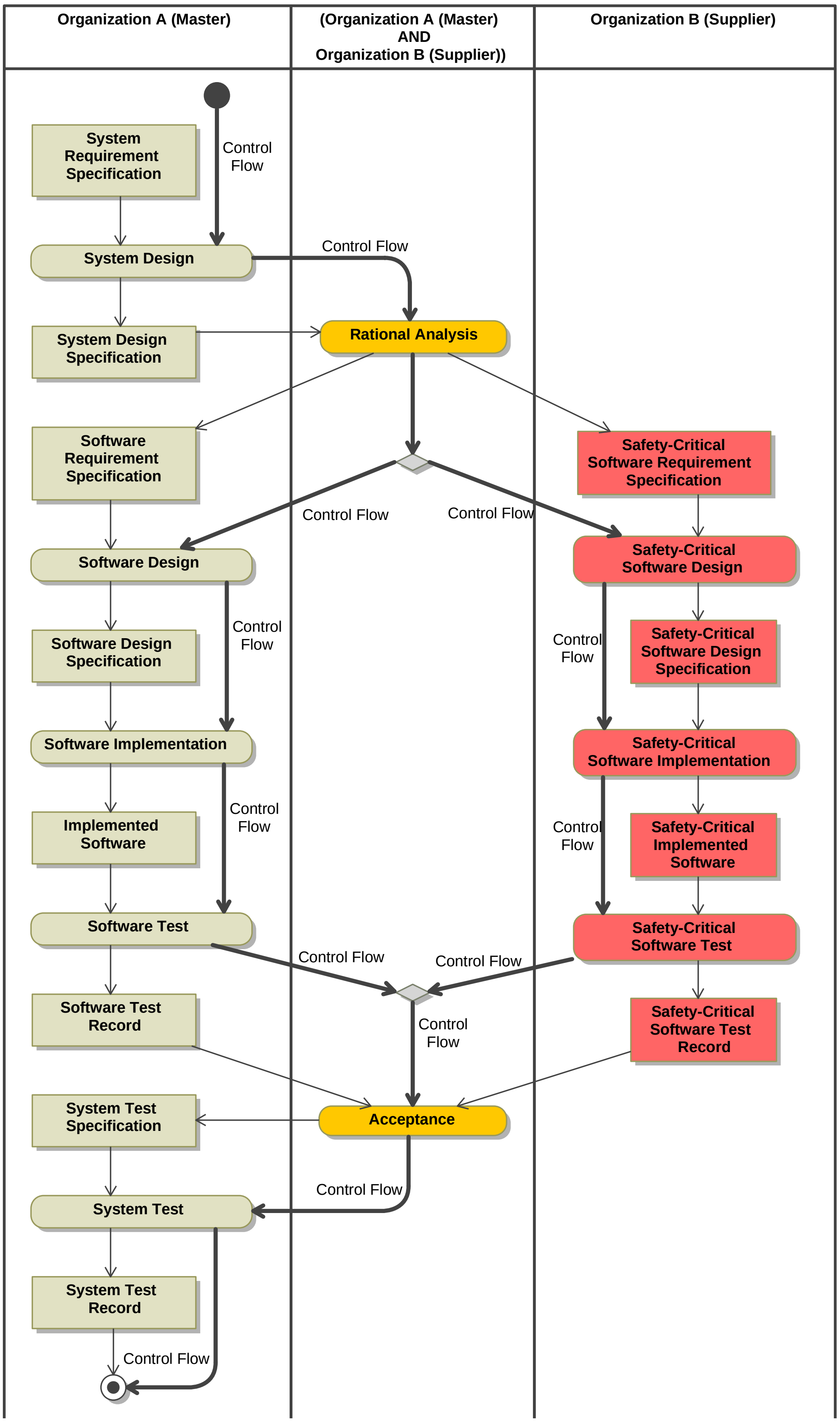

Organization A (Master)
(Organization A (Master) AND Organization B (Supplier))
Organization B (Supplier)
System Requirement Specification
Control Flow
System Design
Control Flow
System Design Specification
Rational Analysis
Software Requirement Specification
Safety-Critical Software Requirement Specification
Control Flow
Control Flow
Software Design
Safety-Critical Software Design
Software Design Specification
Control Flow
Control Flow
Safety-Critical Software Design Specification
Software Implementation
Safety-Critical Software Implementation
Implemented Software
Control Flow
Control Flow
Safety-Critical Implemented Software
Software Test
Safety-Critical Software Test
Control Flow
Control Flow
Software Test Record
Control Flow
Safety-Critical Software Test Record
System Test Specification
Acceptance
Control Flow
System Test
System Test Record
Control Flow

Figure 62: Collaborative Process: 'Alternative Vertical Integration'

4.2.5 Merging Integration

The scenario ‘Merging Integration’ brings in a significant difference compared to all previous scenarios. As the naming already shows this scenario comes up with the possibility to have newly defined activities. This means that activities are, e.g., enriched by merging them with other activities from other organizations, resulting in new, evolutionary activities. The mediator for this scenario has several characteristics, depending on how many organizations are participating on the merge. This means, in turn, that a merge can also be conducted within one single organization. These two different cases are explained in the following.

4.2.5.1 Integration with more than one organization

This scenario shows that some actions are executed conjointly, i.e., all participating organizations are doing actions or activities together. The mediator for this scenario is depicted in Figure 63.

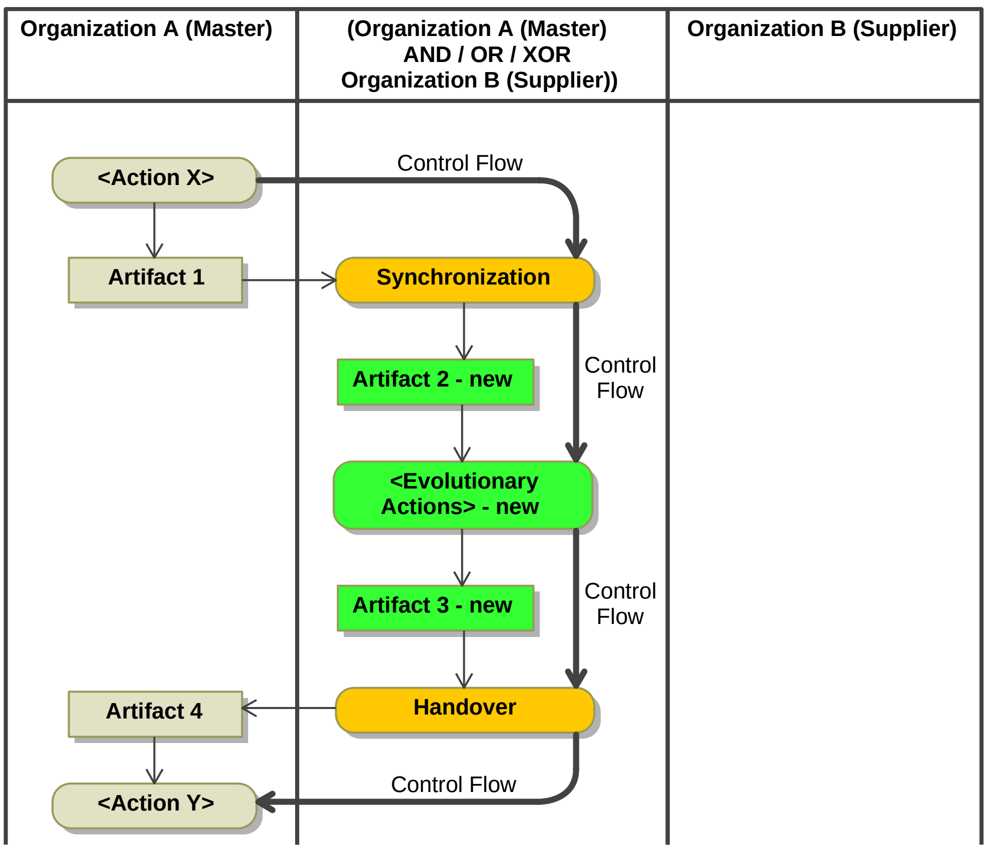


Figure 63: Mediator for ‘Merging Integration’

<Action X> of the Master symbolizes again the connecting point to the Master’s processes. Remark that no connecting action from Organization B is defined. The role model defines a *Synchronization* action between Organization A and B with the “AND”, “OR”, or “XOR” connector, which gives the Supplier the chance to share the Master’s intention what work should be done jointly. This is the preparation step towards the jointly executed actions containing mutually agreed artifacts. The resulting artifact ‘*Artifact 2 – new*’ documents

Synchronization and functions as input for '<*Evolutionary Actions*> - *new*'. After finishing the conjoint part, an additional *Handover* action needs to be added, which converts the output '*Artifact 3 – new*' to '*Artifact 4*'. This conversion seems somewhat redundant; however, it makes sure that Organization A gets exactly the right input artifacts for further process execution. Having *Artifact 2* as input, <*Action Y*> is again the connector back to the Master's organization.

The root processes for this scenario are illustrated in Figure 64, which is identical to Figure 53.

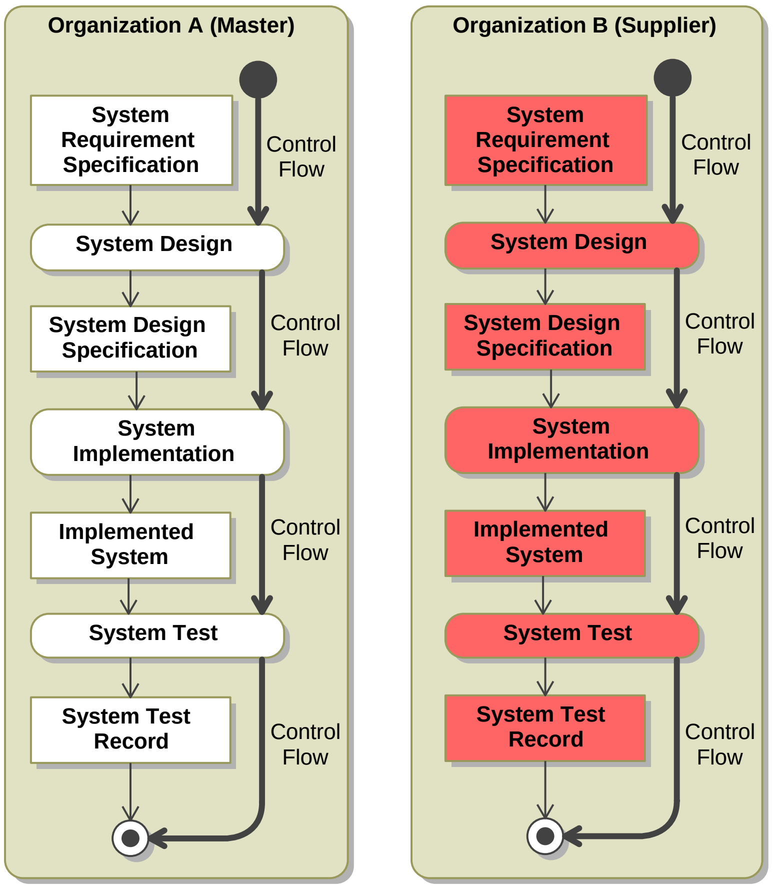


Figure 64: Root process 'Merging Integration'

The scenario assumes that the Master organization intends to conduct system implementation of a product with the supplier jointly. The Supplier (Organization B) is typically not familiar with the Master's product requirements. Therefore, it is crucial for the Master to have a *Synchronization* action for both participating organizations implemented to make the Supplier familiar with those product aspects necessary for implementation. In Figure 65 turns the *Synchronization* turns the *System Specification* from Organization A into a *System Specification* (green colored) usable for Organization B, which is commonly agreed upon both organizations (Figure 65). The green colored actions are those which have to be added to the initial process of Organization A and B. After conjoint implementation, another *Handover* is necessary to give the product back to the Master for *System Test*.

Remarks: The collaborating parties are still distributed, although this scenario might suggest that the participating organizations are in one location.

Application to practice

This scenario is useful in companies that develop leading edge products with an appropriate technology. These organizations typically have a need to hire or acquire the best and special individual resources or even small companies that provide the required know- how for new technologies. These specialized resources need to be simultaneously included into the process, which is possible using the ‘Merging Integration’ scenario. The advantage is that new and innovative ideas can be discussed and tested right when they arise. For leading edge innovations for the future, this is more effective than having synchronizations points from time to time.

For instance, in the United States, a car is only sellable if it features a cup holder; this is essentially due to the culture of the American people who very often have breakfast on the way to work [88]. This enforces development organizations to establish specialists who define requirements that are a “must have” for a product to be sold successfully in a special country.

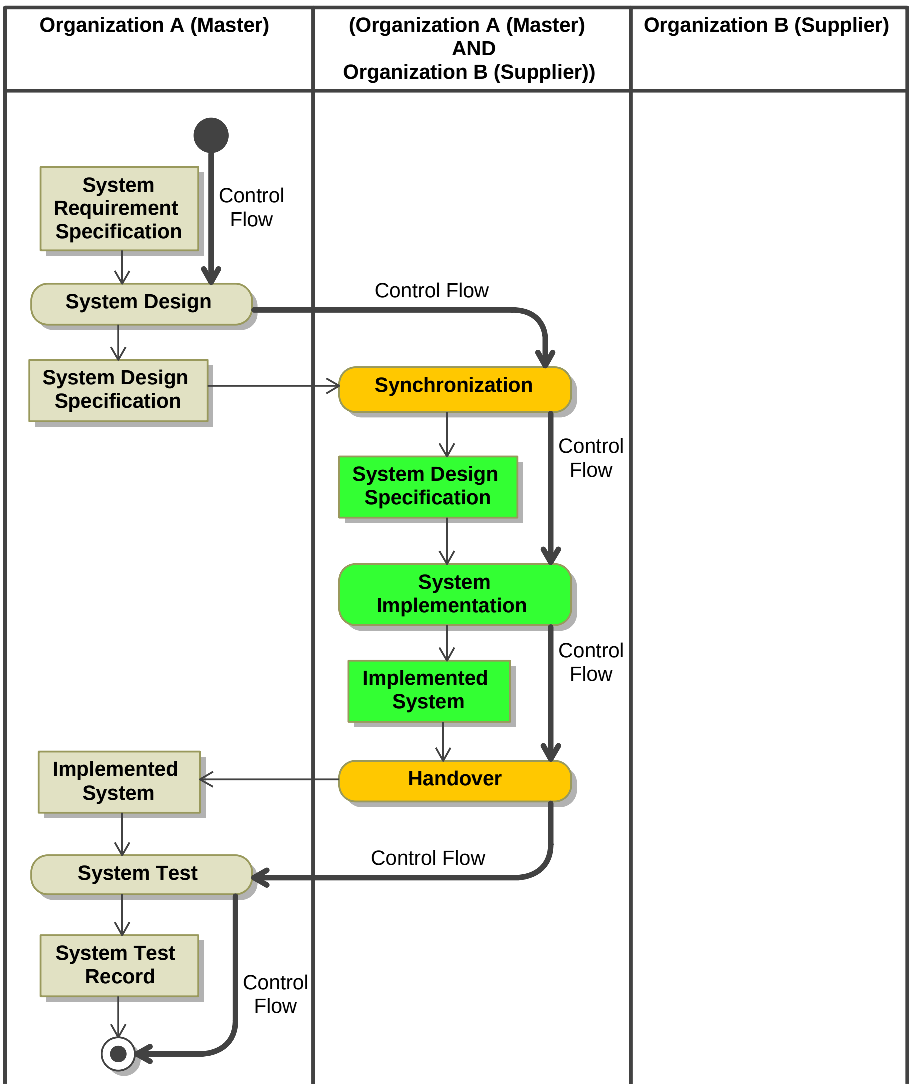


Figure 65: Collaborative Scenario: ‘Merging Integration’

4.2.5.2 Integration with one organization

A ‘Merging Integration’ scenario is not restricted to a minimum of at least two participating organizations. The respective mediator for the integration type within one single organization is depicted in Figure 66.

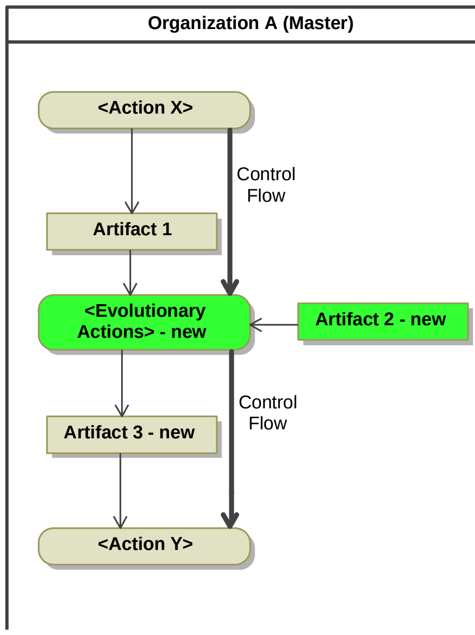


Figure 66: Mediator “Evolutionary Integration” (single organization)

It seems peculiar that this mediator only needs one swim lane, which is the one of Organization A (Master). An explicit connection point for any intermediate actions like *Handover* or *Decomposition* is not needed here. This makes this kind of mediator somewhat “artificial”. Organization A decides to include another input artifact for a respective action which will be executed during project run. Since the original activity will be changed by including additional inputs the new action is also named <*Evolutionary Actions-new*>. Actually, this is also a kind of merged action. However, due to the artificial character, this mediator is rather used as an auxiliary function to support other integration operations, especially Cross Level Integration in chapter 4.2.5.3.

The evolutionary scenario for this integration operation is illustrated in Figure 67. It defines that *Software Specification* will function as input for *System Implementation*, which will consequently change the activity itself. The resulting output artifact *Implemented System* as such will not change since additional input for *System Implementation* came from a finer-grained artifact (*Software Design Specification*). It is assumed that this will, on one hand, refine *Implemented System;* on the other hand, the output from a system level view stays the same.

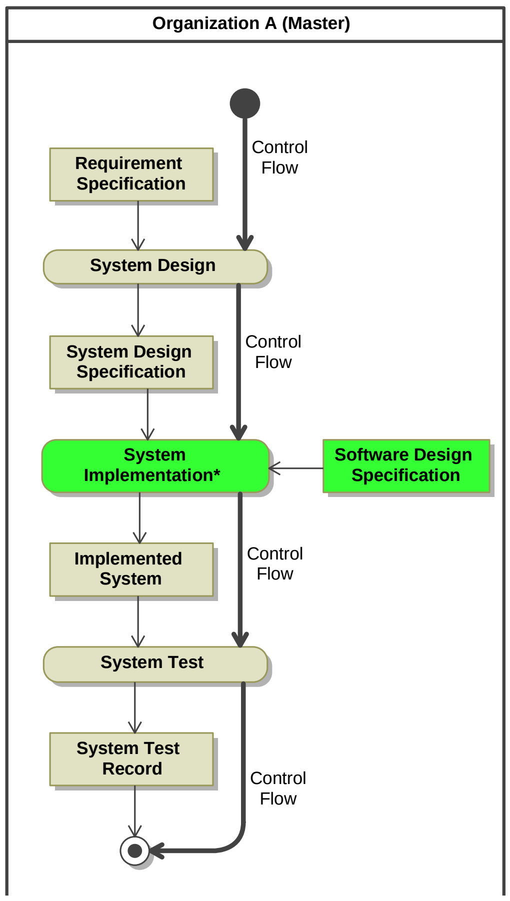


Figure 67: Evolutionary Scenario: 'Merging Integration' (single organization)

4.2.5.3 Cross Level Integration

Having the 'Merging Integration' mediator for single organization defined the following chapter will show how this "supplementary" mediator in Figure 66 (one single organization) is applied to practice.

For cross-level integration, several integration scenarios are possible. Based on the previously described initial processes, the integration functionality within three different integration variants will be shortly discussed in the following. The respective scenarios are kept very simple in order to illustrate the mediator's functionality. Every cross- level integration is actually a combination of two integration steps:

1. Evolutionary Integration (i.e. 'Merging Integration' with one single organization). This is a pre-integration step that follows the mediator in Figure 66.
2. A standard integration scenario, e.g. 'Horizontal Integration' (for handing over a specific artifact). This is the core integration of any other organization's sub-process into the master process as described in chapter 4.2

According to Figure 53 the relevant root processes for this scenario are now those of the Master Organization A and Organization C as the supplying organization. These root processes are again illustrated in Figure 68, showing Organization A has its process defined on a coarser-grained level, i.e., system level, than Organization C. The process definition of

Organization C is on software implementation level, i.e., finer- grained compared to Organization A.

Organization A decides to take advantage from integrating *Software Design Specification* of Organization C. This *Software Design Specification* as such is not defined in the Master's process. The respective integration of the specification takes place in two steps.

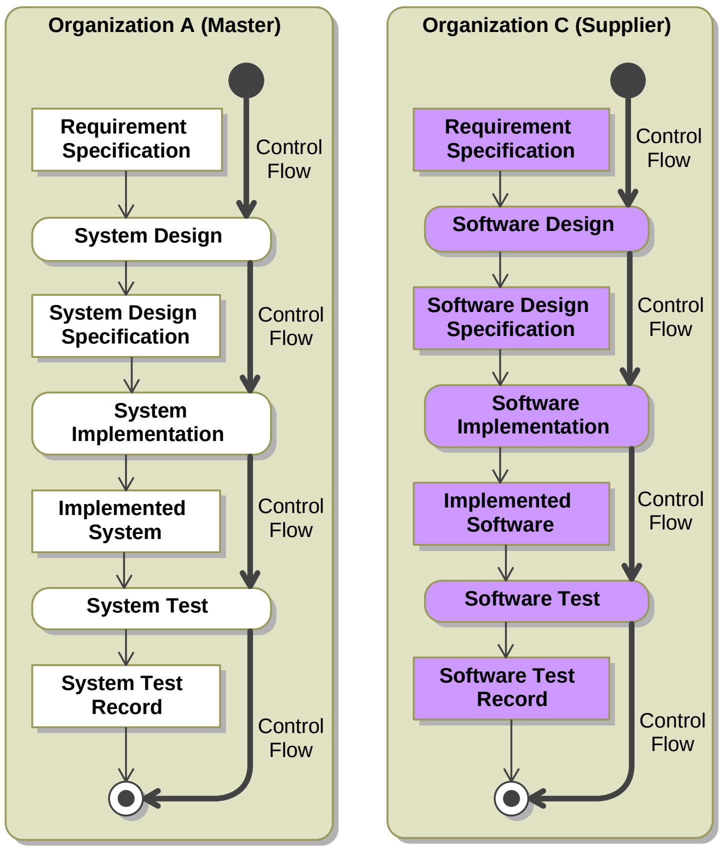


Figure 68: Root process 'Merging Integration' (cross level integration)

Variant 1: Sourcing pre-processing action

Variant 1 of cross- level integration is relevant if some actions/results from a supplier are necessary before any other actions at the Master's site can start. Therefore, it is called the sourcing of pre-processing actions. Prior to execution of a Master's core activity, all necessary inputs for a core activity are collected first. The core activity in Figure 69 is *System Implementation*. Necessary inputs for this core activity are *System Design Specification* and *Software Design Specification*. It is intended that the Master wants to integrate the *Software Design* Process of Organization C.

The evolutionary integration mediator in Figure 66 connects *Software Requirement Specification* from *System Design* that is, therefore, transformed to *System Design*, since the original *System Design* (Master in Figure 68) is not done by the Master alone anymore.

Furthermore, the *Software Design Specification* also requires an evolutionary integration to get connected to *System Implementation*, which is, therefore, converted to *System Implementation*.

The second part encompasses ‘Horizontal Integration’ of the *Software Design* process of Organization C as depicted in Figure 68. The integration method takes *Software Design* – as the process to be integrated – and “flanges” it to both *Software Requirement Specification* at its beginning and *Software Design Specification* at its end.

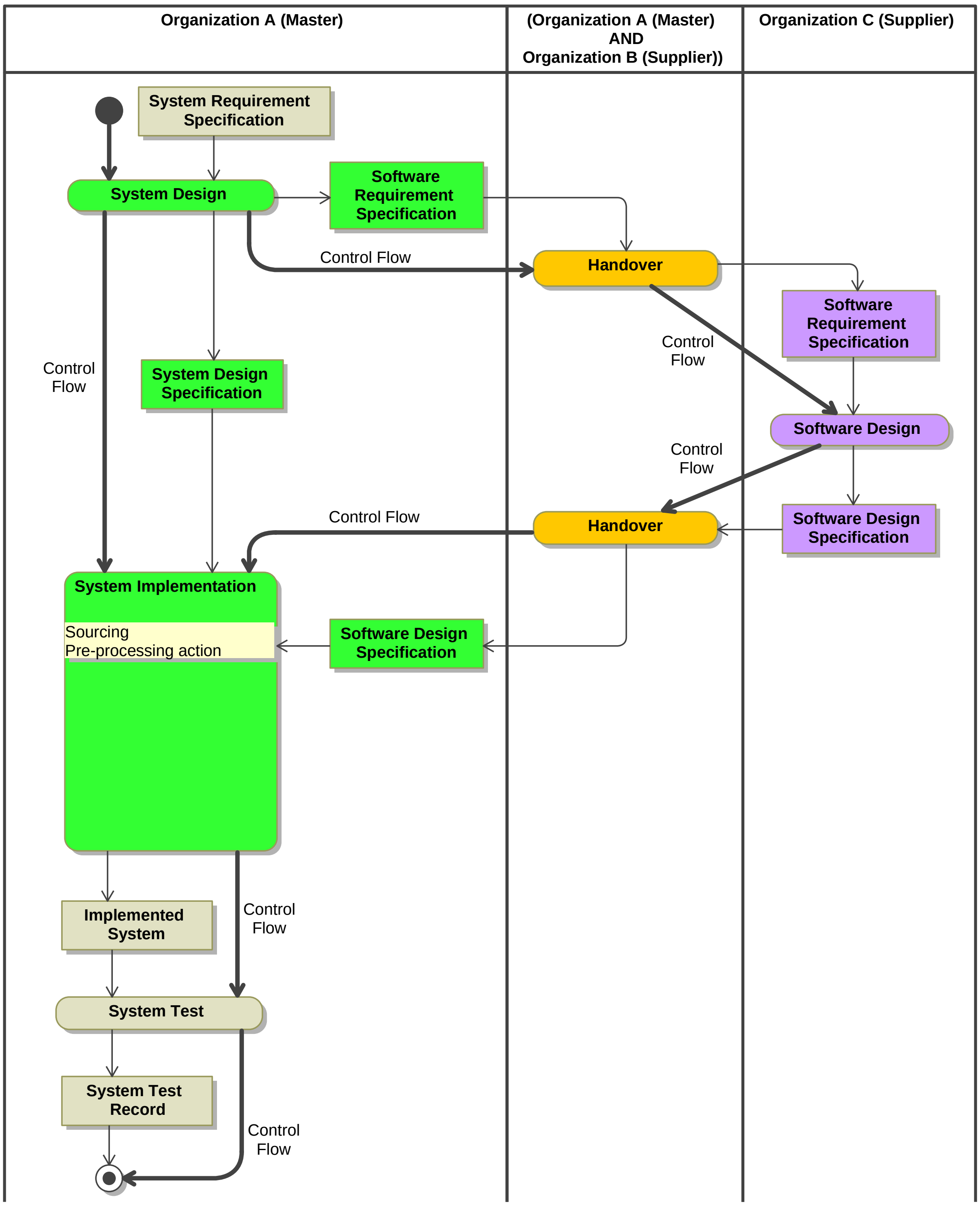

Organization A (Master)
(Organization A (Master) AND Organization B (Supplier))
Organization C (Supplier)
System Requirement Specification
System Design
Software Requirement Specification
Control Flow
Handover
Software Requirement Specification
Control Flow
Control Flow
System Design Specification
Software Design
Control Flow
Control Flow
Handover
Software Design Specification
System Implementation
Sourcing
Pre-processing action
Software Design Specification
Implemented System
Control Flow
System Test
System Test Record
Control Flow

Figure 69: Cross Level Integration: Pre-processing activities

The newly derived *Software Requirement Specification* is necessary to source *Software Design* of Organization C. The *Requirement Specification* transferred through the *Handover* activity of Organization A and C. After *Software Design* is conducted, *Software Design Specification* itself is again converted from the supplier's format (Figure 68) into the Master's format, which has been already created by the evolutionary integration step above.

In parallel, the Master creates his part of *System Design Specification* as a result of *System Design*. Due to different granularity levels, the usage of the already defined scenario 'Additive Vertical Integration' is not possible. Therefore, the type of parallelism in Figure 69 is crucial since *System Implementation* needs input from two different artifacts. Integration of *Software Design Specification* and *System Design Specification* is done within *System Design*.

After Handover of *Software Design Specification,* implementation of the system is started. The *Implemented System* is system- tested via *System Test* resulting in *System Test Record*. This terminates the process.

Variant 2: Sourcing core-processing action

The following variant for cross- level integration (Figure 70) deals with the fact that the Master wants to have any additional functionality integrated during the run of *System Implementation*. This approach is basically the same as within Variant 1.

The evolutionary integration allows for connecting *Software Requirement Specification* (output) and the placeholder <*any output*> as input to the core activity *System Implementation*.

The second step connects via 'Horizontal Integration' the required input from Organization C to the collaborative workflow.

Integration of the input coming from the supplier is again done within *System Implementation*. Again, process parallelism ('Additive Vertical Integration') cannot be applied since distinguished process level granularities.

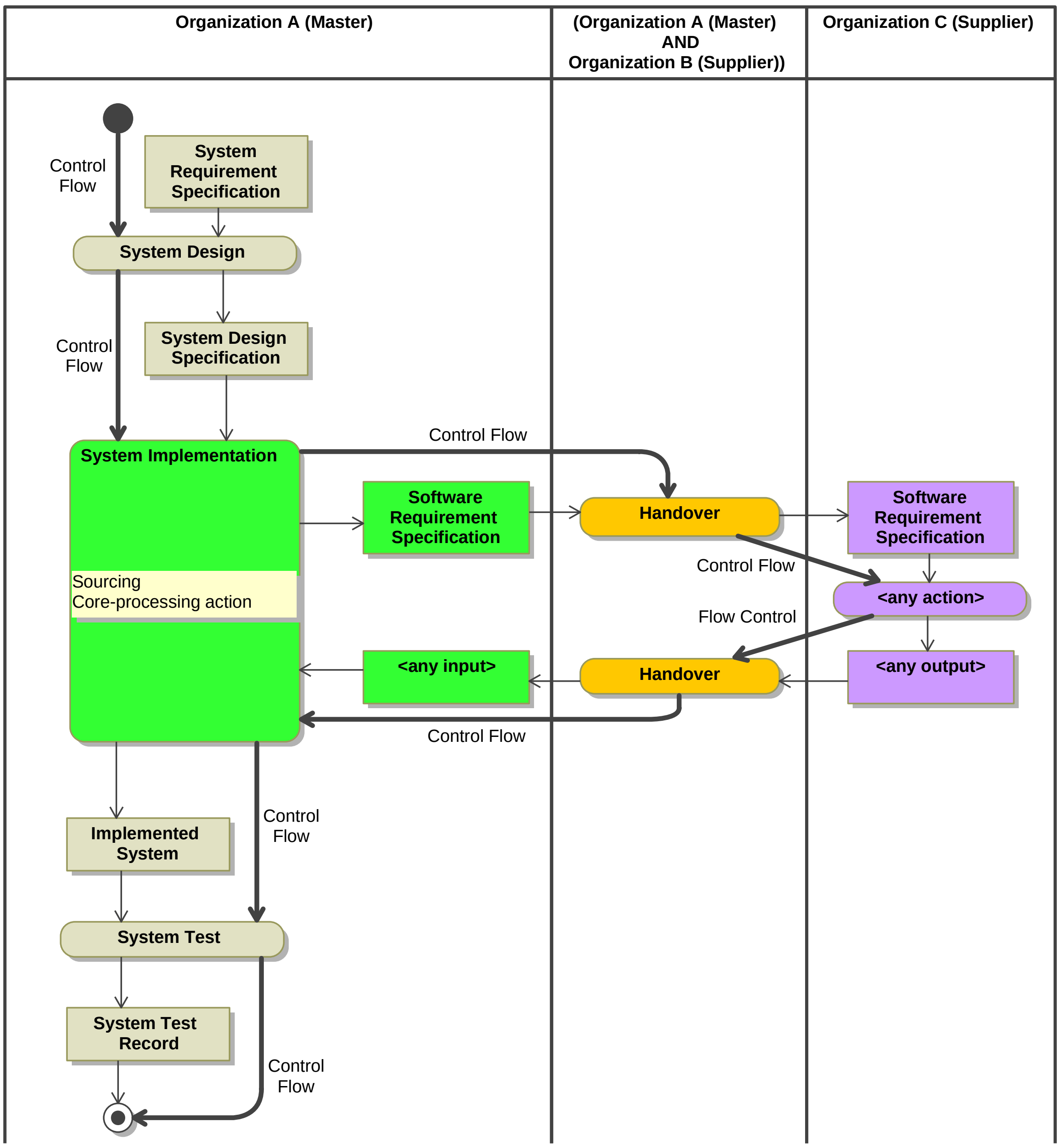


Figure 70: Cross Level Integration: Core-processing activities

Variant 3: Sourcing Post-Processing Action

Figure 71 shows Variant 3 of cross- level integration if the Master organization requires additional inputs towards the end of core activity's execution (*System Implementation*). All required inputs are now collected by *System Test*. However, the basic principle is the same as explained above in detail in Variant 1 and Variant 2.

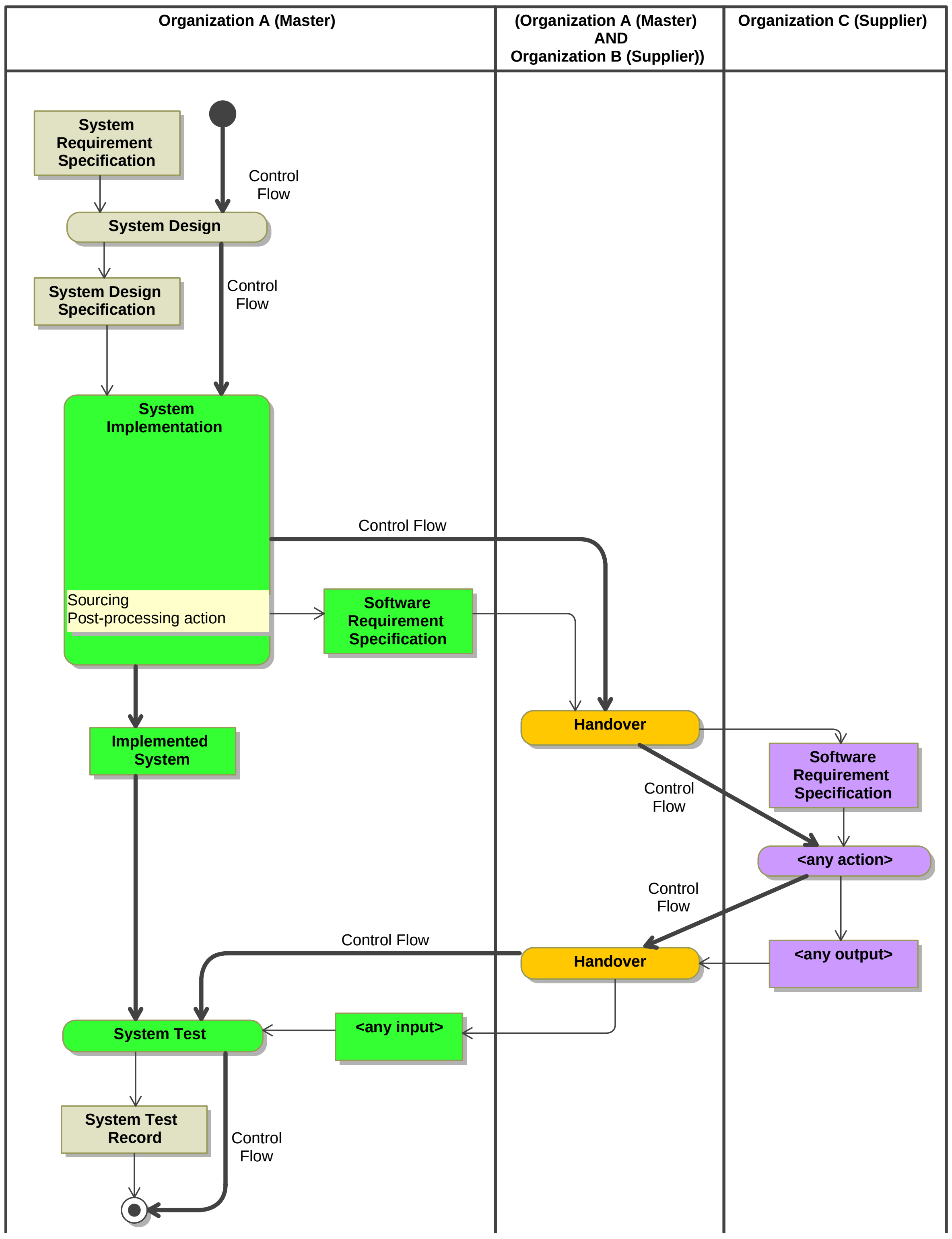


Figure 71: Cross Level Integration: Post-processing activities

4.2.6 Hierarchical Integration

'Hierarchical Decomposition' is an essential structural mean in process modeling activities. This technique primarily promotes complexity handling of organizational processes tremendously and leads towards a better usability in day-to-day work. Furthermore, hierarchical decomposition improves reusability and maintainability. Consequently, process hierarchies are typically used in complex process development environments, including sophisticated software and/or hardware development process. This also contributes to the supportive character of development processes.

The basic methodology and constraints of this dissertation are now crucial for the way of solving the issue of hierarchical process integration. Among others, these constraints are:

- Integration of activities or actions results in a NEW collaborative process (Refer to section 1.3 Scope of the Dissertation)
- Processes to be integrated are recursively defined
- Existing initial processes should not be changed

The following section gives a short definition of the problem to be solved if organizations want to integrate hierarchical processes.

4.2.6.1 Initial Scenario

Organization A (Master) has the hierarchical process *Software Implementation* defined, which is depicted on the left side in Figure 72. Additionally, Organization B (Supplier) has also a hierarchical process *System Implementation* in place. These hierarchical processes can be identified by recognizing the hierarchical action with the trident including the *Software Implementation* process (Master) and *System Implementation* (Supplier). It is assumed that the Master intends to take advantage of buying in *Requirement Specification*, *Software Design Specification* and *Software Implementation* skills into its own system implementation process. This would replace the Master's requirement specification and software realization process. The *Software Design* process is part of a hierarchical process in Organization B (Supplier). The challenge is to include the encircled part of the Supplier's process (in Figure 72) into the placeholder indicated in Organization A's process.

Remembering that hierarchical decomposition in process environments is a mean for a more efficient way to handle process complexity and to support reusability, processes of Organization A and B can also be visualized in a different way by having a different view on it. The derivation of this view is exemplary for Organization B modeled in Figure 73. All three parts are briefly explained in the following.

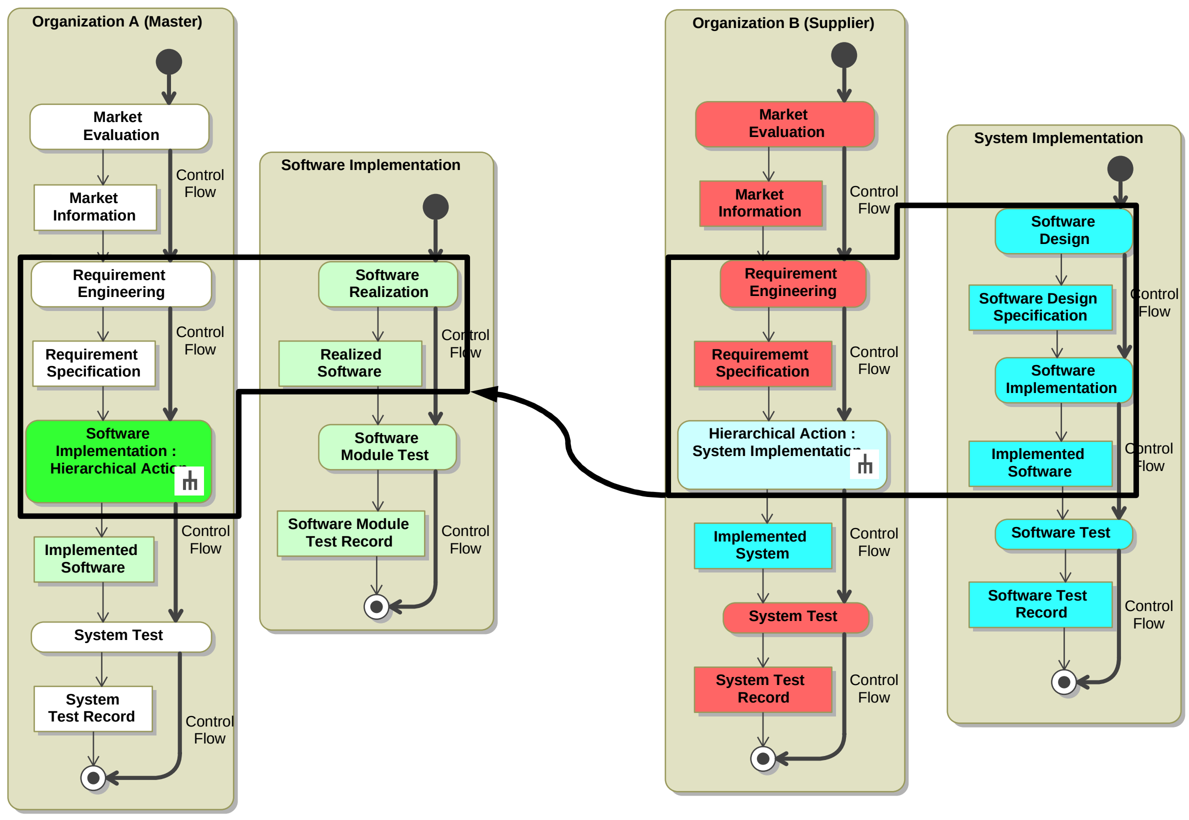


Figure 72: Initial Processes for 'Hierarchical Integration'

b) Without Hierarchy

In this case (Figure 73 b) the hierarchical action is dissolved ("flattened") and control and object flow are defined according UML [100]. Focusing the relevant hierarchical interface portions control flow goes from *Requirement Engineering* to *Software Design* and from *Software Test Record* to *System Test*. Furthermore, data flow is defined from *Requirement Specification* to *Software Design* and from *Software Test Record* to *System Test*.

a) With hierarchy

This case leaves out the entire detailed *System Implementation* sub-process and shows the hierarchical action only. Specifically, the illustration in Figure 73 a) shows the control flow defined from *Requirement Engineering* to the hierarchical action *System Implementation* and additionally from hierarchical action *System Implementation* to *System Test.*

Object Flow definition follows the UML specification [100] respectively; this means that the artifact *Requirement Specification* serves as input for *System Implementation, which* generates the *Implemented System*.

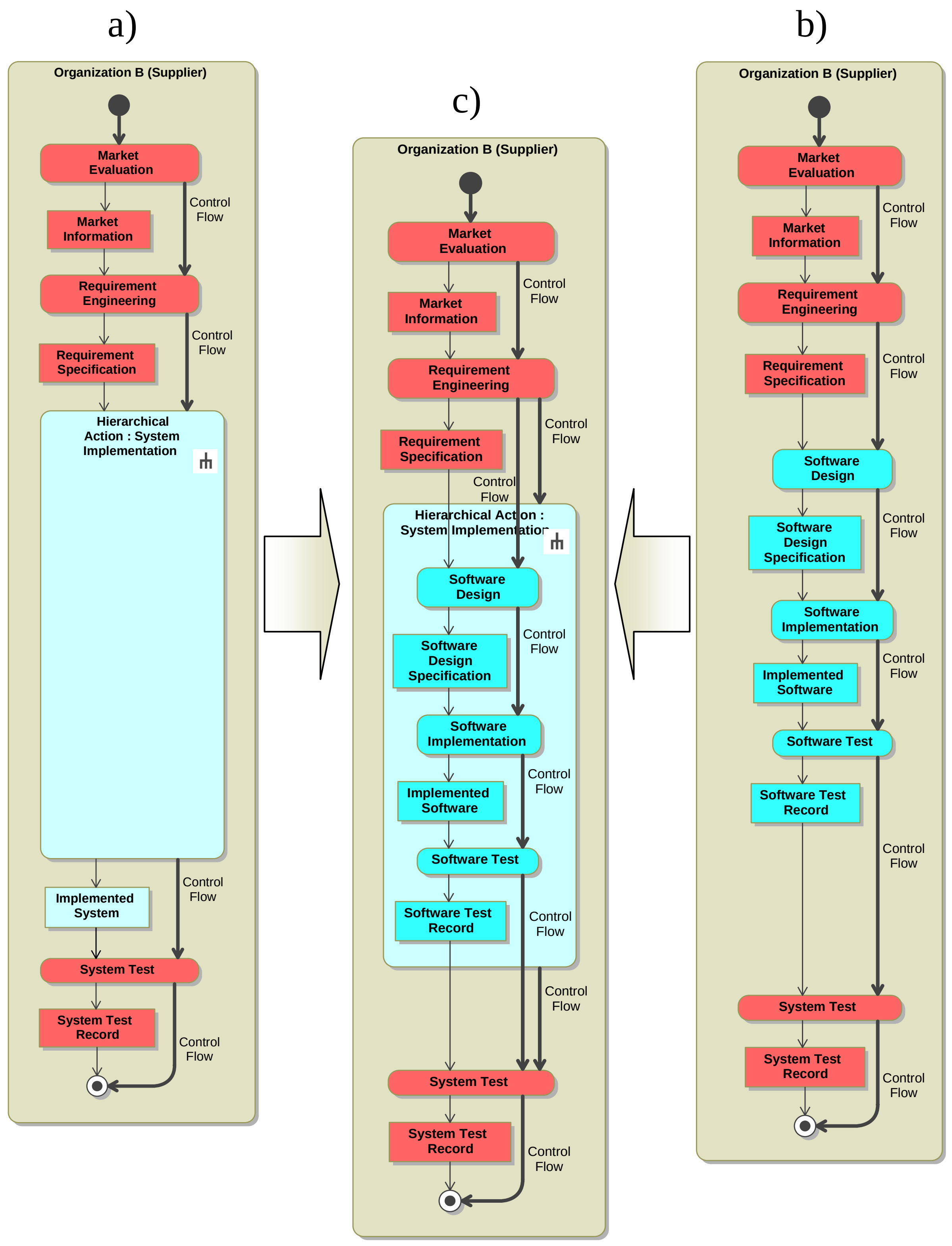


Figure 73: **a) Hierarchical Process;**
b) Dissolved hierarchical process;
c) Dissolved hierarchical process ("Hybrid view")

c) With and without hierarchy ("Hybrid View")

Figure 73 c) now depicts a combination of both a process with and without hierarchy, a so-called "hybrid view". This "hybrid view" is necessary, because the definition of organizational hierarchical processes follow a certain purpose. The process integration approach of this work wants to keep original processes as far as possible, which means that hierarchy should not be given up for setting up collaborative processes. This increases recognition of processes and therefore acceptance within collaborative process users.

For better optical orientation, the control flows arrows are also illustrated in bold. From a control flow perspective the original hierarchy is dissolved, i.e., lower hierarchical levels are brought up to the first process level. However, the hierarchical action still exists (*System Implementation*).

Remarks: The modeling of the hierarchical action does not follow actually the official UML modeling rules of a call behavior action.

The basic modeling concept of the "hybrid-view" in Figure 73 c) has to bring together also the two process variants, with and without hierarchy. This always means that two control flows going into the hierarchical action, i.e., in Figure 73 c) *Requirement Engineering* is connected with the hierarchical action *System Implementation* and with *Software Design* (as the inner part of hierarchical action). Only one object flow connects *Requirement Specification* with *Software Design*. A further object flow between *Requirement Engineering* and *System Implementation* is not necessary, since the hierarchical action *System Implementation* symbolizes just a pointer to the hierarchical sub-process and has consequently no content.

Furthermore, two control flows and one object are coming from the hierarchical action, i.e., *System Implementation* to *System Test* and *Software Design* to *System Test*. The outgoing object flow is defined from *Software Test Record* to *System Test*.

4.2.6.2 Integration Procedure

The hybrid view in Figure 73 c) essential supports integration of actions and activities, because one can use already- defined integration mechanism and scenarios of this work respectively. The hierarchical integration itself is done in several steps, which will be explained in the following.

Dissolution of Hierarchy
The defined hierarchy is first of all dissolved. This means that especially the process parts to be integrated are modeled sequentially. This is depicted in Figure 74.

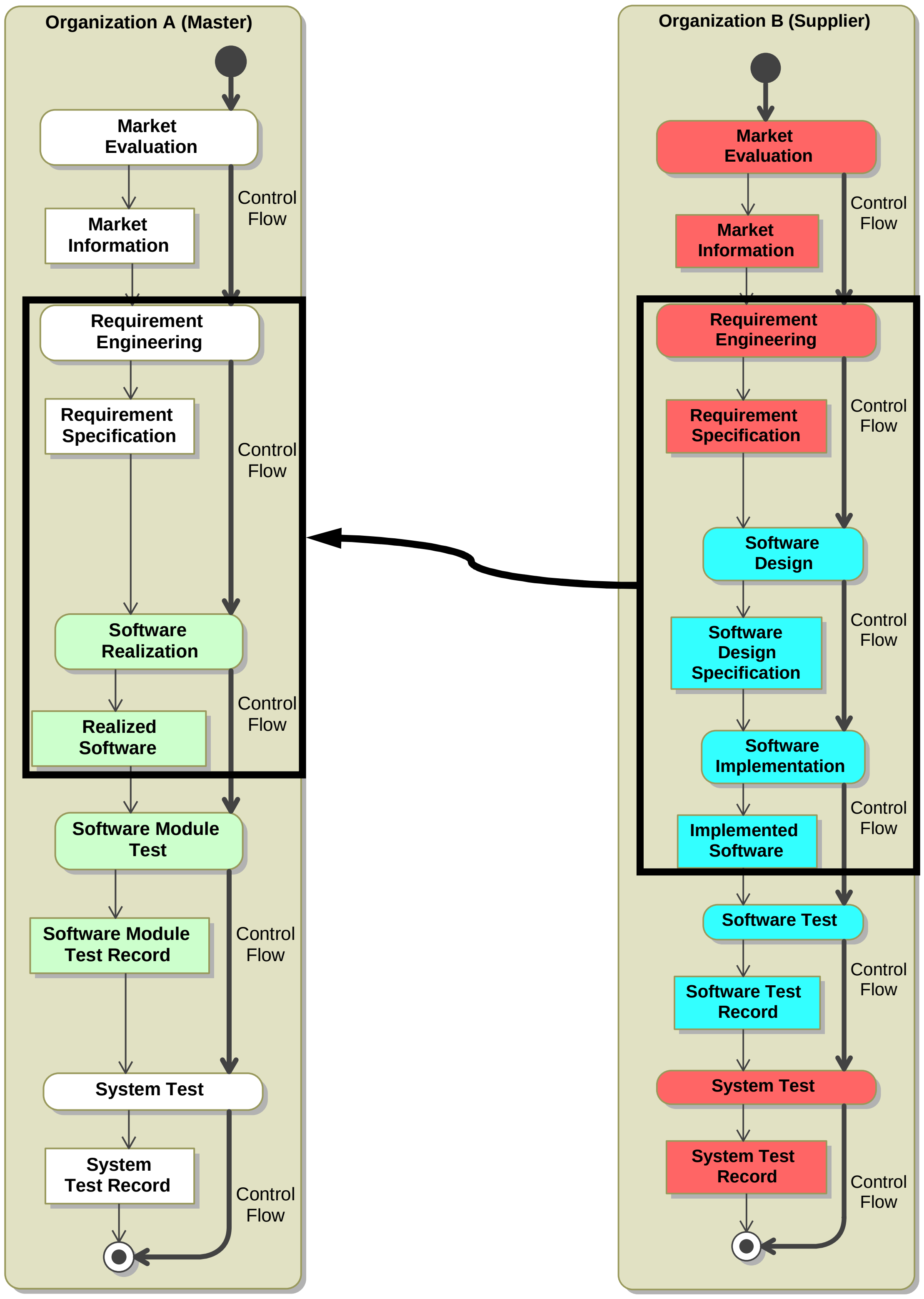


Figure 74: Dissolution of Hierarchies of Organization A (left) and B (right)

The usage of the mediator for ‘Horizontal Integration’ is in this case appropriate (please refer to Figure 55). The collaborative process is illustrated in Figure 75 without having hierarchy incorporated. Different hierarchical levels from various organizations are still depicted with the original color code.

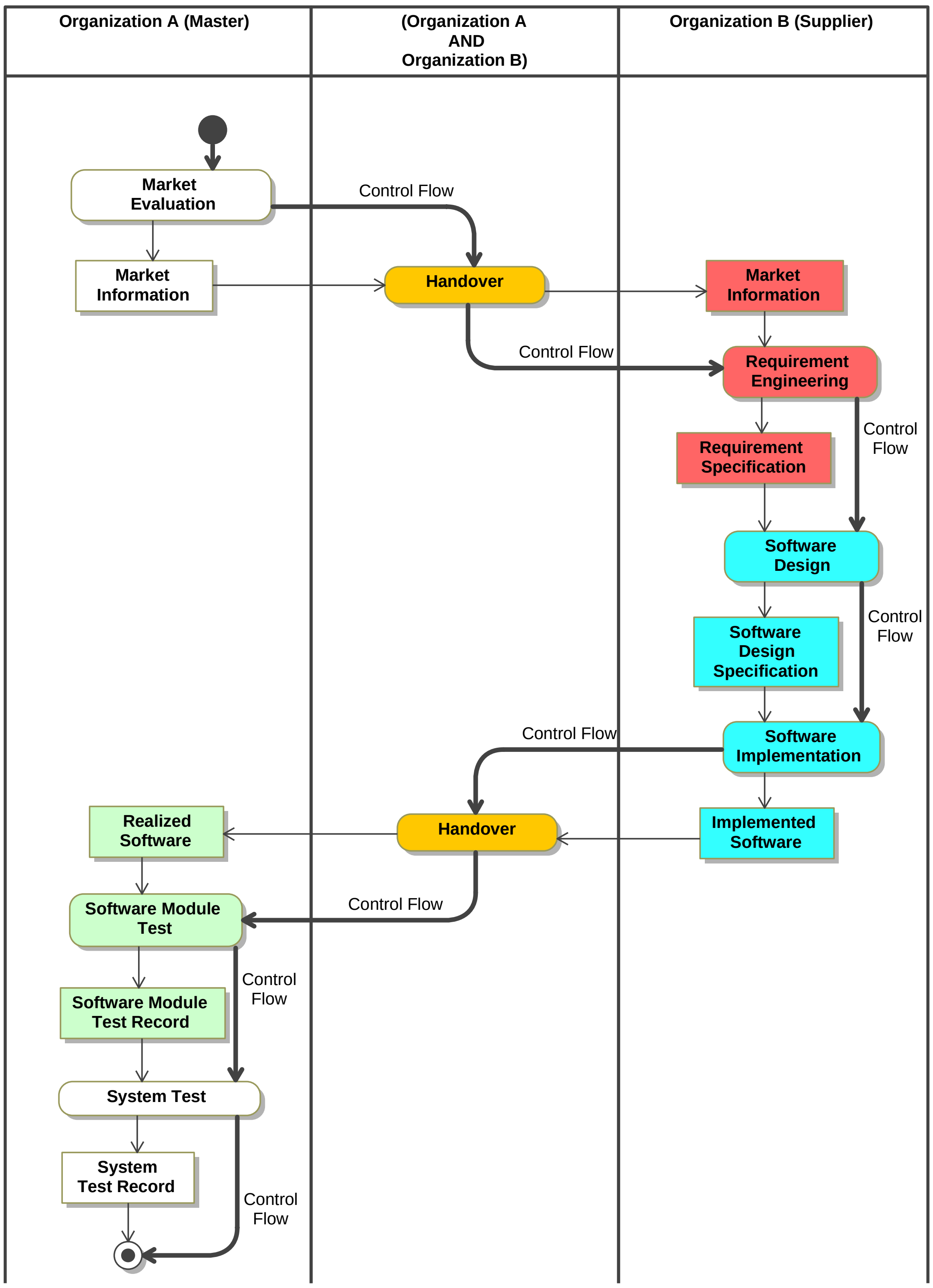


Figure 75: Process Integration by Means of defined Mediators

Based on Figure 75, the new hierarchy shall be defined, which should be in accordance with former hierarchy definitions to get the best possible recognition from process users. In Figure 76, all Software Design, Implementation, and Test actions are assigned to Software *Implementation* as a hierarchical action. This is advantageous for both organizations A and B, since all those actions have been also hierarchically defined prior to integration, which increase recognition and acceptance cross-organizationally.

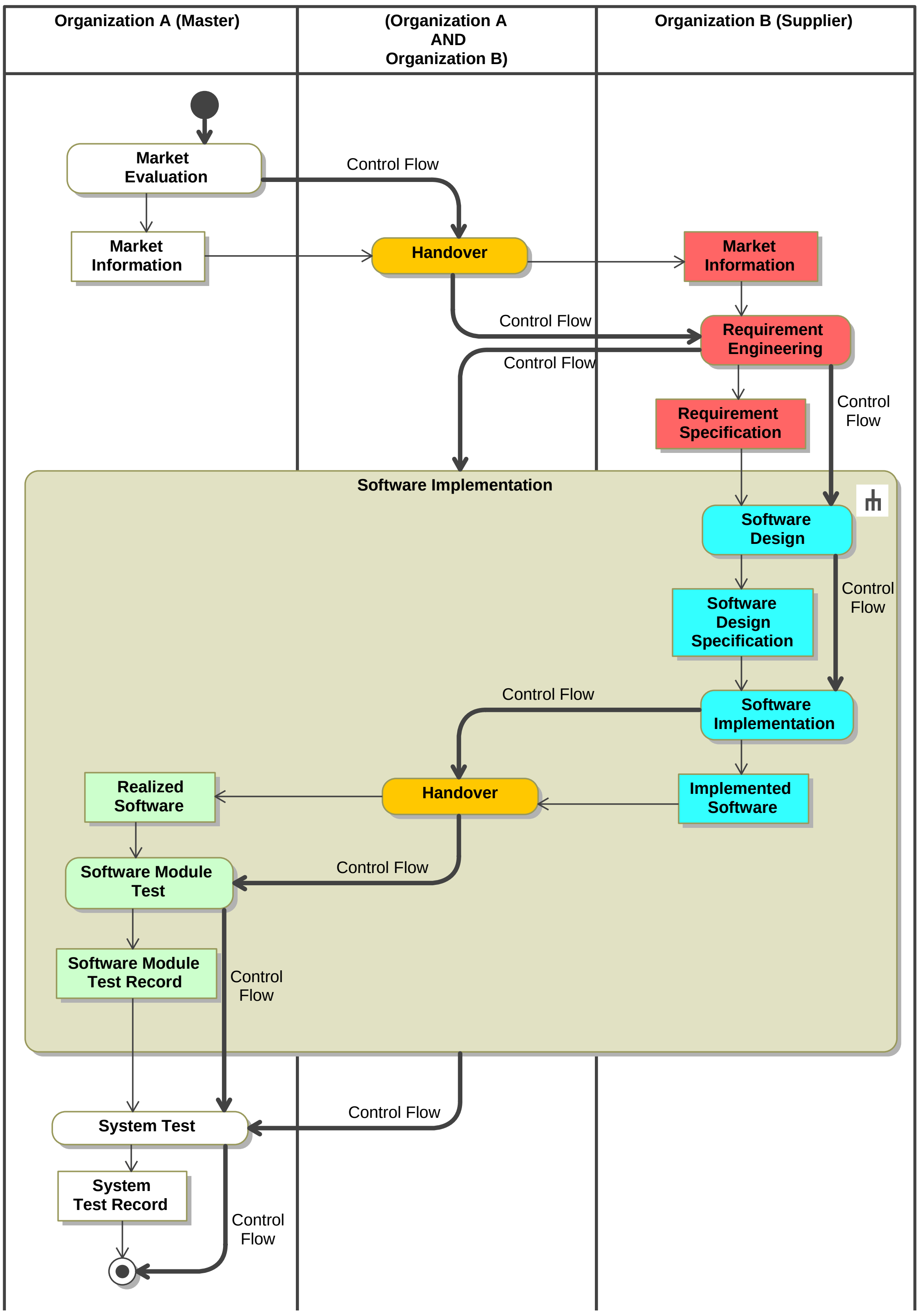


Figure 76: Re-definition of Hierarchy

4.2.6.3 Further Hierarchical Mediator Definitions – Special Cases

This section briefly depicts special mediators for those cases integrating only hierarchical processes partly or entirely from one organizational process exclusively. In this case, the patterns discussed in chapter 4.2 are slightly modified by having those actions to be integrated in hybrid-view.

The mediator for hierarchical integration based on ‘Horizontal Integration’ is illustrated in Figure 77 and is identical to the standard mediator as shown in Figure 47. However, the action to be integrated from Organization B (<Hierarchical Actions>) comes originally from a hierarchical sub-process. This is the reason why those actions are depicted in hybrid-mode in the mediator for building up a collaborative process based on ‘Horizontal Integration’ (Figure 77).

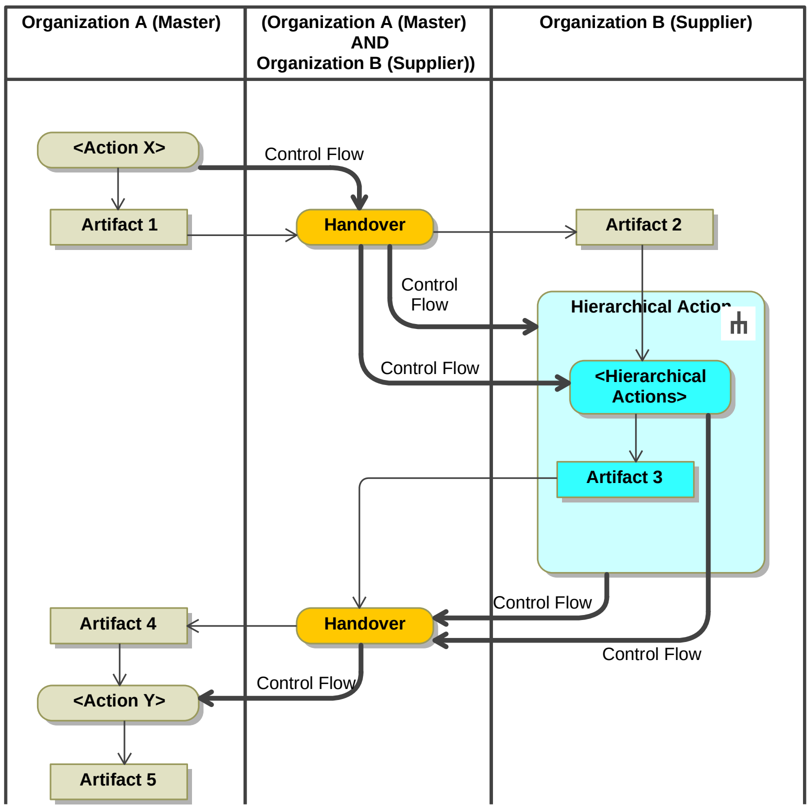


Figure 77: Mediator for 'Hierarchical Integration’ based on ‘Horizontal Integration’

Figure 78 depicts the hierarchical mediator for ‘Additive Vertical Integration’. Remark that Artifact 3 is not defined within the *Hierarchical Action*. The mediator defines two control flows starting from the Fork Node, one towards the *Hierarchical Action* and another one towards the first action of <*Sub Workflow B*>, which is sourced by *Artifact 3*. In turn, there are also two control flows defined into the Join Node starting from *Hierarchical Action* and from the last action of the <*Sub Workflow B*>.

Integration of hierarchical action using ‘Alternative Vertical Integration’ as shown in the mediator pattern in Figure 79 follows the same methodology as the one for ‘Additive Vertical Integration’.

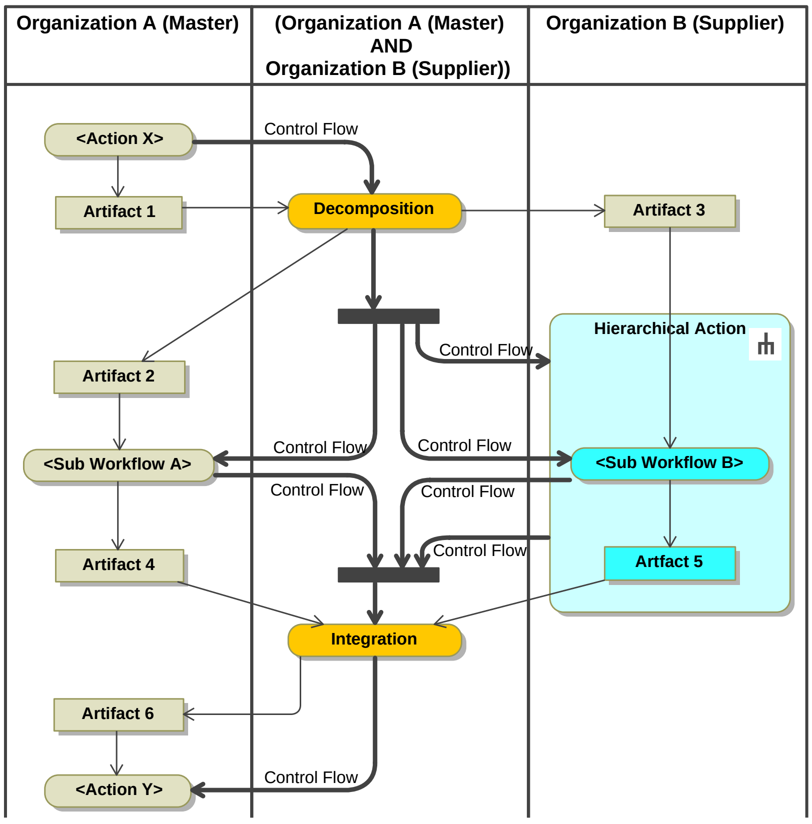


Figure 78: Mediator for ‚Hierarchical Integration' based on 'Additive Vertical Integration'

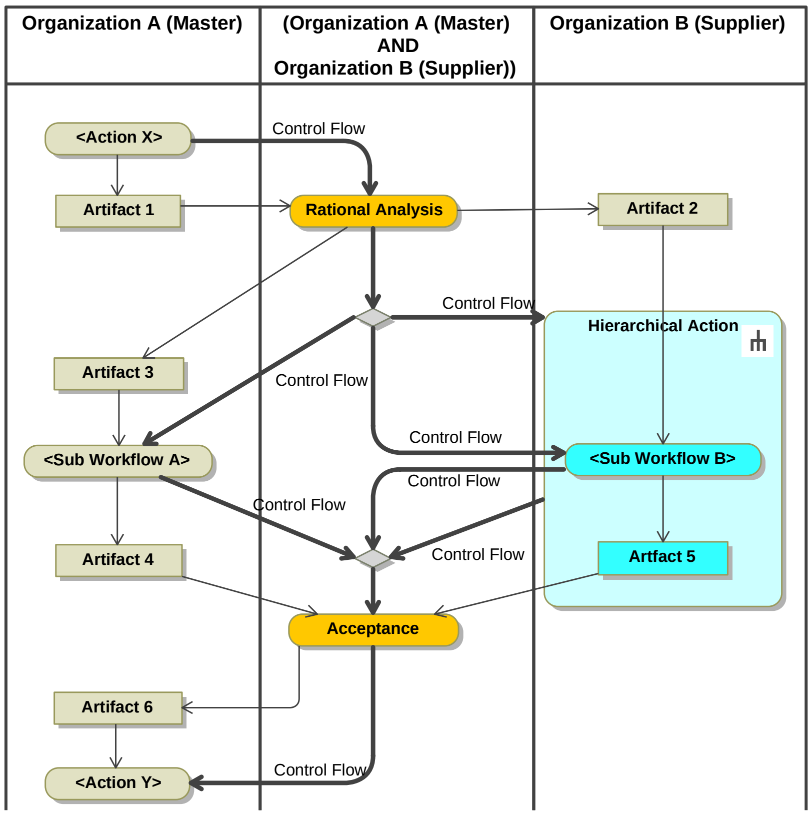


Figure 79: Mediator for 'Hierarchical Integration' based on 'Alternative Vertical Integration'

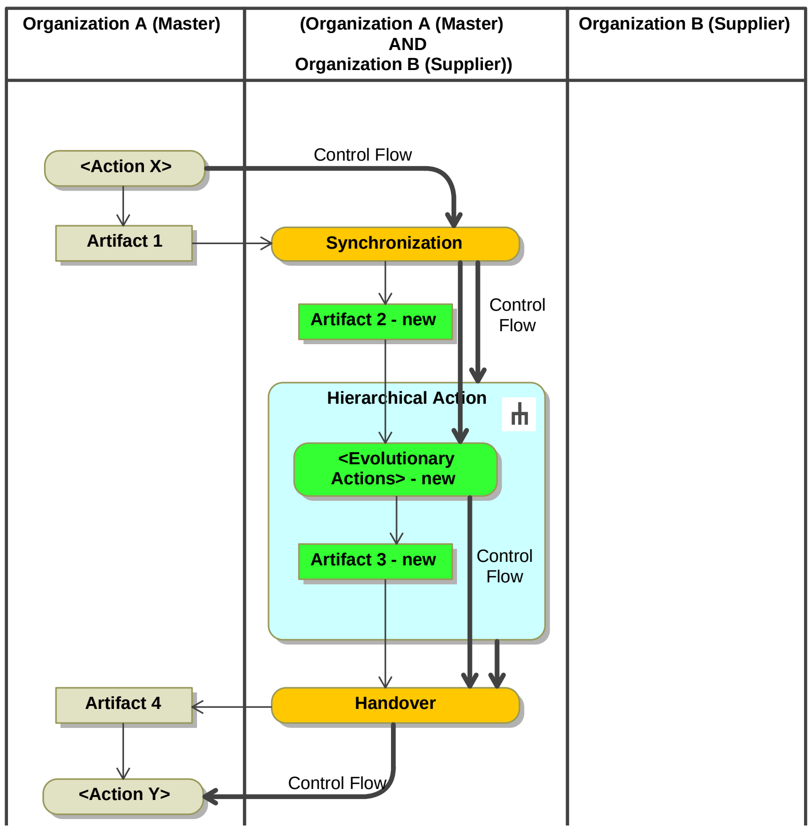


Figure 80: Mediator for 'Hierarchical Integration' based on 'Merging Integration'

In case of having hierarchical actions to be integrated where the pattern from scenario 'Merging Integration' should be used, it is also assumed that – as in the scenarios before – Organization B has hierarchical processes defined. These actions/activities Organization A (Master) want to have integrated and executed jointly as depicted in Figure 80. The remaining methodology of the approach is analog to the other mediator pattern for hierarchical process integration.

General remarks: All scenarios within the introduced special cases could also be defined vice versa, if Organization A had a hierarchical (sub-) process defined. For this scenario, the *Hierarchical Activity* in the mediator (Figure 79) would have shifted to the Master.

4.2.7 Alternative Approaches for Software Development Collaborations

Besides the mediator pattern approach for process integration, there are some other ways and methodologies to collaborate. In the following, these possibilities are briefly introduced and discussed.

Artifact-centric approach

An artifact based ("artifact-centric") process approach is a methodology, which is also often used in practice to set-up collaborations and cooperation between various parties. The core idea addresses the fact that several artifacts are necessary for any collaboration/cooperation to develop a product. These artifacts are driven by business data or business entities. In contrast to a process-or activity centric approach, this approach describes how these data are updated or changed by typically using a status model. Figure 81 shows that several domains are interested in artifacts and contribute to them towards finalization [152].

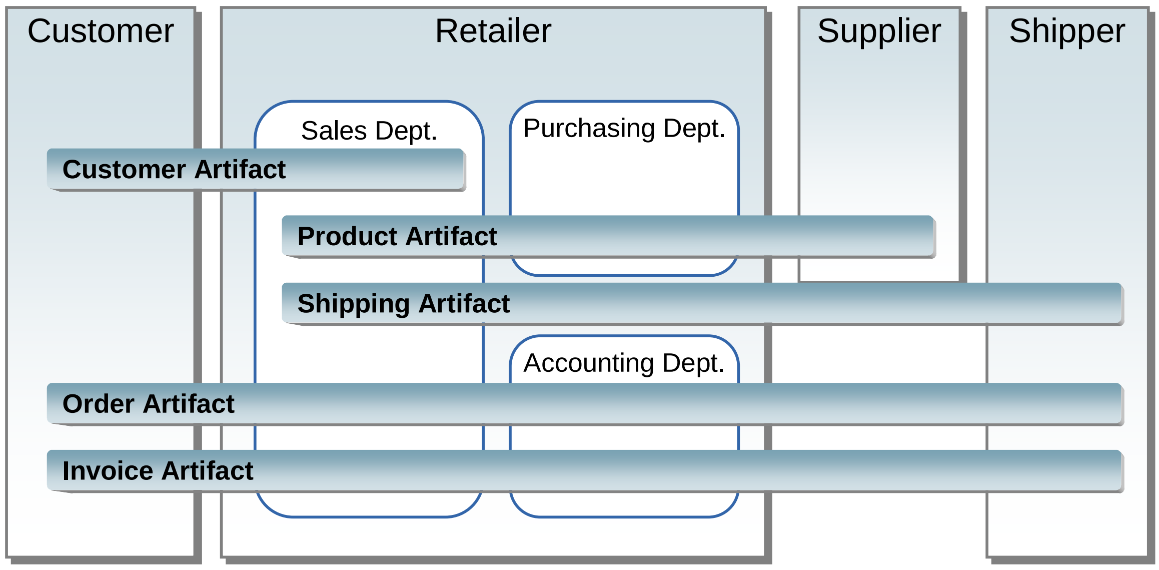


Figure 81: Business artifacts in collaborative business processes

Dependencies regarding process control flow between collaborating organizations are not obviously defined, since there is no process landscape institutionalized. Processes are amorphous from an activity perspective, i.e., they are not explicitly described and documented. This fact makes it very difficult to define proper process integration points to make sure that artifacts are available at critical points (activities) in a workflow.

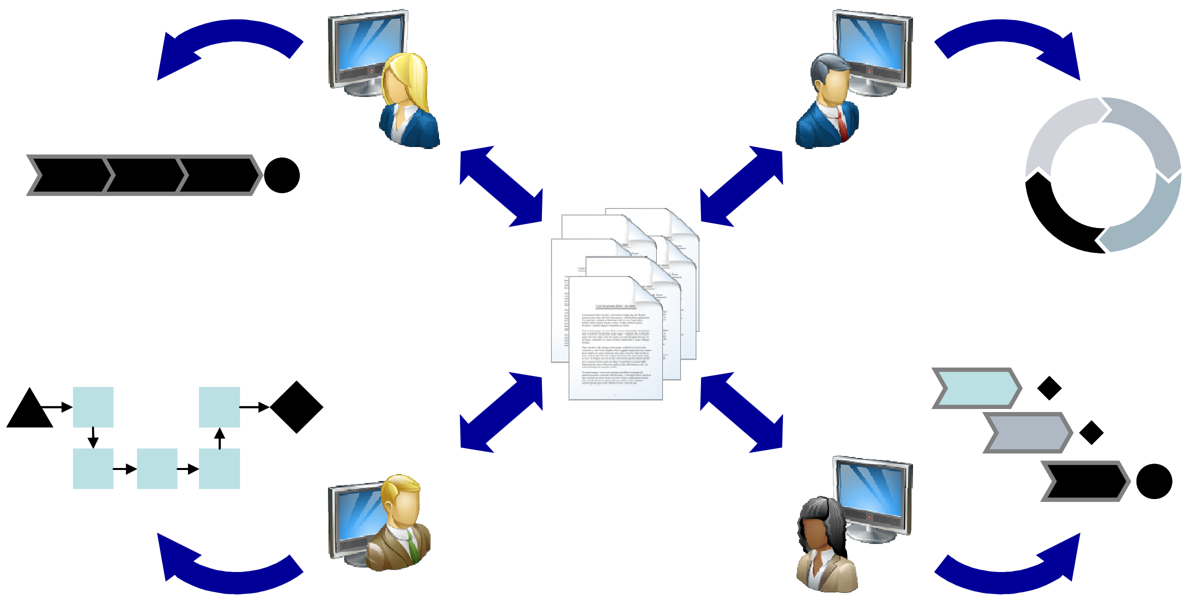


Figure 82: Artifact oriented development approach

Compared to structured collaborative process integration (e.g. pattern-based like in this dissertation) no defined connection points are set up when following this approach. One could argue that an activity diagram as used in this collaborative process integration approach is also able to model artifacts by the means of an object flow. However, one process step might create more than one artifact. This makes it very challenging to define collaboration processes that illustrate all dependencies and simultaneously supporting all organizations by a clearly represented process. Additionally, the artifact centric approach does not use the advantage of a Master organization, which is a coordinating instance. Figure 82 depicts that approach and illustrates that no coordinating instance is defined.

Ad hoc drawing approach
This approach uses trivial means for defining and creating collaborative processes, i.e., any drawing tool, which might also produce the goal of being able to create proper process for development collaborations. However, this kind of instant or ad hoc approach does not provide benefits (e.g. quick and structured process set-up through pattern availability) of structured pre-defined interfaces an organization can rely on during process definition and execution. Especially, if processes and interfaces in further projects are defined once again, they will slightly differ from the one in a previous project. This negatively affects the process consistency and stability of organizational workflows, especially if projects turn more complex and complicated.

4.3 Formalization

Collaborative scenarios and patterns are useful means to get a process set up for globally-defined project, especially with dispersed development teams. These scenarios visualize processes executed during development run.

A major challenge of process handling is the definition phase. In order to provide hands-on support for collaborative process definition, the entire procedure shall be subject to automation. For automation purposes, it is crucial to define exactly how integration approach works. Therefore, the activity diagram- based representations need to be formalized. In order to adequately address this issue, a graph- based representation has been chosen, which is the most intuitive and obvious way to formalize UML activity diagrams.

Generally, an activity diagram consists of nodes and edges. An edge is described as a pair of nodes. A node is either a 3-tupel consisting of an identifier (i), a type (t), and a role (r). 4-tuple nodes additionally encompass or point to a (hierarchical) activity diagram *ad*.

A node ND$\in$AD is defined in Definition 1. AD is defined according Definition 1.

Definition 1.

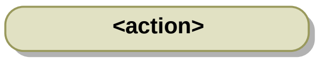
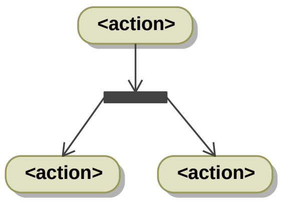
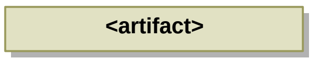
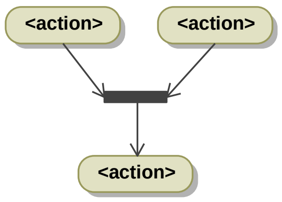
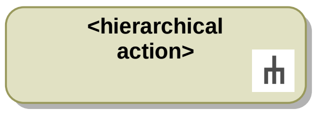
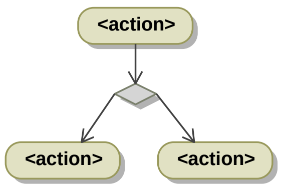
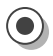
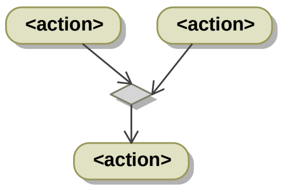
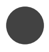
$$ND =_{DEF} \left(ND_{id} \times ND_{type} \setminus \{\mathrm{y}_{\mathrm{ha}}\} \times < Role > \right) \cup \left(ND_{id} \times \{y_{ha}\} \times < Role > \times AD\right)$$

where

- ND_{id} is the infinite set of all node identifiers
- the set of *ROLE* defined in chapter 4.3.1
- $ND_{type} =_{\mathrm{DEF}} \{ \mathrm{y}_{\mathrm{action}}, \mathrm{y}_{\mathrm{ha}}, \mathrm{y}_{\mathrm{artifact}}, \mathrm{y}_{\mathrm{start}}, \mathrm{y}_{\mathrm{end}}, \mathrm{y}_{\mathrm{fork}}, \mathrm{y}_{\mathrm{join}}, \mathrm{y}_{\mathrm{branch}}, \mathrm{y}_{\mathrm{merge}} \}$ is the finite set of all node types (for visualization please also refer to Table 9)
- *AD* is the infinite set of activity diagrams defined in Definition 2

A node ND is defined either as 3-tupel or 4-tupel, depending on whether a concrete node *nd* is a hierarchical node. Consequently, the cross product “ND_{id} x {y_{ha}} x ROLE x AD“, is a 4-tupel, because the considered node AD references another (sub-) activity diagram. Thereby, Nd_{type} is automatically defined as {y_{ha}} according to Definition 1.
For better illustration, the finite set of all possible node types ND_{type} is listed in Table 9:

Table 9: Finite Set of Possible Node Types and Role Relationships

y_{action}		y_{fork}	
$y_{artifact}$		y_{join}	
y_{ha}		y_{branch}	
y_{start}		y_{merge}	
y_{end}			

The intersection set operation is defined as:

Definition 2.
$\bigcap =_{DEF} AD \times AD \rightarrow AD$

The concrete application of this operation ensures that ad_1 and ad_2 have no node in common:

Definition 3.
$\bigcap(ad_1 = (Nd_1, Ed_1), ad_2 = (Nd_2, Ed_2)) = (Nd_1 \cap Nd_2, Ed_1 \cap Ed_2)$

AD is defined as the infinite set of all possible existing activity diagrams. AD^{FLAT} is the set of all non-hierarchical activity diagrams.

Definition 4.

$$AD^{FLAT} =_{DEF} \left\{ \begin{array}{l} (Nd \subseteq ND, Ed \subseteq Nd \times Nd) \mid \\ \forall (i,t_1,r_1),(i,t_2,r_2) \in Nd : (t_1 = t_2) \wedge (r_1 = r_2) \wedge t_1 \neq y_{HA}, \\ Nd \ finite \end{array} \right\}$$

$$AD =_{DEF} \left\{ \begin{array}{l} ad \mid ad \in AD^{FLAT} \\ \vee \left(\begin{array}{l} ad = \overrightarrow{create\_hierarchy}(ad', ad'_{sub}, id, r, Ed_{A-TO-AS}, Ed_{A-FROM-AS}) \\ \qquad \wedge ad' \in AD \\ \qquad \wedge ad'_{sub} \ is \ a \ closed \ sub-graph \ of \ ad' \ related \ by \ ED_{A-To-As}, Ed_{A-FROM-As} \\ \qquad \wedge id \ is \ an \ identifier \ unused \ in \ ad' \\ \qquad \wedge r \in < ROLE > \end{array} \right) \end{array} \right\}$$

By having processes from two or more different organizations defined, it is important that the used actions are uniquely used throughout the cross organizational set of activity diagrams *Ad*. However, the repetitive use of any node is allowed by calling an activity diagrams *AD* recursively independently from depth of interlacing. Additionally, it is allowed that nodes used in lower level of interlacing, i.e. sub-graphs like *As*, are part of the set upper levels (e.g. *A*): *As* $\subseteq$ *A*. Definition 5 uses the function $\overrightarrow{create\_hierarchy}$, which is generally defined as:

Definition 5.

$$\overrightarrow{create\_hierarchy} : AD \times AD \times ND_{id} \times < ROLE > \times \wp(ND \times ND) \times \wp(ND \times ND) \rightarrow AD \cup \{\otimes\}$$

The symbol "⊗" defines a non-valid return value. A valid hierarchical process definition is ensured if

- *AD* is valid and
- the function $\overrightarrow{create\_hierarchy}$ creates again a valid activity diagram *AD* after hierarchy re-definition

A more detailed definition of $\overrightarrow{create\_hierarchy}$ is given in the following.

$$\overrightarrow{create\_hierarchy} := (ad_A = (Nd_A, Ed_A), ad_{As} = (Nd_{As}, Ed_{As}), id, role, Ed_{A-TO-As}, Ed_{A-FROM-As}) =$$

$$\left\{ \begin{array}{ll} \left(\begin{array}{l} Nd_A \cup \{ nd_{HA} = (id, tpye = y_{HA}, role, ad_{As}) \}, \\ Ed_A \cup \bigcup\limits_{(nd_{TO}, \overline{nd}_{TO}) \in Ed_{A-TO-As}} (nd_{TO}, \overline{nd}_{HA}) \cup \bigcup\limits_{(nd_{FROM}, \overline{nd}_{FROM}) \in Ed_{A-FROM-As}} (nd_{HA}, \overline{nd}_{FROM}) \end{array} \right) & \\ \quad \bullet \mathrm{ad_{As} \ is \ closed \ sub\text{-}graph \ of \ ad_A \ related \ by \ Ed_{A\text{-}To\text{-}As}, Ed_{A\text{-}FROM\text{-}As}} & \\ \quad \bullet \mathrm{id \ is \ an \ unused \ identifier \ in \ ad_A} & \\ \quad if \left(\begin{array}{l} \bigcup\limits_{(nd_{TO}, \overline{nd}_{TO}) \in Ed_{A-TO-As}} \overrightarrow{partial\_graph}(ad_A, \overline{nd}_{TO}, \overrightarrow{successor}) \cap \\ \qquad \bigcup\limits_{(nd_{FROM}, \overline{nd}_{FROM}) \in Ed_{A-FROM-As}} \overrightarrow{partial\_graph}(ad_A, nd_{TO}, \overrightarrow{predecessor}) \end{array} \right) = Nd_{As} & \\ \otimes \quad otherwise & \end{array} \right.$$

where

- ad_A represents the entire activity diagram prior to hierarchy re-definition
- ad_{As} represents an activity diagram referenced by the new hierarchical action nd_{HA}
- *id* is the node ID of the hierarchical action (HA) after hierarchy re-definition
- *role* is the role of the new hierarchical action
- $Ed_{A-TO-As}$ is the set of edges to identify the entry node of the hierarchical graph ad_{As}
- $Ed_{A-FORM-As}$ is the set of edges to identify the exit node of the hierarchical graph ad_{As}

For better illustration of the meaning of each parameter within $\overrightarrow{create\_hierarchy}$ all function arguments are graphically depicted in Figure 83.

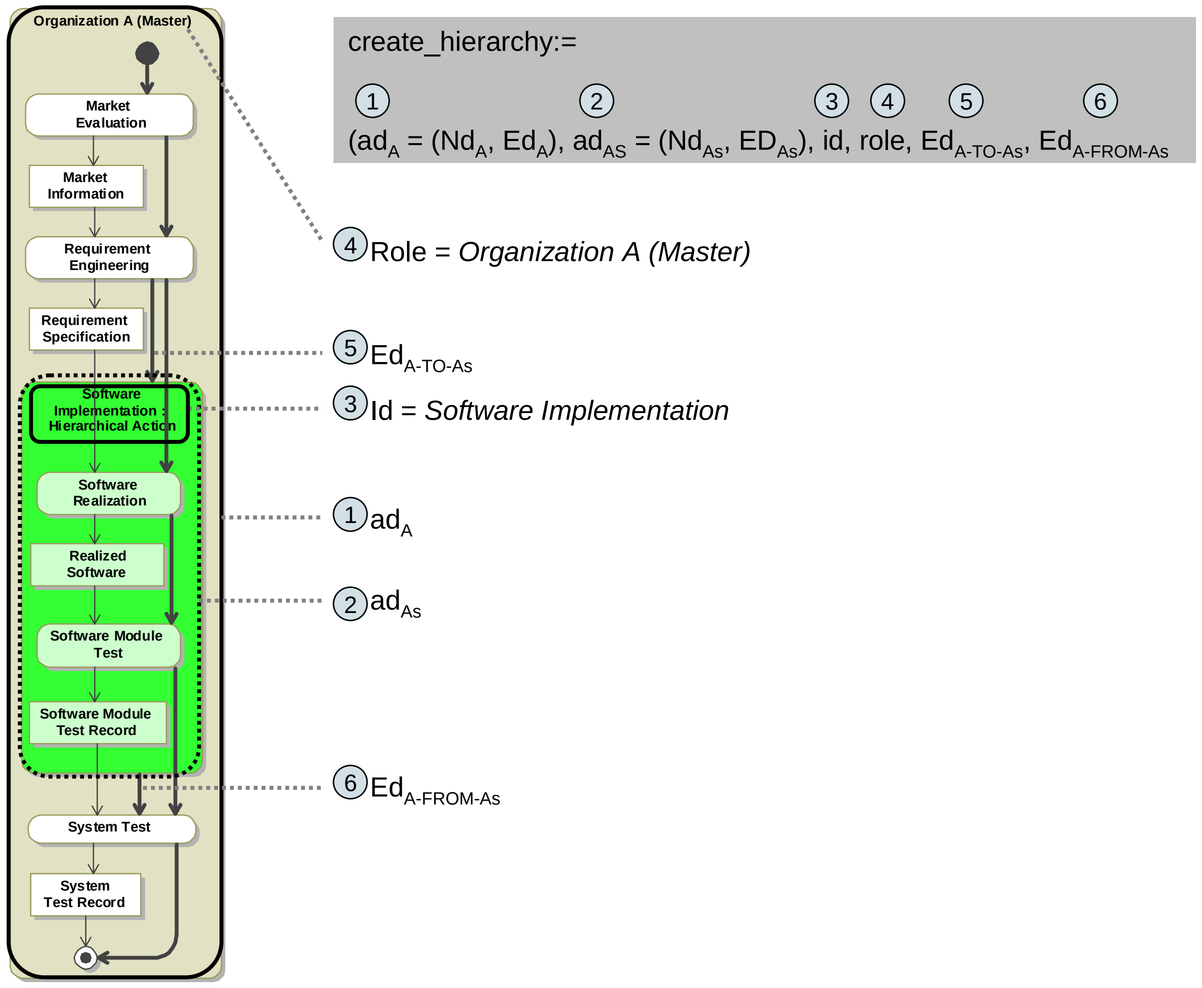

Organization A (Master)
Market Evaluation
Market Information
Requirement Engineering
Requirement Specification
Software Implementation : Hierarchical Action
Software Realization
Realized Software
Software Module Test
Software Module Test Record
System Test
System Test Record
create_hierarchy:=
1 2 3 4 5 6
(ad_A = (Nd_A, Ed_A), ad_AS = (Nd_As, ED_As), id, role, Ed_A-TO-As, Ed_A-FROM-As
4 Role = Organization A (Master)
5 Ed_A-TO-As
3 Id = Software Implementation
1 ad_A
2 ad_As
6 Ed_A-FROM-As

Figure 83: Parameter explanation of $\overrightarrow{create\_hierarchy}$ function

That part of the hierarchical graph ad_{As}, which will be integrated, shall be a *closed_sub_graph* of ad_{As} with $Ed_{A\text{-}TO\text{-}As}$ and $Ed_{A\text{-}FROM\text{-}As}$ as the respective set of edges going into and coming from ad_{As}.

In order to get validity of ad_{As} approved, the auxiliary function $\overrightarrow{partial\_graph}$ is used. This function creates the concrete hierarchical graph ad_{As} by collecting the set of successor and predecessor of one node.

In Definition 6, $\overrightarrow{partial\_graph}$ is recursively defined and maps to the partition set $\wp$ of ND.

Definition 6.

$$\overrightarrow{partial\_graph} : AD \times ND \times \{\overrightarrow{predessesor}, \overrightarrow{succesor}\} \to \wp(ND)$$

$$\overrightarrow{partial\_graph}(ad = (Nd, Ed), nd, \vec{f}) = \begin{cases} \{nd\} & , \ if \ \vec{f}(ad, nd) = \emptyset \\ \{nd\} \cup \bigcup_{\overline{\overline{nd}} \in \vec{f}(ad,nd)} \overrightarrow{partial\_graph}(ad, \overline{\overline{nd}}, \vec{f}), & otherwise \end{cases}$$

The function $\vec{f}$ takes as input the concrete activity diagram (hierarchical graph) *ad* and any node *nd* $\in$ *ad*. The set of Roles does not need to be explicitly mentioned in the set of parameters within $\overrightarrow{partial\_graph}$, since it is implicitly defined through the set of nodes ND in Definition 1.
The function $\overrightarrow{successor}$ maps the corresponding nodes to the partition set $\wp$ of ND and collects all nodes in the set union $\bigcup \overline{nd}$.

Definition 7.

$$\overrightarrow{successor} : AD \times ND \to \wp(ND)$$

$$\overrightarrow{successor}(ad = (Nd, Ed), nd) = \bigcup_{(nd, \overline{nd}) \in Ed} \overline{nd}$$

The function $\overrightarrow{predessesor}$ maps the corresponding nodes to the partition set $\wp$ of ND and collects all nodes in the set union $\bigcup nd$.

Definition 8.

$$\overrightarrow{predecessor} : AD \times ND \to \wp(ND)$$

$$\overrightarrow{predecessor}(ad = (Nd, Ed), \overline{nd}) = \bigcup_{(nd, \overline{nd}) \in Ed} nd$$

4.3.1 Role Definition

The syntax to define a role descriptor is as follows:

Definition 9.

$$< ROLE > := \left(< ROLE >< OpID >< ROLE >\right) \, \| < SINGLE\_ROLE >$$

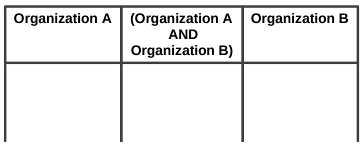
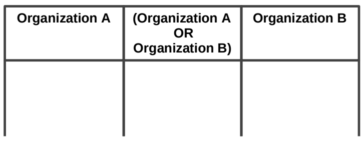
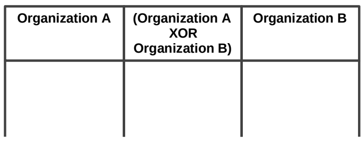
where

- <*OpID*> := AND || OR || XOR
- <*SINGLE_ROLE*> := ([{‘A’-‘Z’}], [{‘a’-‘z’}], [{‘0’-‘9’}])*

<SINGLE_ROLE> accepts values that follow the Extended Backus-Naur-Form (EBNF) notation [73]. The formalization mechanism has to use parentheses as a mean for role groupings. For

instance, *Org_A* **AND** ***(****Org*_B **OR** *Org_C****)*** is not equal to ***(****Org_A* **AND** *Org_B****)*** **OR** *Org_C*. The finite set of all possible role types is listed in Table 10:

Table 10: Finite Set of Possible Role Connectors

AND (relationship)	 Organization A (Organization A AND Organization B) Organization B	OR (relationship)	 Organization A (Organization A OR Organization B) Organization B
XOR (relationship)	 Organization A (Organization A XOR Organization B) Organization B		

4.3.2 Closed Sub-graph Definition

Differentiation between processes and sub-processes is crucial for formalizing integration operations used in this dissertation. In general, a closed sub-graph defines those nodes and edges that are to be integrated in another process.
The notion of a so- called '*closed sub-graph*' is defined in Definition 10:

Definition 10.

$$As \in AD\, is\_closed\_sub\_graph\_of\ A \in AD \Leftrightarrow Nd_{As} \subseteq Nd_{A} \wedge Ed_{As} \subseteq Ed_{A}$$

Figure 84 illustrates this formalism in general and relates Graph (A) to Sub-graph (As) in an activity diagram. In this case *As* is exactly the set of those nodes that will be integrated into another organization's process.

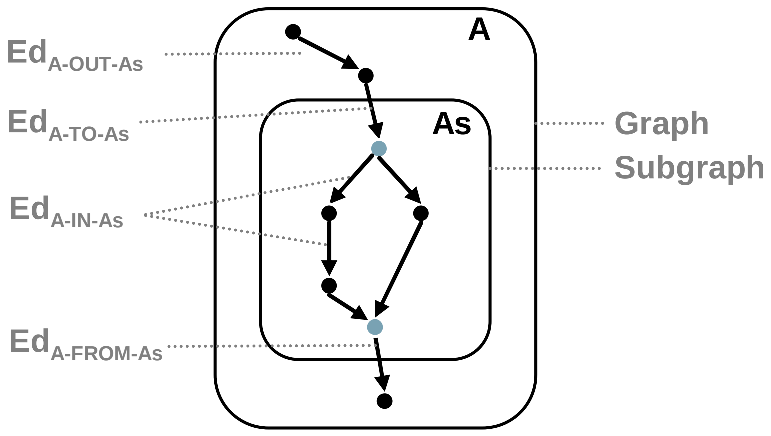


Figure 84: Abstract differentiation of Graph and Sub graph

In addition to nodes, edges between the nodes have to be defined. Basically, four different types of edges are defined, which are also labeled in Figure 84:

- edges lying outside of As (OUT), e.g. $Ed_{A\text{-out-As}}$
- edges leading into As (TO), e.g. $Ed_{A\text{-TO-As}}$
- edges lying inside of As (IN), e.g. $Ed_{A\text{-IN-As}}$, and
- edges moving out of As (FROM), $Ed_{A\text{-FROM-As}}$

A sub-graph restricts its edges to stay within sub-graph's nodes – that's why it is closed. The formalization of edges is defined as the following:

- $Ed_{A\text{-}OUT\text{-}As} =_{DEF} \{ (s,t) \in Ed_A \mid s \notin Ed_{As} \wedge t \notin Ed_{As} \}$.
- $Ed_{A\text{-}TO\text{-}As} =_{DEF} \{ (s,t) \in Ed_A \mid s \notin Ed_{As} \wedge t \in Ed_{As} \}$.
- $Ed_{A\text{-}IN\text{-}As} =_{DEF} \{ (s,t) \in Ed_A \mid s \in Ed_{As} \wedge t \in Ed_{As} \}$.
- $Ed_{A\text{-}FROM\text{-}As} =_{DEF} \{ (s,t) \in Ed_A \mid s \in Ed_{As} \wedge t \notin Ed_{As} \}$.

4.3.3 Hierarchical Graph Definition

As described in chapter 4.2.6, formalization of 'Hierarchical Integration' is special insofar that the hierarchical action points to the sub-process to be integrated.

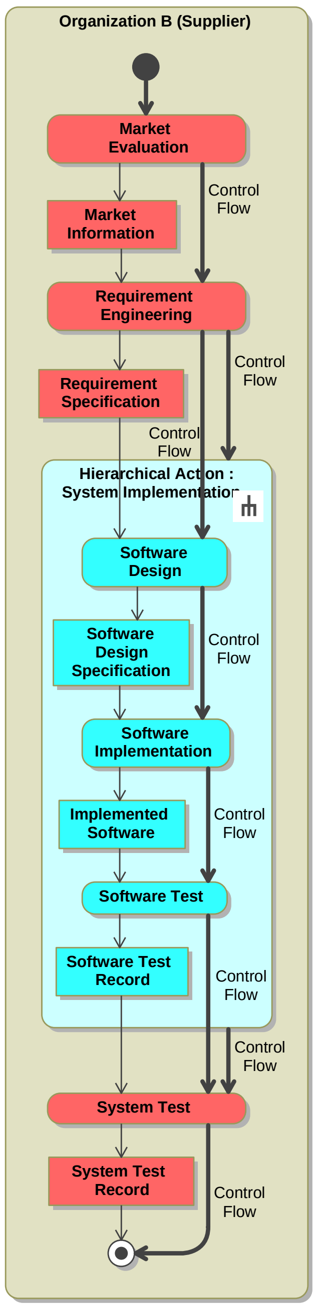


Figure 85: "Hybrid-View" of Hierarchical Processes

This sub-process is actually an entire process or activity diagram (*System Implementation*). Formalization of hierarchical processes takes place by using the “hybrid-view” as basis (Figure 85). The abstract view of the Hybrid View is depicted in Figure 86. The hierarchical action *System Implementation is* represented by the HA action node (dotted line). The content of *System Implementation* – which is basically the *Software Design, -Implementation,* and *-Test* – is defined with the sub-graph ad_{As}.

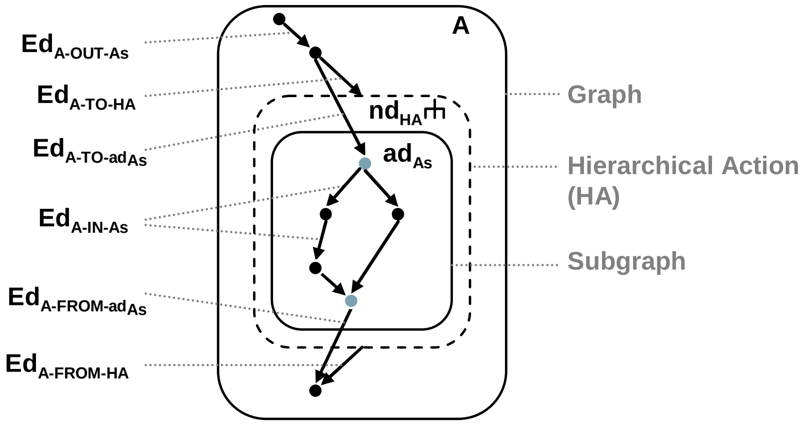


Figure 86: Abstract view of “Hybrid View”

The “hybrid view” incorporates two new edges, $Ed_{A\text{-TO-HA}}$ and $Ed_{A\text{-FROM-HA}}$. The formalization of edges is defined as the following:

- $Ed_{A\text{-TO-HA}} =_{DEF} \{ (s,t) \in Ed_A \mid s \in Ed_A \wedge t \in \{y_{HA}\} \}$
- $Ed_{A\text{-FROM-HA}} =_{DEF} \{ (s,t) \in Ed_A \mid s \in \{y_{HA}\} \wedge t \notin Ed_{adHA} \}$

Furthermore, all nodes from $ad_{As} \subseteq A$. The graph ad_{As} differs from sub-graph *As* that $As \subseteq ad_{As}$.

4.3.4 Mapping Methodology Definition

Furthermore, mappings are necessary to describe exactly, which node of the Master’s process to connect with a node of the Supplier’s process and vice versa. In order to include a sub-graph into a Master process, it is sufficient to just connect the sub-graph’s border nodes. To be more precisely, these are the nodes that are connected with a ‘TO’ or ‘FROM’-edge.

The set of all possible mappings for a given pair *As* and *Bs* of sub-graphs is defined as follows:

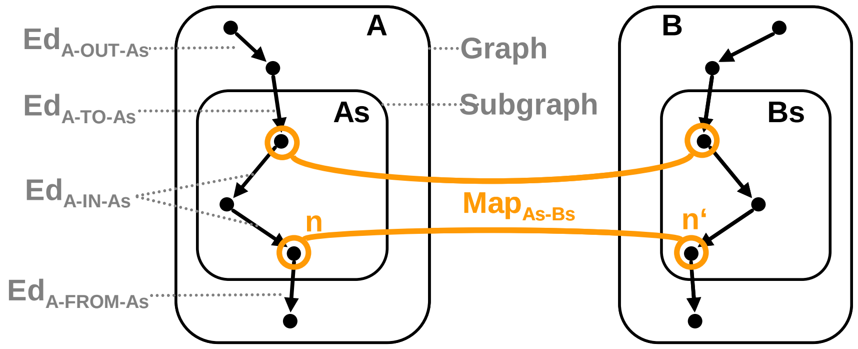
Definition 11.

$$MAP_{As-Bs} =_{DEF} \left\{ \begin{array}{l} MAP_{As-Bs} \subseteq Nd_{As} \times Nd_{Bs} \mid \\ \forall (s,n) \in Ed_{A-TO-As} \Rightarrow \exists (n,n') \in MAP_{As-Bs}, \\ \forall (s,n) \in Ed_{A-FROM-As} \Rightarrow \exists (n,n') \in MAP_{As-Bs} \end{array} \right\}$$

The formula says that for all edges Ed going from graph A into a sub-graph *As* (=$Ed_{A\text{-TO-As}}$) a corresponding node must be existent within graph B. This is the corresponding node, which is connected with an edge to be defined in an appropriate scenario. The mapping works also the other way back to graph A (=$Ed_{A\text{-FROM-As}}$). Figure 87 visualizes the mapping, showing orange arcs that symbolize those nodes to be mapped towards integration.

Remarks:

This mapping considers a pair of **nodes**; the orange arcs are **not edges** that connect any pair of nodes.



A
B
As
Bs
$Ed_{A\text{-}OUT\text{-}As}$
$Ed_{A\text{-}TO\text{-}As}$
$Ed_{A\text{-}IN\text{-}As}$
$Ed_{A\text{-}FROM\text{-}As}$
Graph
Subgraph
$Map_{As\text{-}Bs}$
n
n‘

Figure 87: Abstract illustration of mapping functionality

Having the edges and 'mapping' methodology as the basis, it is now possible to create an integration operation, which takes two graphs, two respective sub-graphs, and one mapping as arguments and produces an integration operation in order to generate a combined process:

Definition 12.

$\text{integrate\_graph}(A, As, B, Bs, MAP_{As-Bs}) =_{DEF} AB,$

where

- Nd_{AB} = $((Nd_A \cup Nd_B) \setminus Nd_{Del}) \cup Nd_{Add}$
- Ed_{AB} = $((Ed_A \cup Ed_B) \setminus Ed_{Del}) \cup Ed_{Add}$
- Nd_{Del} = *//to be defined by concrete integration operation*
- Nd_{Add} = *//to be defined by concrete integration operation*
- Ed_{Del} = *//to be defined by concrete integration operation*
- Ed_{Add} = *//to be defined by concrete integration operation*

Thereby,

- Nd_{AB} / Ed_{AB} is the set of nodes / edges from the original graphs A and B
- Nd_{Del} / Ed_{Del} is set of nodes /edges to be deleted
- Nd_{Add} / Ed_{Add} is set of nodes /edges to be added

This integration function is the basis for every subsequent integration function defined for each collaborative scenario. Considering the constraints of the function *integrate_graph*(), the set of nodes and edges need to be appropriately restricted to the amount that are crucial for the collaborative graph (or process respectively).

The abstract diagrams created to illustrate the collaborative process incorporate a defined color code:

Red:	delete edge from root process
Light Blue:	delete node from root process
Green:	added node and edge for collaborative process

The following legend explains how formalization is applied within the process integration scenarios.

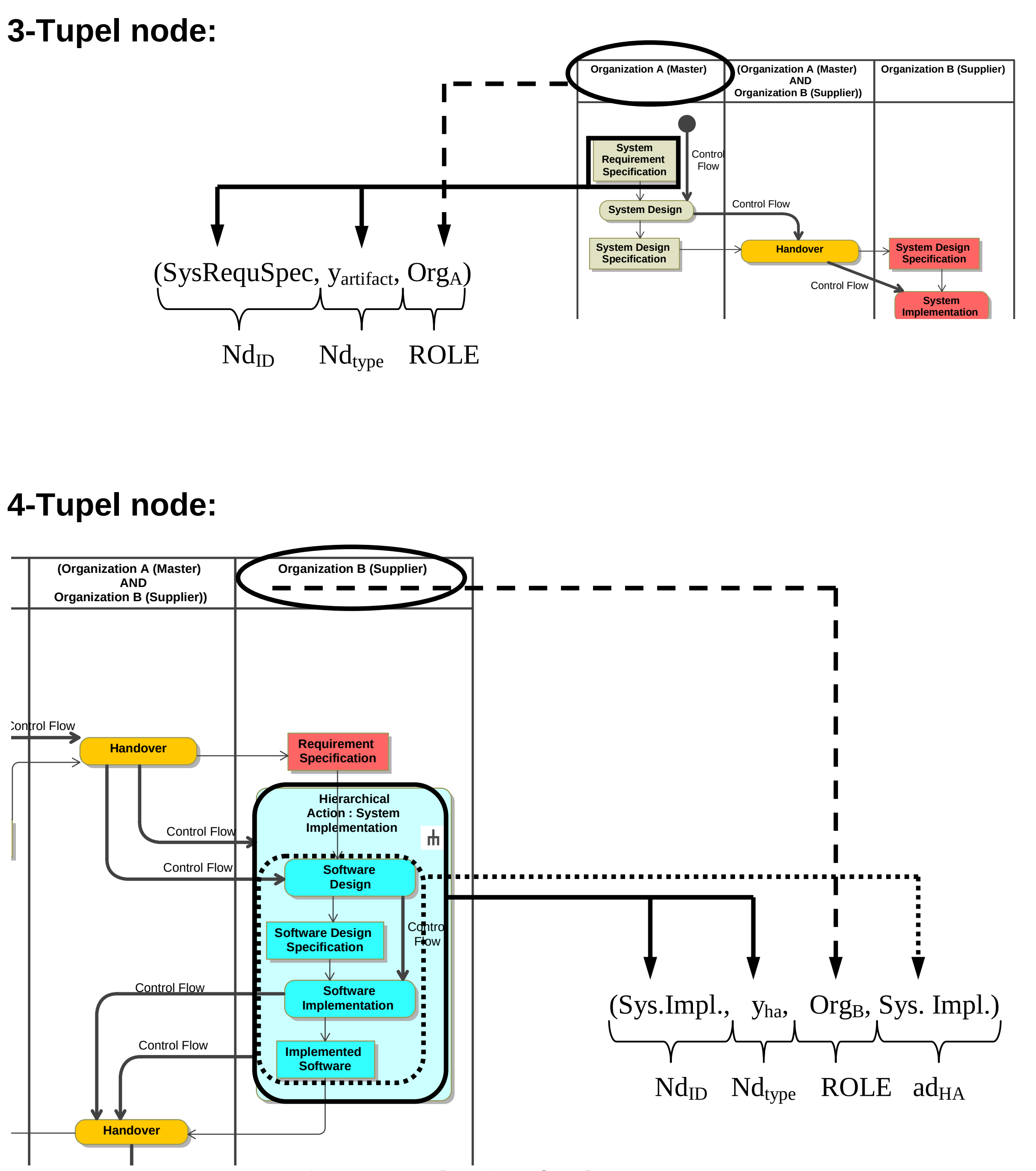

3-Tupel node:
Organization A (Master)
(Organization A (Master) AND Organization B (Supplier))
Organization B (Supplier)
System Requirement Specification
Control Flow
System Design
Control Flow
System Design Specification
Handover
System Design Specification
Control Flow
System Implementation
(SysRequSpec, y_artifact, Org_A)
Nd_ID
Nd_type
ROLE
4-Tupel node:
(Organization A (Master) AND Organization B (Supplier))
Organization B (Supplier)
Control Flow
Handover
Requirement Specification
Hierarchical Action : System Implementation
Control Flow
Control Flow
Software Design
Software Design Specification
Control Flow
Control Flow
Software Implementation
Control Flow
Implemented Software
Handover
(Sys.Impl., y_ha, Org_B, Sys. Impl.)
Nd_ID
Nd_type
ROLE
ad_HA

Figure 88: Explanation of Node Parameters

4.3.5 Semantically Equivalent Processes

In the following chapter, all scenarios are formalized using the concurrent examples initially introduced in chapter 4.1.2.

The collaborative scenario with semantically equivalent processes does not need any mediator, because actions and activities are not only named equally, they mean exactly the same. Consequently, desired actions can just be included into the Master's workflow.

The initial root process is depicted in Figure 53 that models control- and data flow. The formalization of the **control flow** for Organization A with respect to Definition 1 is defined as follows:

Definition 13.

A = ({

(Start, y_{start}, Org_A), (End, y_{end}, Org_A), (SysDes, y_{action}, Org_A), (SysImpl, y_{action}, Org_A), (SysTest, y_{action}, Org_A)

} , {

((Start,y_{start}, Org_A),(SysDes, y_{action}, Org_A)),
((SysDes,y_{action}, Org_A),(SysImpl,y_{action}, Org_A)),
((SysImpl,y_{action}, Org_A),(SysTest,y_{action}, Org_A)),
((SysTest,y_{action}, Org_A),(End,y_{end}, Org_A))
})

Please note that the first curly bracket in Definition 13 contains the set of all nodes needed for setting up the control flow graph A. A is meant to be the activity diagram of Organization A, which is the master. The second curly bracket contains all edges needed for the graph set-up. For better readability, each line always contains one edge including the starting and ending node of the edge.

Generally, the control flow in Definition 13 is not sufficient for a comprehensive understanding of a collaborative process, since practice shows that many development processes are defined and executed '*artifact-oriented*', i.e., process follows the required artifacts, which, in turn, requires data flow formalizations as an essential aspect. In the following data flow and root process for Organization A (Master) and B (Supplier) are defined:

Definition 14.

A = ({

(Start,y_{start}, Org_A), (End,y_{end}, Org_A),
(SysDesign, y_{action}, Org_A), (SysImpl, y_{action}, Org_A), (SysTest, y_{action}, Org_A),
(SysRequSpec, $y_{artifact}$, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A), (ImplSys,$y_{artifact}$, Org_A),
(SysTestRec, $y_{artifact}$, Org_A)

} , {

//[Controlflow]
((Start, y_{start}, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SysImpl, y_{action}, Org_A)),
((SysImpl, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (End, y_{end}, Org_A))

//[Dataflow]
((SysRequSpec, $y_{artifact}$, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A)),
((SysDesignSpec, $y_{artifact}$, Org_A), (SysImpl, y_{action}, Org_A)),
((SysImpl, y_{action}, Org_A), (ImplSys, $y_{artifact}$, Org_A)),
((ImplSys, $y_{artifact}$, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A)),
((SysTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))
}).

Definition 15.

B = ({
(Start, y_{start}, Org_B), (End, y_{end}, Org_B),
(SysDesign, y_{action}, Org_B), (SysImpl, y_{action}, Org_B), (SysTest, y_{action}, Org_B),
(SysRequSpec, $y_{artifact}$, Org_B), (SysDesignSpec, $y_{artifact}$, Org_B), (ImplSys, $y_{artifact}$, Org_B),
(SysTestRec, $y_{artifact}$, Org_B)

} , {

//[Controlflow]
((Start, y_{start}, Org_B), (SysDesign, y_{action}, Org_B)),
((SysDesign, y_{action}, Org_B), (SysImpl, y_{action}, Org_B)),
((SysImpl, y_{action}, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (End, y_{end}, Org_B))

//[Dataflow]
((SysRequSpec, $y_{artifact}$, Org_B), (SysDesign, y_{action}, Org_B)),
((SysDesign, y_{action}, Org_B), (SysDesignSpec, $y_{artifact}$, Org_B)),
((SysDesignSpec, $y_{artifact}$, Org_B), (SysImpl, y_{action}, Org_B)),
((SysImpl, y_{action}, Org_B), (ImplSys, $y_{artifact}$, Org_B)),
((ImplSys, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B)),
((SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))
}).

It is now necessary to define an abstract operation that integrates one action or activity into another. Based on the function *integrate_graph*() defined in Definition 12, the integration of semantically equivalent processes follows the function in Definition 16:

Definition 16.
integrate_SemEq() extends *integrate_graph()*
where

- $Nd_{Del} = Nd_{As} \cup (Nd_B \setminus Nd_{Bs})$
- $Nd_{Add} = \varnothing$
- $Ed_{Del} = Ed_{A\text{-}TO\text{-}As} \cup Ed_{A\text{-}FROM\text{-}As} \cup Ed_{A\text{-}IN\text{-}As} \cup Ed_{B\text{-}TO\text{-}Bs} \cup Ed_{B\text{-}FROM\text{-}Bs} \cup Ed_{B\text{-}OUT\text{-}Bs}$
- $Ed_{Add} = Ed_{AddA2B} \cup Ed_{AddB2A}$
- $Ed_{AddA2B} = \{ (s,t') \in ED \mid \exists (s,t) \in Ed_{A\text{-}TO\text{-}As}, (t,t') \in Map_{As\text{-}Bs} \}$
- $Ed_{AddB2A} = \{ (s',t) \in ED \mid \exists (s,t) \in Ed_{A\text{-}FROM\text{-}As}, (s,s') \in Map_{As\text{-}Bs} \}$

Figure 89 shows the integration of semantically equivalent processes. Remark, that there are only two edges to be added, the one that points to the sub-graph to be included (Ed_{AddA2B}) and the one that leads back to the Master (Ed_{AddB2A}). Furthermore, a mapping needs to be done including the nodes (t, t'). In this case, it works without any complications, since the action delivers exactly those artifacts needed by t or t' (semantic equivalence!).

As the color code shows, the arrows in red and nodes in light blue are erased; whereas, the green edges are added, if a semantically equivalent action from organization B is integrated into Organization's A process.

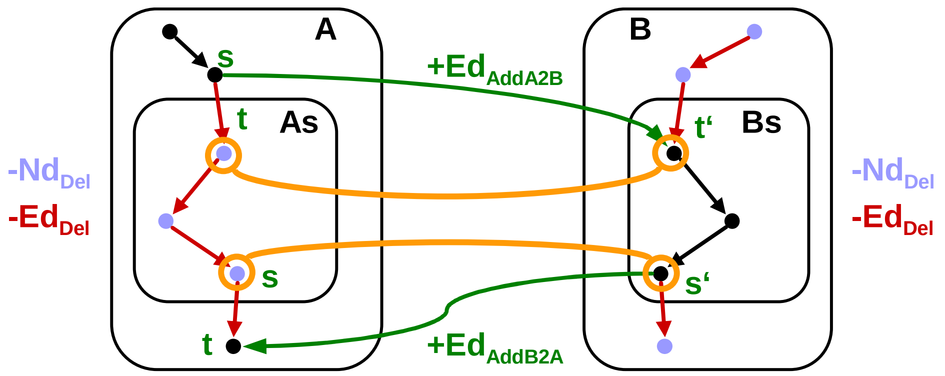


Figure 89: Abstract modeling for Integration of semantically equivalent processes

4.3.6 Horizontal Integration

Due to the fact that processes in the 'Horizontal Integration' scenario are not semantically equivalent, the Mediator concept needs to be used. For this purpose, another function is defined, which considers the inclusion of appropriate *Handover* action to be included (compare Figure 56).

The abstract syntax for 'Horizontal Integration' function is defined as follows:

Definition 17.
integrate_horiz() extends *integrate_graph()*,
where

- $Nd_{Del} = Nd_{As} \cup (Nd_B \setminus Nd_{Bs})$
- Nd_{Add} = HandovBeg ∪ HandovEnd
- HandovBeg = $\{hob_{(s,t)} \in ND_{id} \times \{y_{action}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}TO\text{-}As} \}$
- HandovEnd = $\{hoe_{(s,t)} \in ND_{id} \times \{y_{action}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}FROM\text{-}As} \}$
- $Ed_{Del} = Ed_{A\text{-}TO\text{-}As} \cup Ed_{A\text{-}FROM\text{-}As} \cup Ed_{A\text{-}IN\text{-}As}$
  $\cup \ Ed_{B\text{-}TO\text{-}Bs} \cup Ed_{B\text{-}FROM\text{-}Bs} \cup Ed_{B\text{-}OUT\text{-}Bs}$
- $Ed_{Add} = Ed_{AddA2Sync} \cup Ed_{AddSync2B} \cup Ed_{AddB2Sync} \cup Ed_{AddSync2A}$
- $Ed_{Add\_A2hob} = \{ (s, hob_{(s,t)}) \in ED \mid hob_{(s,t)} \in$ HandovBeg $\}$
- $Ed_{Add\_hob2B} = \{ (hob_{(s,t)},t') \in ED \mid hob_{(s,t)} \in$ HandovBeg, $(t,t') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_B2hoe} = \{ (s',hoe_{(s,t)}) \in ED \mid hoe_{(s,t)} \in$ HandovEnd, $(s,s') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_hoe2A} = \{ (hoe_{(s,t)}, t) \in ED \mid hoe_{(s,t)} \in$ HandovEnd $\}$

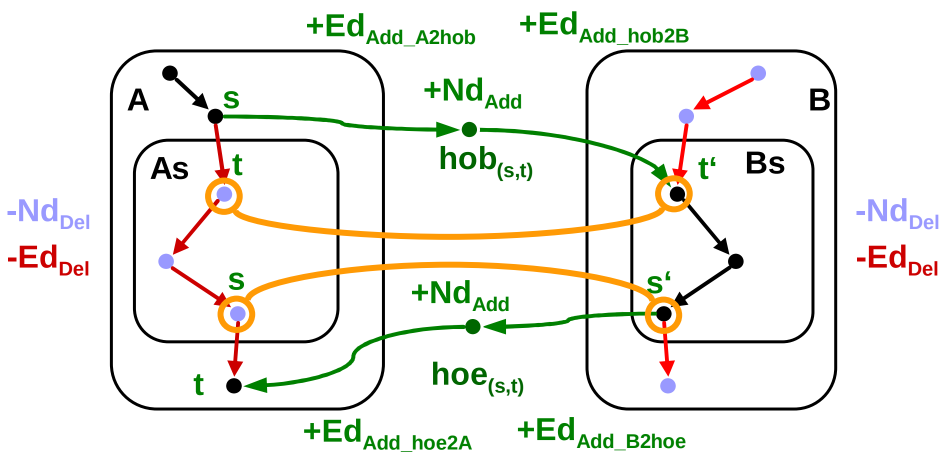


Figure 90: Abstract modeling of 'Horizontal Integration'

HandovBeg and HandovEnd are a set of nodes within a Mediator necessary to connect semantically non- equivalent processes. Thereby, responsibility is allocated by having an "AND" connection defined.
Figure 90 shows the abstract modeling of 'Horizontal Integration' including the concrete nodes of "HandovBeg" (= hob) and "HandovEnd" (= hoe).

Applying the abstract syntax to the concurrent example results in the following Definition 18 (control and data flow) according to Figure 56.

Definition 18.
integrate_horiz() extends *integrate_graph()*,
where

- Nd_{Del} = (SysImpl, y_{action}, Org_A), (ImplSys, $y_{artifact}$, Org_A), (Start, y_{start}, Org_B), (End, y_{end}, Org_B), (SysDes, y_{action}, Org_B), (SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B), (SysRequSpec, $y_{artifact}$, Org_B)

- Nd_{Add} = (HandovBeg, y_{action}, (Org_A AND Org_B)), (HandovEnd, y_{action}, (Org_A AND Org_B))

- Ed_{Del} =
  //[Controlflow]
  ((SysDes, y_{action}, Org_A), (SysImpl, y_{action}, Org_A)),
  ((SysImpl, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
  ((Start, y_{start}, Org_B), (SysDes, y_{action}, Org_B)),
  ((SysDes, y_{action}, Org_B), (SysImpl, y_{action}, Org_B)),
  ((SysImpl, y_{action}, Org_B), (SysTest, y_{action}, Org_B)),
  ((SysTest, y_{action}, Org_B), (End, y_{end}, Org_B))

  //[Dataflow]
  ((SysDesignSpec, $y_{artifact}$, Org_A), (SysImpl, y_{action}, Org_A)),
  ((SysImpl, y_{action}, Org_A), (ImplSys, $y_{artifact}$, Org_A)),
  ((SysRequSpec, $y_{artifact}$, Org_B), (SysDes, y_{action}, Org_B)),
  ((SysDes, y_{action}, Org_B), (SysDesignSpec, $y_{artifact}$, Org_B)),
  ((ImplSys, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B)),
  ((SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B)),
  ((SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))

- Ed_{Add} =
  //[Controlflow]
  ((SysDes, y_{action}, Org_A), (HandovBeg, y_{action}, (Org_A AND Org_B))),
  ((HandovBeg, y_{action}, (Org_A AND Org_B)), (SysImpl, y_{action}, Org_B)),
  ((SysImpl, y_{action}, Org_B), (HandovEnd, y_{action}, (Org_A AND Org_B))),
  ((HandovEnd, y_{action}, (Org_A AND Org_B)), (SysTest, y_{action}, Org_A))

  //[Dataflow]
  ((SysDesignSpec, $y_{artifact}$, Org_A), (HandovBeg, y_{action}, (Org_A AND Org_B))),
  ((HandovBeg, y_{action}, (Org_A AND Org_B)), (SysDesignSpec, $y_{artifact}$, Org_B)),
  ((ImplSys, $y_{artifact}$, Org_B), (HandovEnd, y_{action}, (Org_A AND Org_B))),
  ((HandovEnd, y_{action}, (Org_A AND Org_B)), (ImplSys, $y_{artifact}$, Org_A))

4.3.7 Additive Vertical Integration

Figure 91 shows the abstract modeling of ‘Additive Vertical Integration’, which makes additional Fork and Join nodes necessary due to the property of parallelism.

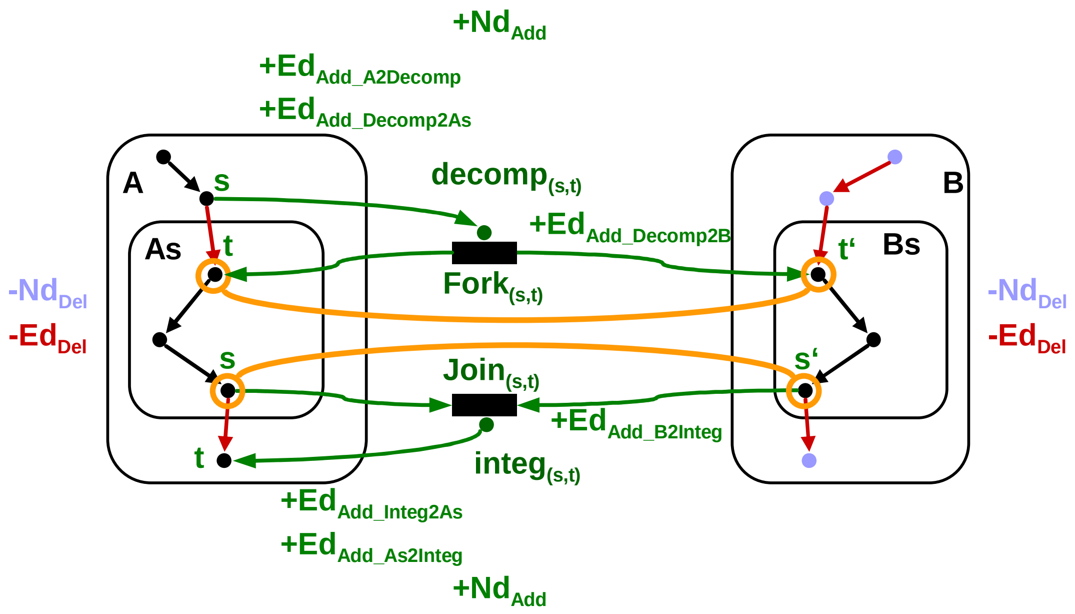


Figure 91: Abstract modeling of ‘Additive Vertical Integration’

The abstract syntax for ‘Additive Vertical Integration’ is defined using the subsequent function.

Definition 19.
integrate_additive_vertical() extends *integrate_graph()*,
where

- $Nd_{Del} = (Nd_B \setminus Nd_{Bs})$
- $Nd_{Add} = Decomp \cup Integ \cup Join \cup Fork$

- $Decomp = \{decomp_{(s,t)} \in ND_{id} \times \{y_{fork}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}TO\text{-}As} \}$
- $Integ = \{integ_{(s,t)} \in ND_{id} \times \{y_{join}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}FROM\text{-}As} \}$
- $Fork = \{fork_{(s,t)} \in ND_{id} \times \{y_{fork}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}TO\text{-}As} \}$
- $Join = \{join_{(s,t)} \in ND_{id} \times \{y_{join}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}FROM\text{-}As} \}$

- $Ed_{Del} = Ed_{A\text{-}TO\text{-}As} \cup Ed_{A\text{-}FROM\text{-}As} \cup Ed_{B\text{-}TO\text{-}Bs} \cup Ed_{B\text{-}FROM\text{-}Bs} \cup Ed_{B\text{-}OUT\text{-}Bs}$
- $Ed_{Add} = Ed_{AddA2Decomp} \cup Ed_{AddDecomp2B} \cup Ed_{AddDecomp2As} \cup Ed_{AddB2Integ} \cup Ed_{AddInteg2A}$
  $\cup\, Ed_{AddAs2Integ}$

- $Ed_{Add\_Decomp2Fork} = \{ (decomp_{(s,t)}, fork) \in ED \mid fork_{(s,t)} \in Fork, decomp_{(s,t)} \in Decomp\}$
- $Ed_{Add\_Join2Integ} = \{ (join_{(s,t)}, integ) \in ED \mid join_{(s,t)} \in Join, integ_{(s,t)} \in Integ\}$

- $Ed_{Add\_A2Decomp} = \{ (s, decomp_{(s,t)}) \in ED \mid decomp_{(s,t)} \in Decomp\}$
- $Ed_{Add\_Decomp2B} = \{ (decomp_{(s,t)},t') \in ED \mid decomp_{(s,t)} \in Decomp, (t,t') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_Decomp2As} = \{ (decomp_{(s,t)},t) \in ED \mid decomp_{(s,t)} \in Decomp\}$
- $Ed_{Add\_B2Integ} = \{ (s',integ_{(s,t)}) \in ED \mid integ_{(s,t)} \in Integ, (s,s') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_As2Integ} = \{ (s, integ_{(s,t)}) \in ED \mid integ_{(s,t)} \in Integ\}$
- $Ed_{Add\_Integ2A} = \{ (integ_{(s,t)}, t) \in ED \mid integ_{(s,t)} \in Integ\}$

The initial root processes have been modified for showing the functionality of this scenario. Therefore, the definition of node and edges of organization's A and B original processes in Figure 57 are formalized as the following:

Definition 20.

A = ({

(Start, y_{start}, Org_A), (End, y_{end}, Org_A),
(SysDesign, y_{action}, Org_A), (SWRequEng, y_{action}, Org_A), (SWDesign, y_{action}, Org_A),
(SWImpl, y_{action}, Org_A), (SWTest, y_{action}, Org_A), (SysTestDesign, y_{action}, Org_A),
(SysTest, y_{action}, Org_A), (SysRequSpec, $y_{artifact}$, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A),
(SWRequSpec, $y_{artifact}$, Org_A), (SWDesignSpec, $y_{artifact}$, Org_A), (ImplSW, $y_{artifact}$, Org_A),
(SWTestRec, $y_{artifact}$, Org_A), (SysTestSpec, $y_{artifact}$, Org_A), (SysTestRec, $y_{artifact}$, Org_A),

} , {

//[Controlflow]
((Start, y_{start}, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SWRequEng, y_{action}, Org_A)),
((SWRequEng, y_{action}, Org_A), (SWDesign, y_{action}, Org_A)),
((SWDesign, y_{action}, Org_A), (SWImpl, y_{action}, Org_A)),
((SWImpl, y_{action}, Org_A), (SWTest, y_{action}, Org_A)),
((SWTest, y_{action}, Org_A), (SysTestDesign, y_{action}, Org_A)),
((SWTestDesign, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (End, y_{end}, Org_A))

//[Dataflow]
((SysRequSpec, $y_{artifact}$, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A)),
((SysDesignSpec, $y_{artifact}$, Org_A), (SWRequEng, y_{action}, Org_A)),
((SWRequEng, y_{action}, Org_A), (SWRequSpec, $y_{artifact}$, Org_A)),
((SWRequSpec, $y_{artifact}$, Org_A), (SWDesign, y_{action}, Org_A)),
((SWDesign, y_{action}, Org_A), (SWDesignSpec, $y_{artifact}$, Org_A)),
((SWDesignSpec, $y_{artifact}$, Org_A), (SWImpl, y_{action}, Org_A)),
((SWImpl, y_{action}, Org_A), (ImplSW, $y_{artifact}$, Org_A)),
((ImplSW, $y_{artifact}$, Org_A), (SWTest, y_{action}, Org_A)),
((SWTest, y_{action}, Org_A), (SWTestRec, $y_{artifact}$, Org_A)),
((SWTestRec, $y_{artifact}$, Org_A), (SysTestDesign, y_{action}, Org_A)),
((SysTestDesign, y_{action}, Org_A), (SysTestSpec, $y_{artifact}$, Org_A)),
((SysTestSpec, $y_{artifact}$, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A)),
((SysTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))

}).

Definition 21.

B = ({

(Start, y_{start}, Org_B), (End, y_{end}, Org_B),
(HWDesign, y_{action}, Org_B), (HWImpl, y_{action}, Org_B), (HWTest, y_{action}, Org_B),
(HWRequSpec, $y_{artifact}$, Org_B), (HWDesignSpec, $y_{artifact}$, Org_B),
(HWImplSys, $y_{artifact}$, Org_B), (HWTestRec, $y_{artifact}$, Org_B)

} , {

//[Controlflow]
((Start, y_{start}, Org_B), (HWDesign, y_{action}, Org_B)),
((HWDesign, y_{action}, Org_B), (HWImpl, y_{action}, Org_B)),
((HWImpl, y_{action}, Org_B), (HWTest, y_{action}, Org_B)),
((HWTest, y_{action}, Org_B), (End, y_{end}, Org_B))

//[Dataflow]
((HWRequSpec, $y_{artifact}$, Org_B), (HWDesign, y_{action}, Org_B)),
((HWDesign, y_{action}, Org_B), (HWDesignSpec, $y_{artifact}$, Org_B)),
((HWDesignSpec, $y_{artifact}$, Org_B), (HWImpl, y_{action}, Org_B)),
((HWImpl, y_{action}, Org_B), (HWImplSys, $y_{artifact}$, Org_B)),
((HWImplSys, $y_{artifact}$, Org_B), (HWTest, y_{action}, Org_B)),
((HWTest, y_{action}, Org_B), (HWTestRec, $y_{artifact}$, Org_B)),
((HWTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))
}).

Based on the concrete formalized definition of the root processes, the function in Definition 19 is applied in Definition 22 (according to Figure 59):

Definition 22.
integrate_additive_vertical() extends *integrate_graph()*,
where

- Nd_{Del} = (Start, y_{start}, Org_B), (End, y_{end}, Org_B),
  (SWRequEng, y_{action}, Org_A), (SysTestDes, y_{action}, Org_A)

- Nd_{Add} = (Decomp, y_{action}, (Org_A AND Org_B)), (Integ, y_{action}, (Org_A AND Org_B))
  (Fork, y_{fork}, (Org_A AND Org_B)), (Join, y_{join}, (Org_A AND Org_B))

- Ed_{Del} =
  //[Controlflow]
  ((SysDes, y_{action}, Org_A), (SWRequEng, y_{action}, Org_A)),
  ((SWRequEng, y_{action}, Org_A), (SWDesign, y_{action}, Org_A))
  ((SWTest, y_{action}, Org_A), (SysTestDesign, y_{action}, Org_A)),
  ((SysTestDesign, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
  ((Start, y_{start}, Org_B), (HWDesign, y_{action}, Org_B)),
  ((HWTest, y_{action}, Org_B), (End, y_{end}, Org_B))

  //[Dataflow]
  ((SysDesignSpec, $y_{artifact}$, Org_A), (SWRequEng, y_{action}, Org_A)),
  ((SWRequEng, y_{action}, Org_A), (SWRequSpec, $y_{artifact}$, Org_A)),
  ((SWTestRec, $y_{artifact}$, Org_A), (SysTestDesign, y_{action}, Org_A)),
  ((SysTestDesign, y_{action}, Org_A), (SysTestSpec, $y_{artifact}$, Org_A)),
  ((HWTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))

- Ed_{Add} =
  //[Controlflow]
  ((SysDesign, y_{action}, Org_A), (Decomp, y_{action}, (Org_A AND Org_B))),
  ((Decomp, y_{action}, (Org_A AND Org_B)), (Fork, y_{fork}, (Org_A AND Org_B))),

((Fork, y_{fork}, (Org_A AND Org_B), (SWDesign, y_{action}, Org_A)),
((Fork, y_{fork}, (Org_A AND Org_B), (HWDesign, y_{action}, Org_B)),
((HWTest, y_{action}, Org_B), (Join, y_{join}, (Org_A AND Org_B))),
((SWTest, y_{action}, Org_A), (Join, y_{join}, (Org_A AND Org_B))),
((Join, y_{join}, (Org_A AND Org_B)), (Integ, y_{action}, (Org_A AND Org_B))),
((Integ, y_{action}, (Org_A AND Org_B)), (SysTest, y_{action}, Org_A))

//[Dataflow]
((SysDesignSpec, $y_{artifact}$, Org_A), (Decomp, y_{action}, (Org_A AND Org_B))),
((Decomp, y_{action}, (Org_A AND Org_B)), (SWRequSpec, $y_{artifact}$, Org_A)),
((Decomp, y_{action}, (Org_A AND Org_B)), (HWRequSpec, $y_{artifact}$, Org_B)),
((SWTestRec, $y_{artifact}$, Org_A), (Integ, y_{action}, (Org_A AND Org_B))),
((HWTestRec, $y_{artifact}$, Org_B), (Integ, y_{action}, (Org_A AND Org_B))),
((Integ, y_{action}, (Org_A AND Org_B)), (SysTestSpec, $y_{artifact}$, Org_A))

4.3.8 Alternative Vertical Integration

The abstract syntax for 'Additive Vertical Integration' is defined using the following function.

Definition 23.
integrate_alternative_vertical() extends *integrate_graph()*,
where

- $Nd_{Del} = (Nd_B \setminus Nd_{Bs})$
- $Nd_{Add} = Ratio \cup Accept \cup Branch \cup Merge$

- $RatioAlys = \{ratioalys_{(s,t)} \in ND_{id} \times \{y_{branch}\} \times \text{<ROLE>} \mid (s,t) \in Ed_{A\text{-}TO\text{-}As} \}$
- $Accept = \{accept_{(s,t)} \in ND_{id} \times \{y_{merge}\} \times \text{<ROLE>} \mid (s,t) \in Ed_{A\text{-}FROM\text{-}As} \}$
- $Branch = \{branch_{(s,t)} \in ND_{id} \times \{y_{branch}\} \times \text{<ROLE>} \mid (s,t) \in Ed_{A\text{-}TO\text{-}As} \}$
- $Merge = \{merge_{(s,t)} \in ND_{id} \times \{y_{merge}\} \times \text{<ROLE>} \mid (s,t) \in Ed_{A\text{-}FROM\text{-}As} \}$

- $Ed_{Add\_RatioAlys2Branch} = \{ (ratioalys_{(s,t)}, branch) \in ED \mid ratioalys_{(s,t)} \in RatioAlys, branch_{(s,t)} \in Branch\}$
- $Ed_{Add\_Merge2Accep} = \{ (merge_{(s,t)}, accpt) \in ED \mid merge_{(s,t)} \in Merge, accept_{(s,t)} \in Accept\}$

- $Ed_{Del} = Ed_{A\text{-}TO\text{-}As} \cup Ed_{A\text{-}FROM\text{-}As} \cup Ed_{B\text{-}TO\text{-}Bs} \cup Ed_{B\text{-}FROM\text{-}Bs} \cup Ed_{B\text{-}OUT\text{-}Bs}$
- $Ed_{Add} = Ed_{AddA2Decomp} \cup Ed_{AddDecomp2B} \cup Ed_{AddDecomp2As} \cup Ed_{AddB2Integ} \cup Ed_{AddInteg2A} \cup Ed_{AddAs2Integ}$

- $Ed_{Add\_A2RatioAlys} = \{ (s, ratioalys_{(s,t)}) \in ED \mid ratioalys_{(s,t)} \in RatioAlys\}$
- $Ed_{Add\_RatioAlys\ 2B} = \{ (ratioalys_{(s,t)}, t') \in ED \mid ratioalys_{(s,t)} \in RatioAlys, (t,t') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_RatioAlys\ 2As} = \{ (ratioalys_{(s,t)}, t) \in ED \mid ratioalys_{(s,t)} \in RatioAlys\}$
- $Ed_{Add\_B2Accept} = \{ (s', accept_{(s,t)}) \in ED \mid accept_{(s,t)} \in Accept, (s,s') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_As2Accept} = \{ (s, accept_{(s,t)}) \in ED \mid accept_{(s,t)} \in Accept\}$
- $Ed_{Add\_Accept2A} = \{ (accept_{(s,t)}, t) \in ED \mid accept_{(s,t)} \in Accept\}$

As mentioned before, this scenario is very similar to 'Additive Vertical Integration'. The difference is the *Fork* and *Join* functionality, which is replaced by *Branch* and *Merge*.

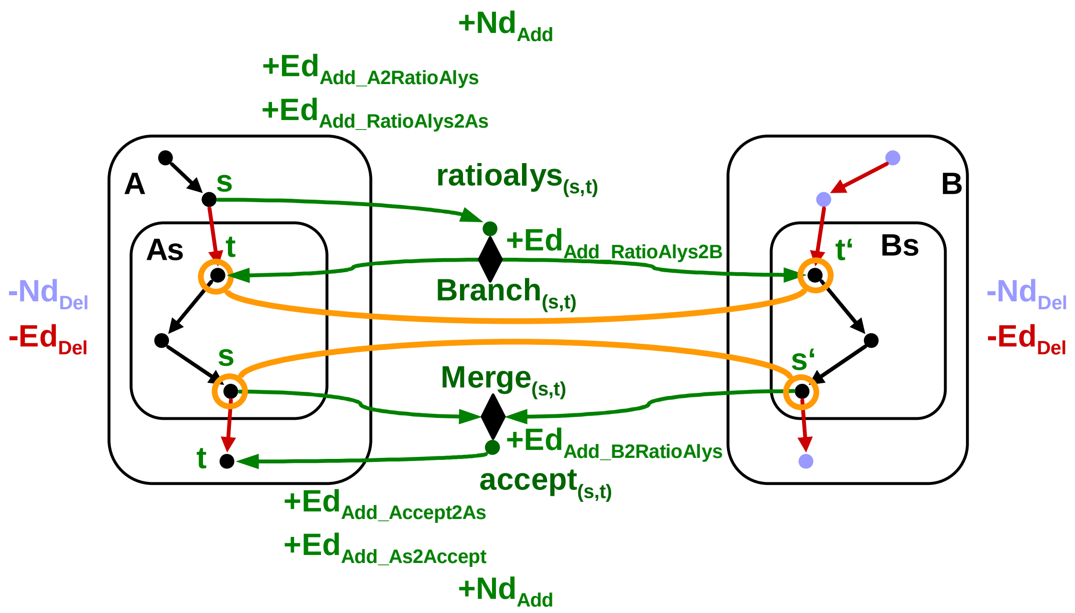


Figure 92: Abstract modeling of 'Alternative Vertical Integration'

In order to apply this scenario, the root processes are again concretely defined following the modeling in Figure 61.

Definition 24.
A = ({
(Start, y_{start}, Org_A), (End, y_{end}, Org_A),
(SysDesign, y_{action}, Org_A), (SWRequEng, y_{action}, Org_A), (SWDesign, y_{action}, Org_A),
(SWImpl, y_{action}, Org_A), (SWTest, y_{action}, Org_A), (SysTestDesign, y_{action}, Org_A),
(SysTest, y_{action}, Org_A), (SysRequSpec, $y_{artifact}$, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A),
(SWRequSpec, $y_{artifact}$, Org_A), (SWDesignSpec, $y_{artifact}$, Org_A), (ImplSW, $y_{artifact}$, Org_A),
(SWTestRec, $y_{artifact}$, Org_A), (SysTestSpec, $y_{artifact}$, Org_A), (SysTestRec, $y_{artifact}$, Org_A),
} , {

//[Controlflow]
((Start, y_{start}, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SWRequEng, y_{action}, Org_A)),
((SWRequEng, y_{action}, Org_A), (SWDesign, y_{action}, Org_A)),
((SWDesign, y_{action}, Org_A), (SWImpl, y_{action}, Org_A)),
((SWImpl, y_{action}, Org_A), (SWTest, y_{action}, Org_A)),
((SWTest, y_{action}, Org_A), (SysTestDesign, y_{action}, Org_A)),
((SWTestDesign, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}), (End, y_{end}))

//[Dataflow]
((SysRequSpec, $y_{artifact}$, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A)),
((SysDesignSpec, $y_{artifact}$, Org_A), (SWRequEng, y_{action}, Org_A)),
((SWRequEng, y_{action}, Org_A), (SWRequSpec, $y_{artifact}$, Org_A)),
((SWRequSpec, $y_{artifact}$, Org_A), (SWDesign, y_{action}, Org_A)),
((SWDesign, y_{action}, Org_A), (SWDesignSpec, $y_{artifact}$, Org_A)),
((SWDesignSpec, $y_{artifact}$, Org_A), (SWImpl, y_{action}, Org_A)),
((SWImpl, y_{action}, Org_A), (ImplSW, $y_{artifact}$, Org_A)),
((ImplSW, $y_{artifact}$, Org_A), (SWTest, y_{action}, Org_A)),

((SWTest, y_{action}, Org_A), (SWTestRec, $y_{artifact}$, Org_A)),
((SWTestRec, $y_{artifact}$, Org_A), (SysTestDesign, y_{action}, Org_A)),
((SysTestDesign, y_{action}, Org_A), (SysTestSpec, $y_{artifact}$, Org_A)),
((SysTestSpec, $y_{artifact}$, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A)),
((SysTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))
}).

Definition 25.
B = ({
(Start, y_{start}, Org_B), (End, y_{end}, Org_B),
(SafCrDesign, y_{action}, Org_B), (SafCrImpl, y_{action}, Org_B), (SafCrTest, y_{action}, Org_B),
(SafCrRequSpec, $y_{artifact}$, Org_B), (SafCrDesignSpec, $y_{artifact}$, Org_B),
(SafCrImplSys, $y_{artifact}$, Org_B), (SafCrTestRec, $y_{artifact}$, Org_B)

} , {

//[Controlflow]
((Start, y_{start}, Org_B), (SafCrDesign, y_{action}, Org_B)),
((SafCrDesign, y_{action}, Org_B), (SafCrImpl, y_{action}, Org_B)),
((SafCrImpl, y_{action}, Org_B), (SafCrTest, y_{action}, Org_B)),
((SafCrTest, y_{action}, Org_B), (End, y_{end}, Org_B)),

//[Dataflow]
((SafCrRequSpec, $y_{artifact}$, Org_B), (SafCrDesign, y_{action}, Org_B)),
((SafCrDesign, y_{action}, Org_B), (SafCrDesignSpec, $y_{artifact}$, Org_B)),
((SafCrDesignSpec, $y_{artifact}$, Org_B), (SafCrImpl, y_{action}, Org_B)),
((SafCrImpl, y_{action}, Org_B), (SafCrImplSys, $y_{artifact}$, Org_B)),
((SafCrImplSys, $y_{artifact}$, Org_B), (SafCrTest, y_{action}, Org_B)),
((SafCrTest, y_{action}, Org_B), (SafCrTestRec, $y_{artifact}$, Org_B)),
((SafCrTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))
}).

Using Definition 23 the concrete integration operation for ‘Alternative Vertical Integration’ is defined (according to Figure 62) Definition 26:

Definition 26.
integrate_alternative_vertical() extends *integrate_graph()*,
where

- Nd_{Del} = ($Start_B$, y_{start}, Org_B), (End_B, y_{end}, Org_B),
  (SWRequEng, y_{action}, Org_A), (SysTestDes, y_{action}, Org_A)

- Nd_{Add} = (RatioAlys, y_{action}, (Org_A AND Org_B)), (Accept, y_{action}, (Org_A AND Org_B)),
  (Branch, y_{branch}, (Org_A AND Org_B)), (Merge, y_{merge}, (Org_A AND Org_B))

- Ed_{Del} =
  //[Controlflow]
  ((SysDes, y_{action}, Org_A), (SWRequEng, y_{action}, Org_A)),
  ((SWRequEng, y_{action}, Org_A), (SWDesign, y_{action}, Org_A))
  ((SWTest, y_{action}, Org_A), (SysTestDesign, y_{action}, Org_A)),
  ((SysTestDesign, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
  ((Start, y_{start}, Org_B), (SafCrDesign, y_{action}, Org_B)),

((SafCrTest, y_{action}, Org_B), (End, y_{end}, Org_B))

//[Dataflow]
((SysDesignSpec, $y_{artifact}$, Org_A), (SWRequEng, y_{action}, Org_A)),
((SWRequEng, y_{action}, Org_A), (SWRequSpec, $y_{artifact}$, Org_A)),
((SWTestRec, $y_{artifact}$, Org_A), (SysTestDesign, y_{action}, Org_A)),
((SysTestDesign, y_{action}, Org_A), (SysTestSpec, $y_{artifact}$, Org_A)),
((SafCrTestRec, $y_{artifact}$, Org_B), (End_B, y_{end}, Org_B))

- Ed_{Add} =
  //[Controlflow]
  ((SysDes, y_{action}, Org_A), (RatioAlys, y_{action}, (Org_A AND Org_B))),
  ((RatioAlys, y_{action}, (Org_A AND Org_B)), (Branch, y_{branch}, (Org_A AND Org_B))),
  ((Branch, y_{branch}, (Org_A AND Org_B)), (SWDesign, y_{action}, Org_A)),
  ((Branch, y_{branch}, (Org_A AND Org_B)), (SafCrDesign, y_{action}, Org_B)),
  ((SafCrTest, y_{action}, Org_B), (Merge, y_{merge}, (Org_A AND Org_B)),
  ((SWTest, y_{action}, Org_A), (Merge, y_{merge}, (Org_A AND Org_B)),
  ((Merge, y_{merge}, (Org_A AND Org_B), (Accept, y_{action}, (Org_A AND Org_B))),
  ((Accept, y_{action}, (Org_A AND Org_B)), (SysTest, y_{action}, Org_A))

  //[Dataflow]
  ((SysDesignSpec, $y_{artifact}$, Org_A), (RatioAlys, y_{action}, (Org_A AND Org_B))),
  ((RatioAlys, y_{action}, (Org_A AND Org_B)), (SWRequSpec, $y_{artifact}$, Org_A)),
  ((RatioAlys, y_{action}, (Org_A AND Org_B)), (SafCrRequSpec, $y_{artifact}$, Org_B)),
  ((SWTestRec, $y_{artifact}$, Org_A), (Accept, y_{action}, (Org_A AND Org_B))),
  ((SafCrTestRec, $y_{artifact}$, Org_B), (Accept, y_{action}, (Org_A AND Org_B))),
  ((Accept, y_{action}, (Org_A AND Org_B)), (SysTestSpec, $y_{artifact}$, Org_A))

4.3.9 Merging Integration

The abstract syntax for ‘Merging Integration’ is defined using the following function.

Definition 27.
integrate_merging() extends *integrate_graph(),*
where

- $Nd_{Del} = Nd_{As} \cup Nd_B$
- $Nd_{Add} = SyncBeg \cup Handov \cup Nd_{Cs}$
- $SyncBeg = \{sb_{(s,t)} \in ND_{id} \times \{y_{action}\} \times \langle ROLE \rangle \mid (s,t) \in Ed_{A\text{-}TO\text{-}As} \}$
- $Handov = \{ho_{(s,t)} \in ND_{id} \times \{y_{action}\} \times \langle ROLE \rangle \mid (s,t) \in Ed_{A\text{-}FROM\text{-}As} \}$

- $Ed_{Del} = Ed_{A\text{-}TO\text{-}As} \cup Ed_{A\text{-}FROM\text{-}As} \cup Ed_{A\text{-}IN\text{-}As} \cup Ed_{B\text{-}TO\text{-}Bs} \cup Ed_{B\text{-}FROM\text{-}Bs} \cup Ed_{B\text{-}OUT\text{-}Bs}$
  $\cup Ed_{B\text{-}IN\text{-}Bs}$
- $Ed_{Add} = Ed_{Add\_A2SyncBeg} \cup Ed_{Add\_SyncBeg2Cs} \cup Ed_{Add\_C\text{-}IN\text{-}Cs} \cup Ed_{Add\_Cs2Ho} \cup Ed_{Add\_Ho2A}$

- $Ed_{Add\_A2SyncBeg} = \{ (s, sb_{(s,t)}) \in ED \mid sb_{(s,t)} \in SyncBeg \}$
- $Ed_{Add\_SyncBeg2Cs} = \{ (sb_{(s,t)}, j) \in ED \mid sb_{(s,t)} \in SyncBeg, (t,j) \in Map_{As\text{-}Cs} , (t,j) \in Map_{Bs\text{-}Cs} \}$
- $Ed_{Add\_C\text{-}IN\text{-}Cs} = \{ (j, j') \in ED \mid j \in Nd_{Cs} \}$
- $Ed_{Add\_Cs2Ho} = \{ (j', ho_{(s,t)}) \in ED \mid ho_{(s,t)} \in Handov, (s,j') \in Map_{As\text{-}Cs} , (s',j') \in Map_{Bs\text{-}Cs} \}$
- $Ed_{Add\_Ho2A} = \{ (ho_{(s,t)}, t) \in ED \mid se_{(s,t)} \in Handov \}$

Cs is the sub-graph of those sets of nodes that are performed conjointly by Organization A and B.

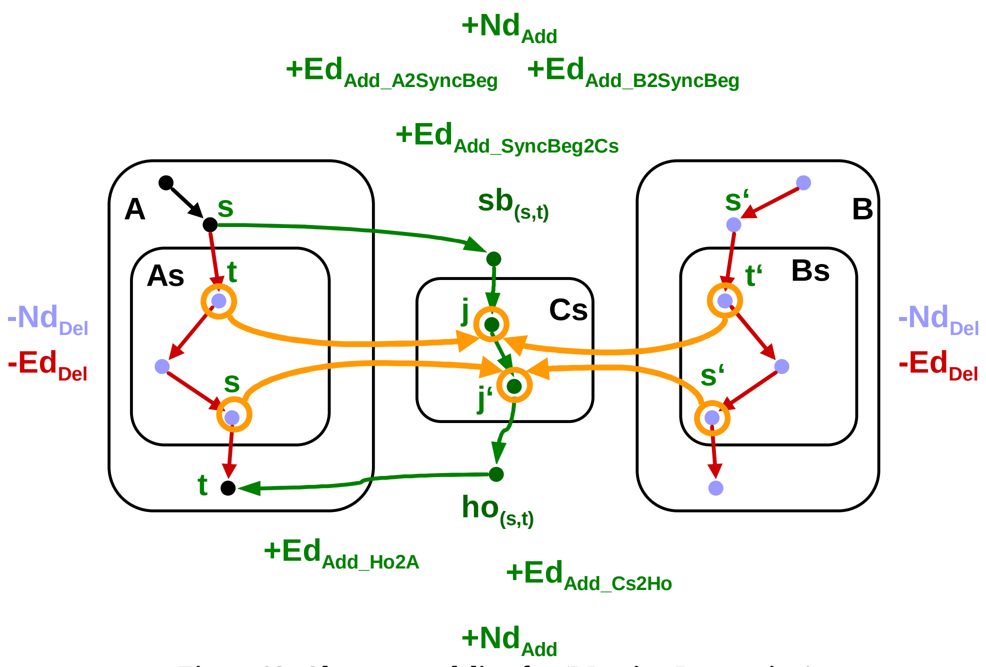


Figure 93: Abstract modeling for 'Merging Integration'

Figure 64 is referred in order to illustrate the concrete formalization. The root process of 'Merging Integration' is defined as the following:

Definition 28.

A = ({

(Start, y_{start}, Org_A), (End, y_{end}, Org_A),
(SysDesign, y_{action}, Org_A), (SysImpl, y_{action}, Org_A), (SysTest, y_{action}, Org_A),
(SysRequSpec, $y_{artifact}$, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A), (ImplSys, $y_{artifact}$, Org_A),
(SysTestRec, $y_{artifact}$, Org_A)

} , {

//[Controlflow]
((Start, y_{start}, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SysImpl, y_{action}, Org_A)),
((SysImpl, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (End, y_{end}, Org_A))

//[Dataflow]
((SysRequSpec, $y_{artifact}$, Org_A), (SysDesign, y_{action}, Org_A)),
((SysDesign, y_{action}, Org_A), (SysDesignSpec, $y_{artifact}$, Org_A)),
((SysDesignSpec, $y_{artifact}$, Org_A), (SysImpl, y_{action}, Org_A)),
((SysImpl, y_{action}, Org_A), (ImplSys, $y_{artifact}$, Org_A)),
((ImplSys, $y_{artifact}$, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A)),
((SysTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))

}).

Definition 29.

B = ({
(Start, y_{start}, Org_B), (End, y_{end}, Org_B),
(SysDesign, y_{action}, Org_B), (SysImpl, y_{action}, Org_B), (SysTest, y_{action}, Org_B),
(SysRequSpec, $y_{artifact}$, Org_B), (SysDesignSpec, $y_{artifact}$, Org_B), (ImplSys, $y_{artifact}$, Org_B),
(SysTestRec, $y_{artifact}$, Org_B)

} , {

//[Controlflow]
((Start, y_{start}, Org_B), (SysDesign, y_{action}, Org_B)),
((SysDesign, y_{action}, Org_B), (SysImpl, y_{action}, Org_B)),
((SysImpl, y_{action}, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (End, y_{end}, Org_B))

//[Dataflow]
((SysRequSpec, $y_{artifact}$, Org_B), (SysDesign, y_{action}, Org_B)),
((SysDesign, y_{action}, Org_B), (SysDesignSpec, $y_{artifact}$, Org_B)),
((SysDesignSpec, $y_{artifact}$, Org_B), (SysImpl, y_{action}, Org_B)),
((SysImpl, y_{action}, Org_B), (ImplSys, $y_{artifact}$, Org_B)),
((ImplSys, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B)),
((SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))
}).

Using Definition 27, the concrete integration operation for ‘Merging Integration’ is defined based on Figure 65 in Definition 30:

Definition 30.
integrate_merging() extends *integrate_graph(),*

where

- Nd_{Del} = (Start, y_{start}, Org_B), (End, y_{end}, Org_B),
  (SysDesign, y_{action}, Org_B), (SysImpl, y_{action}, Org_B), (SysTest, y_{action}, Org_B),
  (RequSpec, $y_{artifact}$, Org_B), (SysDesignSpec, $y_{artifact}$, Org_B), (ImplSys, $y_{artifact}$, Org_B),
  (TestRec, $y_{artifact}$, Org_B), (SysImpl, y_{action}, Org_A),

- Nd_{Add} = (SynchBeg, y_{action}, (Org_A AND Org_B)), (Handov, y_{action}, (Org_A AND Org_B)),
  (SysDesignSpec, $y_{artifact}$, (Org_A AND Org_B)), (SysImpl, y_{action}, (Org_A AND Org_B)),
  (ImplSys, $y_{artifact}$, (Org_A AND Org_B)),

- Ed_{Del} =
  //[Control flow]
  ((SysDesign, y_{action}, Org_A), (SysImpl, y_{action}, Org_A)),
  ((SysImpl, y_{action}, Org_A), (SysTest, y_{action}, Org_A))
  ((Start, y_{start}, Org_B), (SysDesign, y_{action}, Org_B)),
  ((SysDesign, y_{Action}, Org_B), (SysImpl, y_{action}, Org_B)),
  ((SysImpl, y_{Action}, Org_B), (SysTest, y_{Action}, Org_B)),
  ((SysTest, y_{Action}, Org_B), (End, y_{end}, Org_B)),

  //[Data flow]
  ((SysDesignSpec, $y_{artifact}$, Org_A), (SysImpl, y_{action}, Org_A)),
  ((SysImpl, y_{action}, Org_A), (ImplSys, $y_{artifact}$, Org_A)),

((SysRequSpec, $y_{artifact}$, Org_B), (SysDesign, y_{action}, Org_B)),
((SysDesign, y_{action}, Org_B), (SysDesignSpec, $y_{artifact}$, Org_B)),
((SysDesignSpec, $y_{artifact}$, Org_B), (SysImpl, y_{action}, Org_B)),
((SysImpl, y_{action}, Org_B), (ImplSys, $y_{artifact}$, Org_B)),
((ImplSys, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B)),
((SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))

- Ed_{Add} =

//[Control flow]
((SysDesign, y_{action}, Org_A), (SynchBeg, y_{action}, (Org_A AND Org_B))),
((SynchBeg, y_{action}, (Org_A AND Org_B)), (SysImpl, y_{action}, (Org_A AND Org_B))),
((SysImpl, y_{action}, (Org_A AND Org_B)), (Handov, y_{action}, (Org_A AND Org_B))),
((Handov, y_{action}, (Org_A AND Org_B)), (SysTest, y_{action}, Org_A)),

//[Data flow]
((SysDesignSpec, $y_{artifact}$, Org_A), (SynchBeg, y_{action}, (Org_A AND Org_B))),
((SynchBeg,y_{action},(Org_A AND Org_B)), (SysDesignSpec, $y_{artifact}$, (Org_A AND Org_B))),
((SysDesignSpec, $y_{artifact}$, (Org_A AND Org_B)), (SysImpl, y_{action}, (Org_A AND Org_B))),
((SysImpl, y_{action}, (Org_A AND Org_B)), (ImplSys, y_{action}, (Org_A AND Org_B))),
((ImplSys, y_{action}, (Org_A AND Org_B)), (Handov, y_{action}, (Org_A AND Org_B))),
((Handov, y_{action}, (Org_A AND Org_B)), (ImplSys, $y_{artifact}$, Org_A))

4.3.10 Hierarchical Integration

Hierarchical process integration consists of several integration steps and makes use of already- existing integration methods and tools that have been previously defined. The major idea is that the hierarchy is dissolved ("flattened") and consequently integration is done without having hierarchical process. After integration, a new hierarchy is defined. These steps are step- by- step illustrated in the following.

Step 1: Basic processes for hierarchical integration

Figure 94 depicts the root processes for hierarchical integration. It is assumed that both organizations have hierarchical actions (HA-nodes with dotted line) in place. Within nd_{HA} nodes, the appropriate ad_{As} graphs are defined.

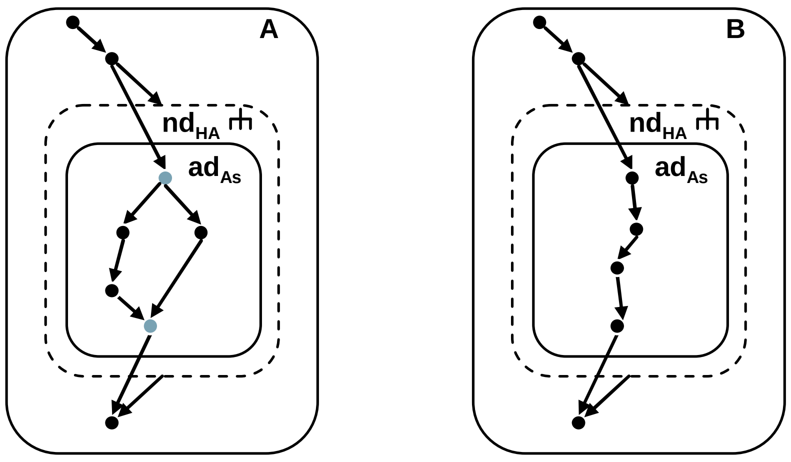


Figure 94: Root Processes for Hierarchical Integration

Step 2: Deletion of hierarchical elements

In order to prepare processes for integration, hierarchy is dissolved (= ‘temporarily deleted’). This is illustrated in Figure 95.

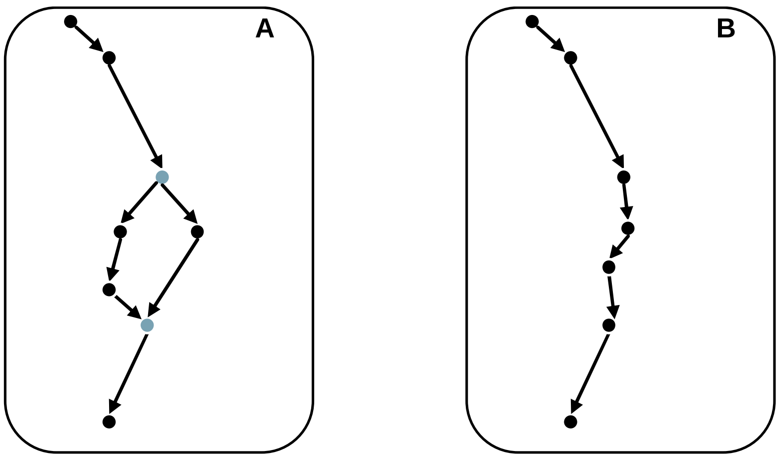


Figure 95: Dissolution of Hierarchical Processes

The respective function $\overrightarrow{delete\_hierarchy}$ is defined as follows:

Definition 31.

$$\overrightarrow{delete\_hierarchy} : AD \times AD \to AD \cup \{\otimes\}$$

$$\overrightarrow{delete\_hierarchy}\left(ad_A = (Nd, Ed), nd_{HA}\right) =_{DEF} \begin{cases} \left(ND \setminus \{\mathrm{nd}_{HA}\}, Ed \setminus \bigcup_{nd \in Nd} \left\{ (\mathrm{nd}, \mathrm{nd}_{HA}), (\mathrm{nd}_{HA}, \mathrm{nd}) \right\} \right) \\ \qquad \bullet\ \mathrm{nd}_{HA} \text{ is a hierarchical action node} \\ \otimes \qquad \textit{otherwise} \end{cases}$$

Step 3: Integration of action/activities

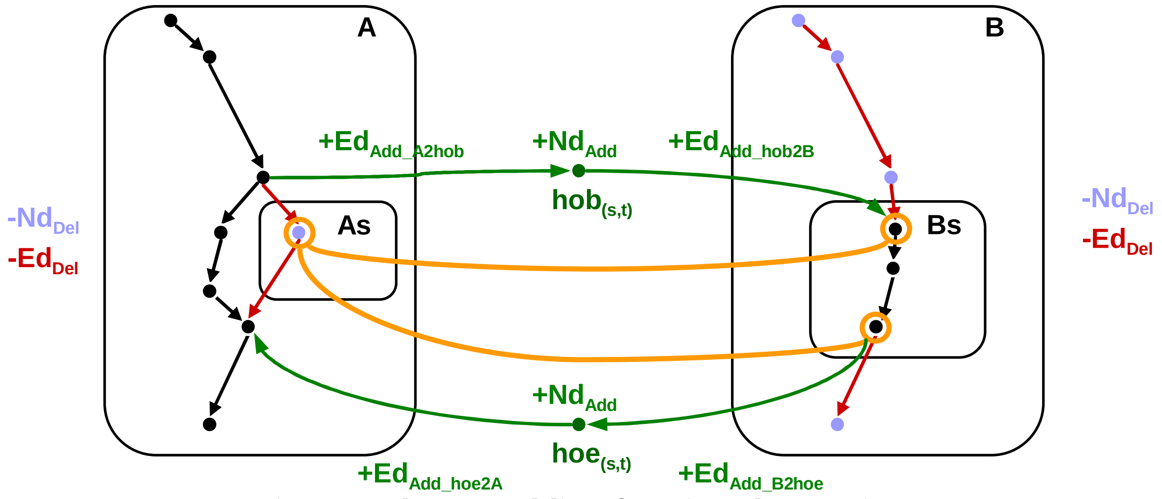


Figure 96: Abstract modeling of ‘Horizontal Integration’

Next, integration is done according to familiar methods and tools. The sub-graphs As and Bs need to be defined first, which symbolize the process nodes to be replaced (As) and to be integrated (Bs). Figure 96 shows the respective integration procedure using the mediator for 'Horizontal Integration'.

Definition 32.

integrate_horiz() extends *integrate_graph(),*
where

- $Nd_{Del} = Nd_{As} \cup (Nd_B \setminus Nd_{Bs})$
- $Nd_{Add} = HandovBeg \cup HandovEnd$
- $HandovBeg = \{hob_{(s,t)} \in ND_{id} \times \{y_{action}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}TO\text{-}As} \}$
- $HandovEnd = \{hoe_{(s,t)} \in ND_{id} \times \{y_{action}\} \times <ROLE> | (s,t) \in Ed_{A\text{-}FROM\text{-}As} \}$
- $Ed_{Del} = Ed_{A\text{-}TO\text{-}As} \cup Ed_{A\text{-}FROM\text{-}As} \cup Ed_{A\text{-}IN\text{-}As}$
  $\cup \quad Ed_{B\text{-}TO\text{-}Bs} \cup Ed_{B\text{-}FROM\text{-}Bs} \cup Ed_{B\text{-}OUT\text{-}Bs}$
- $Ed_{Add} = Ed_{AddA2Sync} \cup Ed_{AddSync2B} \cup Ed_{AddB2Sync} \cup Ed_{AddSync2A}$
- $Ed_{Add\_A2hob} = \{ (s, hob_{(s,t)}) \in ED | hob_{(s,t)} \in HandovBeg \}$
- $Ed_{Add\_hob2B} = \{ (hob_{(s,t)},t') \in ED | hob_{(s,t)} \in HandovBeg, (t,t') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_B2hoe} = \{ (s',hoe_{(s,t)}) \in ED | hoe_{(s,t)} \in HandovEnd, (s,s') \in Map_{As\text{-}Bs}\}$
- $Ed_{Add\_hoe2A} = \{ (hoe_{(s,t)}, t) \in ED | hoe_{(s,t)} \in HandovEnd \}$

Step 4: Re-definition of Hierarchy

The last step includes re-definition of hierarchy (Figure 97). Thereby, the original hierarchies are used as orientation for new the hierarchy and will be incorporated as far as it makes sense and it is possible. This decision is not automated, but will be taken by a process engineer.

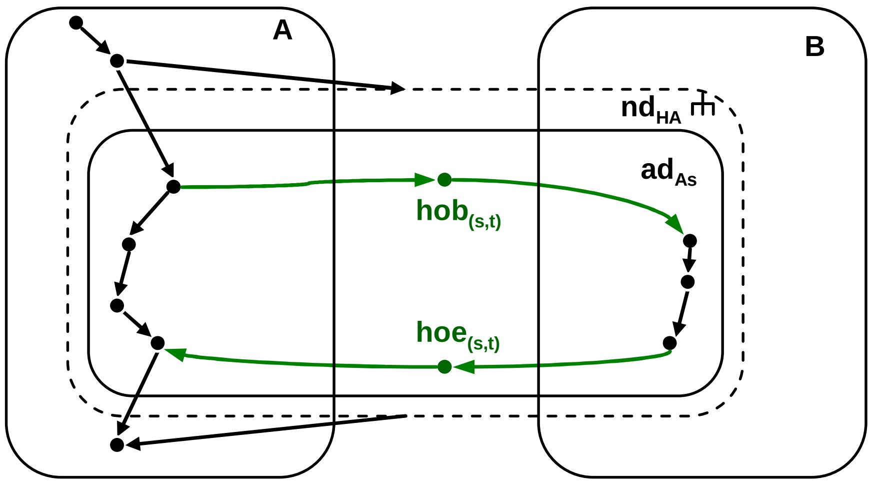


Figure 97: Re-defined Hierarchy

The creation of hierarchy follows the function $\overrightarrow{create\_hierarchy}$, which is defined in Definition 5.

Having the abstract definition in Figure 94 until Figure 97 depicted the concrete integration steps 1 – 4 are accordingly described in the following.

Step 1:

The basic processes for hierarchical integration according to Figure 72 are described below.

Definition 33.

A = ({
(Start, y_{start}), (End, y_{end}),
(RequEng, y_{action}, Org_A), (SWImpl, y_{HA}, Org_A, $SWImpl_{HA}$), (SysTest, y_{action}, Org_A),
(RequSpec, $y_{artifact}$, Org_A), (ImplSW, $y_{artifact}$, Org_A), (SysTestRec, $y_{artifact}$, Org_A),
(MarketEval, y_{action}, Org_A), (MarketInfo, $y_{artifact}$, Org_A)
} , {

$SWImpl_{HA}$ = {(Start, y_{start}, Org_A), (End, y_{end}, Org_A), ($SWReal_A$, y_{action}, Org_A),
(RealSW, $y_{artifact}$, Org_A), (SWModTest, y_{action}, Org_A),
(SWTestRec, $y_{artifact}$, Org_A)
}

//[Control flow]
$SWImpl_{HA}$= {
((Start, y_{start}, Org_A), (SWReal, y_{action}, Org_A)),
((SWReal, y_{action}, Org_A), (SWModTest, y_{action}, Org_A)),
((SWModTest, y_{action}, Org_A), (End, y_{end}, Org_A)),
}

((Start, y_{start}, Org_A), (MarketEval, y_{action}, Org_A)),
((MarketEval, y_{action}, Org_A), (RequEng, y_{action}, Org_A)),
((RequEng, y_{action}), (SWImpl, y_{ha}, Org_A, $SWImpl_{HA}$)),
((SWImpl, y_{ha}, Org_A, $SWImpl_{HA}$), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (End, y_{end}, Org_A))

//[Data flow]
$SWImpl_{HA}$ = {
((SWReal, y_{action}, Org_A), (RealSW, $y_{artifact}$, Org_A)),
((RealSW, $y_{artifact}$, Org_A), (SWModTest, y_{action}, Org_A)),
((SWModTest, y_{action}, Org_A), (SWTestRec, $y_{artifact}$, Org_A)),
((SWTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))
}

((MarketEval, y_{action}, Org_A), (MarketInfo, $y_{artifact}$, Org_A)),
((MarketInfo, $y_{artifact}$, Org_A), (RequEng, y_{action}, Org_A)),
((RequEng, y_{action}, Org_A), (RequSpec, $y_{artifact}$, Org_A)),
((RequSpec, $y_{artifact}$, Org_A), (SWImpl, y_{ha}, Org_A, $SWImpl_{HA}$)),
((SWImpl, y_{ha}, Org_A, $SWImpl_{HA}$), (ImplSW, $y_{artifact}$, Org_A)),
((ImplSW, $y_{artifact}$, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A)),
((SysTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))
}).

B = ({
(Start, y_{start}, Org_B), (End, y_{end}, Org_B), (MarketEval, y_{action}, Org_B),
(MarketInfo, $y_{artifact}$, Org_B), (SysImpl, y_{ha}, Org_B, $SysImpl_{HA}$),

(RequEng, y_{action}, Org_B), (SysTest, y_{action}, Org_B), (RequSpec, $y_{artifact}$, Org_B),
(ImplSys, $y_{artifact}$, Org_B), (SysTestRec, $y_{artifact}$, Org_B)

$SysImpl_{HA}$ = {
(Start$_B$, y_{start}, Org_B), (End$_B$, y_{end}, Org_B), (SWDesign$_B$, y_{action}, Org_B),
(SWImpl, y_{action}, Org_B), (SWTest, y_{action}, Org_B),
(SWDesignSpec, $y_{artifact}$, Org_B), (ImplSW, $y_{artifact}$, Org_B),
(SWTestRec, $y_{artifact}$, Org_B)
}

} , {

//[Control flow]
$SysImpl_{HA}$ = {
((Start, y_{start}, Org_B), (SWDesign, y_{action}, Org_B)),
((SWDesign, y_{action}, Org_B), (SWImpl, y_{action}, Org_B)),
((SWImpl, y_{action}, Org_B), (SWTest, y_{action}, Org_B)),
((SWTest, y_{action}, Org_B), (End, y_{end}, Org_B))
}

((Start, y_{start}, Org_B), (MarketEval, y_{action}, Org_B)),
((MarketEval, y_{action}, Org_B), (RequEng, y_{action}, Org_B)),
((RequEng, y_{action}, Org_B), (SysImpl, y_{ha}, Org_B, $SysImpl_{HA}$)),
((SysImpl, y_{ha}, Org_B, $SysImpl_{HA}$), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (End, y_{end}, Org_B))

//[Data flow]
$SysImpl_{HA}$ = {
((SWDesign, y_{action}, Org_B), (SWDesignSpec, $y_{artifact}$, Org_B)),
((SWDesignSpec, $y_{artifact}$, Org_B), (SWImpl, y_{action}, Org_B)),
((SWImpl, y_{action}, Org_B), (ImplSW, $y_{artifact}$, Org_B)),
((ImplSW, $y_{artifact}$, Org_B), (SWTest, y_{action}, Org_B)),
((SWTest, y_{action}, Org_B), (SWTestRec, $y_{artifact}$, Org_B)),
((SWTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))
}

((MarketEval, y_{action}, Org_B), (MarketInfo, $y_{artifact}$, Org_B)),
((MarketInfo, $y_{artifact}$, Org_B), (RequEng, y_{action}, Org_B)),
((RequEng, y_{action}, Org_B), (RequSpec, $y_{artifact}$, Org_B)),
((RequSpec, $y_{artifact}$, Org_B), (SysImpl, y_{ha}, Org_B, $SysImpl_{HA}$)),
((SysImpl, y_{ha}, Org_B, $SysImpl_{HA}$), (ImplSys, $y_{artifact}$, Org_B)),
((ImplSys, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B)),
((SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))
}).

Step 2:

The root processes in Figure 72 need to be prepared for integration, which shall be done by leaving out hierarchical elements. This is done using the function $\overrightarrow{delete\_hierarchy}$

Definition 31).Definition 34 As illustrated in Figure 74, Definition 34 showns the formal definition accordingly.

Definition 34.

A = ({
(Start, y_{start}, Org_A), (End, y_{end}, Org_A),
(MarketEval, y_{action}, Org_A), (RequEng, y_{action}, Org_A), (SysTest, y_{action}, Org_A),
(MarketInfo, $y_{artifact}$, Org_A), (RequSpec, $y_{artifact}$, Org_A), (SysTestRec, $y_{artifact}$, Org_A),
($SWReal_A$, y_{action}, Org_A), (RealSW, $y_{artifact}$, Org_A), (SWModTest, y_{action}, Org_A),
(SWTestRec, $y_{artifact}$, Org_A)

} , {

//[Control flow]
((Start, y_{start}, Org_A), (MarketEval, y_{action}, Org_A)),
((MarketEval, y_{action}, Org_A), (RequEng, y_{action}, Org_A)),
((RequEng, y_{action}, Org_A), (SWReal, y_{action}, Org_A)),
((SWReal, y_{action}, Org_A), (SWModTest, y_{action}, Org_A)),
((SWModTest, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (End, y_{end}, Org_A))

//[Data flow]
((MarketEval, y_{action}, Org_A), (MarketInfo, $y_{artifact}$, Org_A)),
((MarketInfo, $y_{artifact}$, Org_A), (RequEng, y_{action}, Org_A)),
((RequEng, y_{action}, Org_A), (RequSpec, $y_{artifact}$, Org_A)),
((RequSpec, $y_{artifact}$, Org_A), (SWReal, y_{action}, Org_A),
((SWReal, y_{action}, Org_A), (RealSW, $y_{artifact}$, Org_A)),
((RealSW, $y_{artifact}$, Org_A), (SWModTest, y_{action}, Org_A)),
((SWModTest, y_{action}, Org_A), (SWTestRec, $y_{artifact}$, Org_A)),
((SWTestRec, $y_{artifact}$, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A)),
((SysTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))
}).

B = ({
(Start, y_{start}, Org_B), (End, y_{end}, Org_B), (MarketEval, y_{action}, Org_B), (RequEng, y_{action}, Org_B),
(SysTest, y_{action}, Org_B), (MarketInfo, $y_{artifact}$, Org_B), (RequSpec, $y_{artifact}$, Org_B),
(SysTestRec, $y_{artifact}$, Org_B), ($SWDesign_B$, y_{action}, Org_B), (SWImpl, y_{action}, Org_B),
(SWTest, y_{action}, Org_B), (SWDesignSpec, $y_{artifact}$, Org_B), (ImplSW, $y_{artifact}$, Org_B),
(SWTestRec, $y_{artifact}$, Org_B)

} , {

//[Control flow]
((Start, y_{start}, Org_B), (MarketEval, y_{action}, Org_B)),
((MarketEval, y_{action}, Org_B), (RequEng, y_{action}, Org_B)),
((RequEng, y_{action}, Org_B), (SWDesign, y_{action}, Org_B)),
((SWDesign, y_{action}, Org_B), (SWImpl, y_{action}, Org_B)),
((SWImpl, y_{action}, Org_B), (SWTest, y_{action}, Org_B)),
((SWTest, y_{action}, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (End, y_{end}, Org_B))

//[Data flow]
((MarketEval, y_{action}, Org_B), (MarketInfo, $y_{artifact}$, Org_B)),
((MarketInfo, $y_{artifact}$, Org_B), (RequEng, y_{action}, Org_B)),
((RequEng, y_{action}, Org_B), (RequSpec, $y_{artifact}$, Org_B)),
((RequSpec, $y_{artifact}$, Org_B), (SWDesign, y_{action}, Org_B)),
((SWDesign, y_{action}, Org_B), (SWDesignSpec, $y_{artifact}$, Org_B)),
((SWDesignSpec, $y_{artifact}$, Org_B), (SWImpl, y_{action}, Org_B)),
((SWImpl, y_{action}, Org_B), (ImplSW, $y_{artifact}$, Org_B)),
((ImplSW, $y_{artifact}$, Org_B), (SWTest, y_{action}, Org_B)),
((SWTest, y_{action}, Org_B), (SWTestRec, $y_{artifact}$, Org_B)),
((SWTestRec, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B)),
((SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))
}).

Step 3:

As defined in Figure 74, hierarchical processes are integrated by dissolving the hierarchy first. Then, processes are integrated by using existing mediators. According to Figure 96, the mediator for *Horizontal Integration* (chapter 4.3.6) is used.

Definition 35.

integrate_horiz() extends *integrate_graph()*,
where

- Nd_{Del} =
  //[Organization A]
  (RequEng, y_{action}, Org_A), (RequSpec, $y_{artifact}$, Org_A), (SWReal, y_{action}, Org_A)

  //[OrganizationB]
  (Start, y_{start}, Org_B), (MarketEval, y_{action}, Org_B), (MarketInfo, $y_{artifact}$, Org_B),
  (SWTest, y_{action}, Org_B), (SWTestRec, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B),
  (SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B)

- Nd_{Add} = (HandovBeg, y_{action}, (Org_A AND Org_B)), (HandovEnd, y_{action}, (Org_A AND Org_B)),

- Ed_{Del} =
  //[Control flow]
  //[Organization A]
  ((MarketEval, y_{action}, Org_A), (RequEng, y_{action}, Org_A)),
  ((RequEng, y_{action}, Org_A), (SWReal, y_{action}, Org_A)),
  ((SWReal, y_{action}, Org_A), (SWModTest, y_{action}, Org_A)),

  //[Organization B]
  ((Start, y_{start}, Org_B), (MarketEval, y_{action}, Org_B)),
  ((MarketEval, y_{action}, Org_B), (RequEng, y_{action}, Org_B)),
  ((SWImpl, y_{action}), (SWTest, y_{action})),
  ((SWTest, y_{action}), (SysTest, y_{action}, Org_B)),
  ((SysTest, y_{action}, Org_B), (End, y_{end}, Org_B))

//[Data flow]
//[Organization A]
((MarketInfo, $y_{artifact}$, Org_A), (RequEng, y_{action}, Org_A)),
((RequEng, y_{action}, Org_A), (RequSpec, $y_{artifact}$, Org_A)),
((RequSpec, $y_{artifact}$, Org_A), (SWReal, y_{action}, Org_A),
(($SWReal_A$, y_{action}, Org_A), (RealSW, $y_{artifact}$, Org_A)),

//[Organization B]
((MarketEval, y_{action}, Org_B), (MarketInfo, $y_{artifact}$, Org_B)),
((ImplSW, $y_{artifact}$, Org_B), (SWTest, y_{action}, Org_B)),
((SWTest, y_{action}, Org_B), (SWTestRec, $y_{artifact}$, Org_B)),
((SWTestRec, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_B)),
((SysTest, y_{action}, Org_B), (SysTestRec, $y_{artifact}$, Org_B)),
((SysTestRec, $y_{artifact}$, Org_B), (End, y_{end}, Org_B))

- Ed_{Add} =

//[Control flow]
((MarketEval, y_{action}, Org_A), (HandovBeg, y_{action}, (Org_A AND Org_B))),
((HandovBeg, y_{action}, (Org_A AND Org_B)), RequEng, y_{action}, Org_B)),
((SWImpl, y_{action}, Org_B), (HandovEnd, y_{action}, (Org_A AND Org_B))),
((HandovEnd, y_{action}, (Org_A AND Org_B)), (SWModTest, y_{action}, Org_A))

//[Data flow]
((MarketInfo, $y_{artifact}$, Org_A), (HandovBeg, y_{action}, (Org_A AND Org_B))),
((HandovBeg, y_{action}, (Org_A AND Org_B)), (MarketInfo, $y_{artifact}$, Org_B)),
((ImplSW, $y_{artifact}$, Org_B), (HandovEnd, y_{action}, (Org_A AND Org_B))),
((HandovEnd, y_{action}, (Org_A AND Org_B)), (RealSW, $y_{artifact}$, Org_A))

Step 4:

After integration of processes, hierarchical elements are re-defined again. Formally, this is generated by using the function $\overrightarrow{create\_hierarchy}$ (Definition 5) according to Figure 97. This re-definition should be based on former hierarchical definitions and appropriate actions as depicted in
Figure 76. The formal description of the collaborative hierarchical process (Chp) is decribed below:

Chp = ({
(Start, y_{start}, Org_A), (End, y_{end}, Org_A),
(MarketEval, y_{action}, Org_A), (MarketInfo, $y_{artifact}$, Org_A), (MarketInfo, $y_{artifact}$, Org_B),
(HandovBeg, y_{action}, (Org_A AND Org_B)), (RequEng, y_{action}, Org_B),
(RequSpec, $y_{artifact}$, Org_B), (SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A),
(SWImpl, y_{ha}, (Org_A, (Org_A AND Org_B), Org_B), $SWImpl_{HA}$)

} , {

$SWImpl_{HA}$ = {
(SWDesign, y_{action}, Org_B), (SWDesignSpec, $y_{artifact}$, Org_B),
(SWImpl, y_{action}, Org_B), (ImplSW, $y_{artifact}$, Org_B),
(HandovEnd, y_{action}, (Org_A AND Org_B)), (RealSW, $y_{artifact}$, Org_A),

(SWModTest, y_{action}, Org_A), (SWTestRec, $y_{artifact}$, Org_A)
}

//[Control flow]
$SWImpl_{HA}$ = {
((SWDesign, y_{action}, Org_B), (SWImpl, y_{action}, Org_B)),
((SWImpl, y_{action}, Org_B), (HandovEnd, y_{action}, (Org_A AND Org_B)),
((HandovEnd, y_{action}, (Org_A AND Org_B)), (SWModTest, y_{action}, Org_A)),
}

((Start, y_{start}, Org_A), (MarketEval, y_{action}, Org_A)),
((MarketEval, y_{action}, Org_A), (HandovBeg, y_{action}, (Org_A AND Org_B))),
((HandovBeg, y_{action}, (Org_A AND Org_B)), (RequEng, y_{action}, Org_B)),
((RequEng, y_{action}, Org_B), (SWDesign, y_{action}, Org_B)),

// additional control flows begin
((RequEng, y_{action}, Org_B), (SWImpl, y_{ha}, (Org_A, (Org_A AND Org_B), Org_B), $SWImpl_{HA}$)),
((SWImpl, y_{ha}, (Org_A, (Org_A AND Org_B), Org_B), $SWImpl_{HA}$), (SysTest, y_{action}, Org_A)),
// additional control flows: end

((SWModTest, y_{action}, Org_A), (SysTest, y_{action}, Org_A)),
((SysTest, y_{action}, Org_A), (End, y_{end}, Org_A))

//[Data flow]
$SWImpl_{HA}$ = {
((SWDesign, y_{action}, Org_B), (SWDesignSpec, $y_{artifact}$, Org_B)),
((SWDesignSpec, $y_{artifact}$, Org_B), (SWImpl, y_{action}, Org_B)),
((SWImpl, y_{action}, Org_B), (ImplSW, $y_{artifact}$, Org_B)),
((ImplSW, $y_{artifact}$, Org_B), (HandovEnd, y_{action}, (Org_A AND Org_B))),
((HandovEnd, y_{action}, (Org_A AND Org_B)), (RealSW, $y_{artifact}$, Org_A)),
((ImplSW, $y_{artifact}$, Org_A), (SWModTest, y_{action}, Org_A)),
((SWModTest, y_{action}, Org_A), (SWTestRec, $y_{artifact}$, Org_A))
}

((MarketEval, y_{action}, Org_A), (MarketInfo, $y_{artifact}$, Org_A)),
((MarketInfo, $y_{artifact}$, Org_A), (HandovBeg, y_{action}, (Org_A AND Org_B))),
((HandovBeg, y_{action}, (Org_A AND Org_B)), (MarketInfo, $y_{artifact}$, Org_B)),
((MarketInfo, $y_{artifact}$, Org_B), (RequEng, y_{action}, Org_B)),
((RequEng, y_{action}, Org_B), (RequSpec, $y_{artifact}$, Org_B)),
((RequSpec, $y_{artifact}$, Org_B), (SWDesign, y_{action}, Org_B)),
((SWDesign, y_{action}, Org_B), SWDesignSpec, $y_{artifact}$, Org_B)), //hierarchical process
((SWDesignSpec, $y_{artifact}$, Org_B), (SWImpl, y_{action}, Org_B)), //hierarchical process
((SWImpl, y_{action}, Org_B), (ImplSW, $y_{artifact}$, Org_B)), //…
((ImplSW, $y_{artifact}$, Org_B), (HandovEnd, y_{action}, (Org_A AND Org_B))), //…
((HandovEnd, y_{action}, (Org_A AND Org_B)), (RealSW, $y_{artifact}$, Org_A)), //…
((RealSW, $y_{artifact}$, Org_A), (SWModTest, y_{action}, Org_A)), //…
((SWModTest, y_{action}, Org_A), (SWTestRec, $y_{artifact}$, Org_A)), //…
((SWTestRec, $y_{artifact}$, Org_A), (SysTest, y_{action}, Org_A)), //hierarchical process
((SysTest, y_{action}, Org_A), (SysTestRec, $y_{artifact}$, Org_A)),
((SysTestRec, $y_{artifact}$, Org_A), (End, y_{end}, Org_A))
}).

5 Case Study

5.1 General Aspects

A case study is a research methodology common in social science. It is based on an in-depth investigation of a single individual, group, or event. It provides a systematic way of looking at events, collecting data, analyzing information, and reporting the results. As a result, the case study conductor may gain a sharpened understanding of why any instance happens as it does, and what might become important to examine more extensively in future research.

There are several types of case studies defined, which include the following [26]:

Illustrative Case Studies
These are mostly descriptive studies taking one or two instances of an event to show what a situation is like. Illustrative case studies serve primarily to make the unfamiliar familiar and to give readers a common language about the topic in question.

Exploratory (or pilot) Case Studies
These are condensed case studies performed before implementing a large scale investigation. Their basic purpose is to help identify questions and select types of measurement prior to the main investigation. The primary pitfall of this type of study is that initial findings may seem convincing enough to be released prematurely as conclusions.

Cumulative Case Studies
These serve to aggregate information from several sites collected at different times. The idea behind these studies is the collection of past studies will allow for greater generalization without additional cost or time expenditures on new, possibly repetitive studies.

Critical Instance Case Studies
These examine one or more sites for either the purpose of examining a situation of unique interest with little to no interest in generalizing, or to call into question or challenge a highly generalized or universal assertion. This method is useful for answering cause and effect questions.

Many case study supporters indicate that case studies generate much more detailed information than what is available through a statistical analysis. However, Flvybjerg [45] identifies and discusses five misunderstandings about case study research:

1. General, theoretical knowledge is more valuable than concrete, practical knowledge.
2. One cannot generalize on the basis of an individual case and, therefore, the case study cannot contribute to scientific development.
3. The case study is most useful for generating hypotheses; whereas, other methods are more suitable for hypotheses testing and theory building.
4. The case study contains a bias toward verification, i.e., a tendency to confirm the researcher’s preconceived notions.
5. It is often difficult to summarize and develop general propositions and theories on the basis of specific case studies.

The set-up of an adequate case study design for the problem to be solved is very challenging. On one hand, there is no doubt that one single (stand-alone) case study will not deliver sufficient evidence so that any theory is empirically proved (See misunderstanding #2 above

[45]). On the other hand, the set-up of an industrial collaborative scenario is very time-consuming and expects solid efforts from all participating parties in the early and start-up phase as well as during conduction. One reason is that in a global project, setting up the team is dispersed all over the world. This makes, other than project communication, e.g., for additional case study related- discussion very complex. Therefore, the first case study (Chapter 5.2) follows the “Illustrative Case Study” design mainly focusing on the understanding of the problem and to get a feeling how the collaborative approach for process integration of this work is applied.

5.2 Case Study 1: Scenario from Automobile Industry

This case study is basically defined as an “**Illustrative Case Study**” and follows the purpose becoming familiar with typical issues to be solved with the above- defined process integration method.

The illustrative case study takes advantage from some very well- known scientific approaches. This means that the set- up makes use of “Direct Observation”, which is a source of evidence in [150]. These observations have been done prior to and during definition of the solution in this dissertation. Therefore, this process integration approach structurally documents those activities that companies might have done anyway to some extent for collaborative projects.

In conjunction with direct observation, the pattern-matching-strategy is used. This approach has also been applied to define those patterns that build the basic pillars of the integration approach. Actually, this strategy is used for data evaluation purposes in computer science and tries to map real data pattern with theoretically- supposed result patterns [66]. However, the defined patterns of this work are not derived on a representative set of data; moreover, the patterns have been defined based on direct observations from running collaborative projects in industry. Thereby, several process patterns have been identified that allow for modeling each and every scenario, which occurs in global software development organizations [4].

The case study starts with an application scenario that illustrates which functionality of software or system will be needed by a potential customer. Next, a development scenario is shown that depicts the major development parties and (sub-) systems to be developed. These systems and (sub-) systems are marked in bold in chapter 5.2.2.

5.2.1 Case Study Questions

This case study deals with the issue of two or more development organizations' abilities to collaborate on a processes basis. As already stated in section 1.2, this generates the following research questions:

1. How does a (new) collaborative process (control flow) look like that incorporates two or more different processes from cooperating organizations?
2. How do artifacts (data flow) look like? How can artifacts be handled if they are defined in different formats?

Additionally, the case study tries to open up additional questions for further research, especially concerning interface definition and artifact handling.

5.2.2 Application Scenario

A car goes along on a long journey. At any point of time an installed **Road Condition Analysis** system (road side) identifies that due to rain and heavily dropping temperatures 30 km ahead, the probability of slippery roads due to black ice increases sharply.

This is possible, because the **Roadside Information Systems** informs every car at a distance of less than 50 km. Additionally, the **Driver Supporting System** of the car receives that information and immediately checks the cars' functionality. Thereby, breaking system, injection control system, security system, and oil control system are controlled towards functional capability and the quality of the upcoming weather and road conditions. In case of any problem, the **Self Repair System** takes care of it.

The **Roadside Information System** also provides the density of vehicles on the road, which is used to calculate the probability of traffic jams and accidents in the area. If this calculated probability is significantly high, the **Driver Supporting System** checks the availability of an auto rail station in a reachable proximity and if slots for car transportation are still available.

If the **Self Repair System** could not fix any potential problem, the direct way to the next car service point is identified. The availability of necessary spare parts is also requested. This information is communicated to the driver, who can decide whether to go to the car service point, to use the auto rail station for the remaining journey, or to just continue with the car on the road.

The **Navigation System** shows the driver the selected option and calculated bypasses in case of traffic jams. If no bypasses are available, the **Driver Supporting System** calculates the approximated waiting time and suggests hotels for accommodation. If desired by the driver, a hotel room is booked automatically.

5.2.3 Development Scenario

As the scenario above shows, there are several systems to be developed for realizing this application case:

a. Roadside Information System, including Road Condition Analysis
b. Driver Supporting System, including the Self Repair System
c. Navigation System

That means the adaption of the car control system (combustion, breaks, safety systems - airbags etc.) and the new development of the Roadside Information System and the connection to Road Condition Analysis, Driver Supporting System, and Self Repair System.

This scenario contains the following organizations as participating partners:

- Organization A is an automobile supplier delivering the entire protection and safety systems.
- Organization B is specialized on Driver Supporting Systems
- Organization C develops software systems
- Organization D is a small cap specialized on software development for data transfers
- Organization E is a hardware manufacturer and supplier with a partially defined hardware process

The development relationship of organizations and systems is depicted in Figure 98. It is assumed that organization A provides the basis software functionality ("platform") upon, which all other suppliers' software contributions are based.

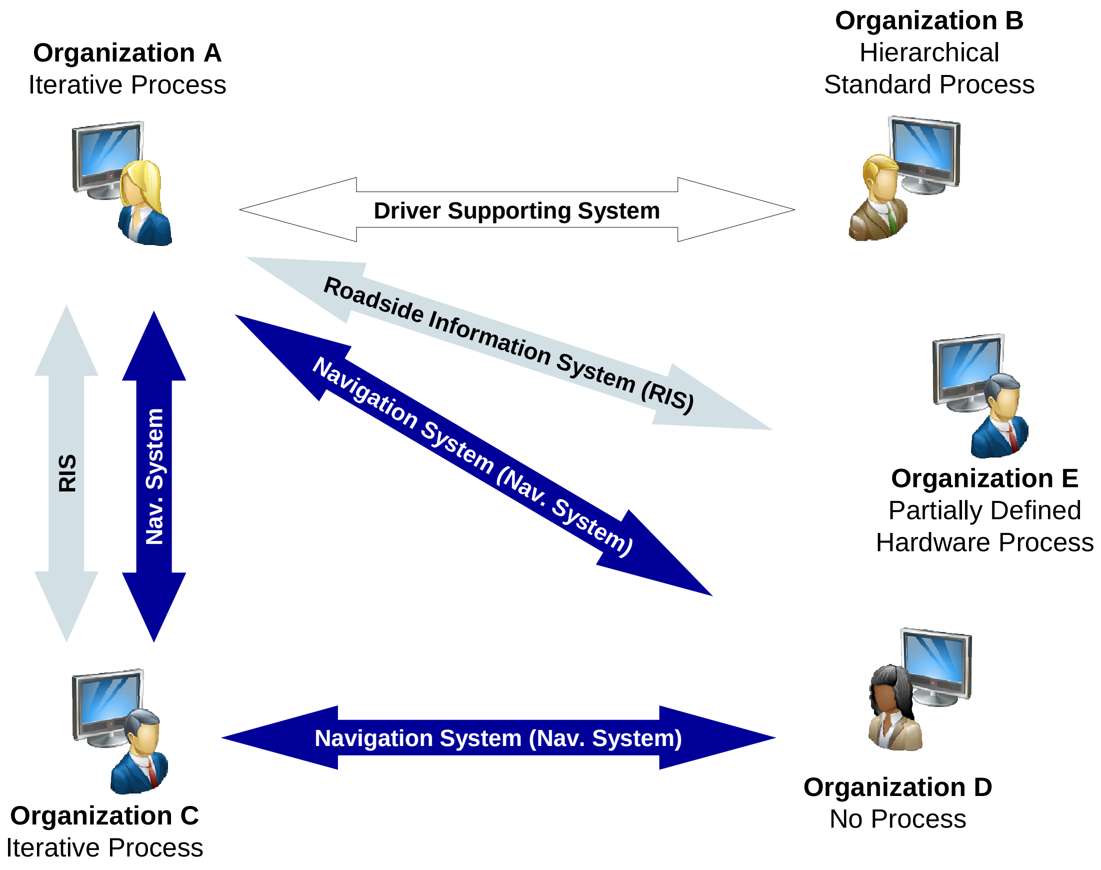

Organization A
Iterative Process
Organization B
Hierarchical Standard Process
Driver Supporting System
Roadside Information System (RIS)
Navigation System (Nav. System)
RIS
Nav. System
Organization E
Partially Defined Hardware Process
Navigation System (Nav. System)
Organization C
Iterative Process
Organization D
No Process

Figure 98: Organizational Relationship of Development Scenario

The **Driver Supporting System** is conjointly developed by Organization A and B, which is specialized on such systems.

The **Roadside Information System** is created by Organization A, C, and E. In this case, A delivers all the sensors and controls, C produces with the software for data analysis and transition. Organization D is responsible for services such as contacting the auto rail station or hotels.

5.2.4 Organizational Process Definitions

Organization A (Master)
As the Master of the collaborative development scenario, Organization A uses for its platform development a comprehensive iterative process, since platform development is usually a longer lasting process. This platform functions as basis and needs more sophisticated mechanisms (change request / claim management) than a simple application development. In order to keep the scenario as simple as possible, sub-processes like change/claim management are omitted, since they have only a supportive character.

The process is illustrated in Figure 99 and starts with *Requirement Engineering*, which is sourced by *Market Information*. Based on the *Requirement Specification* the organization conducts *System Architecture & Design*, which is the basis for all further development contributions and is documented in *System Architecture & Design Specification*. In the following *Module Design* addresses only those functionalities that are developed by the Master itself. The design results in *Module Specification* that is the basis for *Module Implementation*. After *Module Test*, which is documented in *Module Test Record*, the iteration loop checks whether there is still remaining functionality to be designed and implemented. If so, the control flow jumps to *Module Design*. In case all desired modules have been implemented, *System Test* is conducted, which ends up with *System Test Record*.

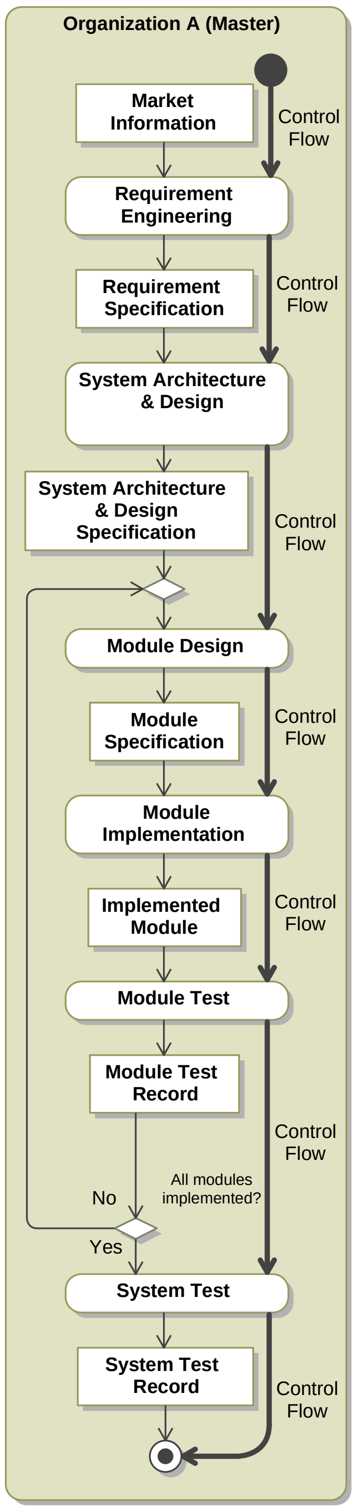


Figure 99: Iterative Process of Organization A

Organization B (Supplier)
Organization B follows a kind of “standard” process, which is hierarchically defined for better complexity handling (Figure 100). Having *Requirement Specification* as input, the process starts with *System Design*. An explicit requirement engineering action is not defined. After *Design Review* of the *Design Specification System*, *System Implementation* – as a hierarchical action – is conducted. *System Implementation*, which is basically the development of software, is sourced by *System Design Specification* from which *Software Design* and a respective *Design Review* is derived. The resulting *Software Design Specification* is input for *Software Implementation*. The *Software Test* of the implemented software is recorded in the *Software Test Record*.

Remarks:

For integration purposes the “hybrid view” has been defined in chapter 4.2.6, Figure 73. However, this view is not shown initially since manageability of the process suffers from it.

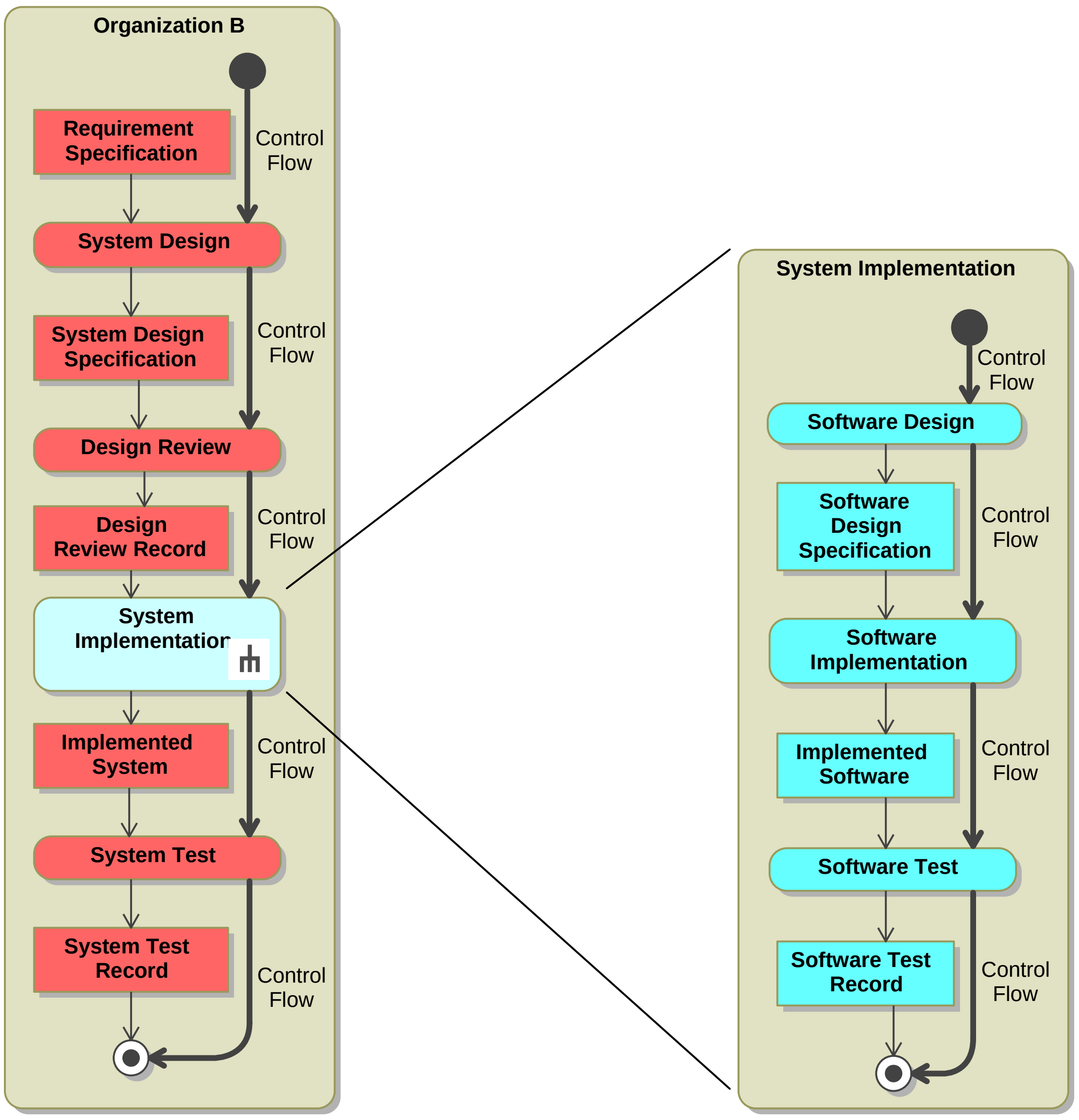


Figure 100: Hierarchical Standard Process of Organization B

Organization C (Supplier)

As illustrated in Figure 101, Organization C has a partially-defined process in place. This means that not the entire development process is documented, but only a small portion of it. In this case, it is the software development part, i.e., *Design Review*, *Software Implementation* and *Software Test* with corresponding artifacts. This often occurs in relative immature companies, which focus their process definition on core competencies.

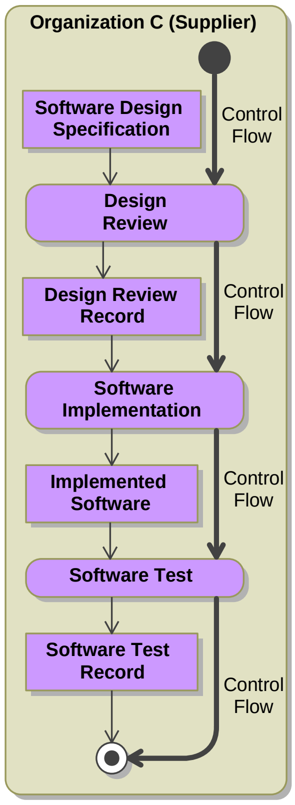


Figure 101: Partially Defined Process of Organization C

Organization D (Supplier)

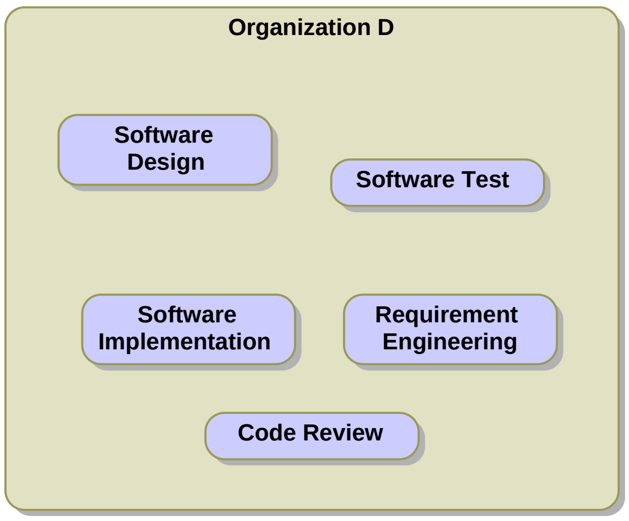


Figure 102: Amorphous Process of Organization D

Organization D has no structured process established. As shown in Figure 102, the process is amorphous, which means that there are some activities, which are conducted during development; however, these activities do not depend logically on each other. Dependencies of in- and outputs are not clearly defined. Roles are also not explicitly documented.

Organization E (Supplier)
Organization E is the major hardware supplier for Organization A (Master), especially for the **Driver Supporting System**. The hardware process is straightforward and starts with *Hardware Requirement Specification* assuming that hardware requirement engineering is already previously done. This is basis for the *Hardware Design*. In contrast to software design, this process step has to create/conduct, e.g.,:

- design/engineering drawings
- reviews using Failure Mode Effect Analysis (FMEA)
- design patterns for development
- bill of material (BOM)
- electric circuit plan
- concept for assembly
- release of assembly parts

These results are input for *Hardware Implementation* basically consisting of manufacturing of the desired hardware lot sizes. *Hardware Test* checks the proper functionality of the hardware, which is documented in *Hardware Test Record*. This is an essential input for integration activities and system test.

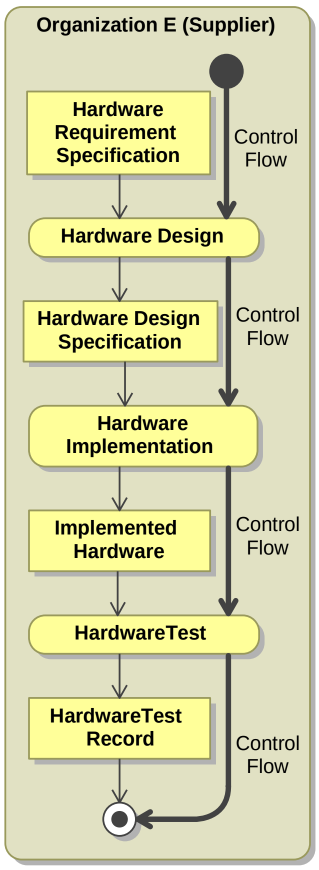


Figure 103: Hardware Process of Organization E

5.2.5 Collaborative Process Definition

As stated above, several different systems have to be developed to realize the scenario from the automobile industry. Since different collaborative partners are necessary for different products, it is also crucial to have different sub-process definitions combined for collaboration. For this reason, the following section describes the derivation of the collaborative process and the collaboration scenario itself based on (sub-) systems or products respectively to be developed.

5.2.5.1 Driver Supporting System

The Driver Supporting System functions as a kind of controller of a car's system conducting system checks and reporting the results - in the worst case errors - to the driver. Since Organization B has special knowledge on such systems, it is supposed that Organization A (Master) wants to collaborate more closely with this company than with the others for developing that system. Given the initial processes in Figure 99 and Figure 100, several mediators are used to set up the collaboration process. The entire scenario is depicted in Figure 104 and consists of several collaborative parts that are in detail described in the following.

Remarks:

Synchronization and handover points are numbered serially for better ability to reference. The additional swim lane diagrams on the right side of each collaborative scenario (black box) show which mediator has been used for process set up.

Starting with requirement engineering, the Master and Organization B decide to conduct this sub-process conjointly since this is an essential part in early development phases. Following the integration approach in chapter 4.2.5, the mediator for 'Merging Integration' needs to be applied, because this pattern addresses conjoint integration best. This mediator is again shown in Figure 105. This means for the collaborative process that after *Market Information* as the identified starting point for integration, *Synchronization-1* has to take place, which creates the same understanding of the available information from the market for both organizations. This information is the basis for the conjointly conducted *Requirement Engineering* resulting in *Requirement Specification* as the corresponding artifact. Furthermore, *Handover-1* turns this *Requirement Specification* into a format the Master can use for *System Architecture and Design*. This activity ends up with the *System Architecture & Design Specification*.

The Master decides to have this specification reviewed by Organization B, which is done by using 'Horizontal Integration' scenario (Figure 106). This makes sense since, on one hand, Organization A gets a second view and opinion on it; on the other hand, Organization B is obliged anyway to implement the **Driver Supporting System** together with Organization A. Additionally, Organization B has a *Design Review* (Figure 100) defined in its process, from which Organization A can take advantage. The review ensures that the Master's development partner gets in touch very closely with the content to be developed. Referring to Figure 104, *Handover-2* converts *System Architecture and Design* to Organization B's format. After review, Handover-3 turns the finalized Design Review Record back to the Master.

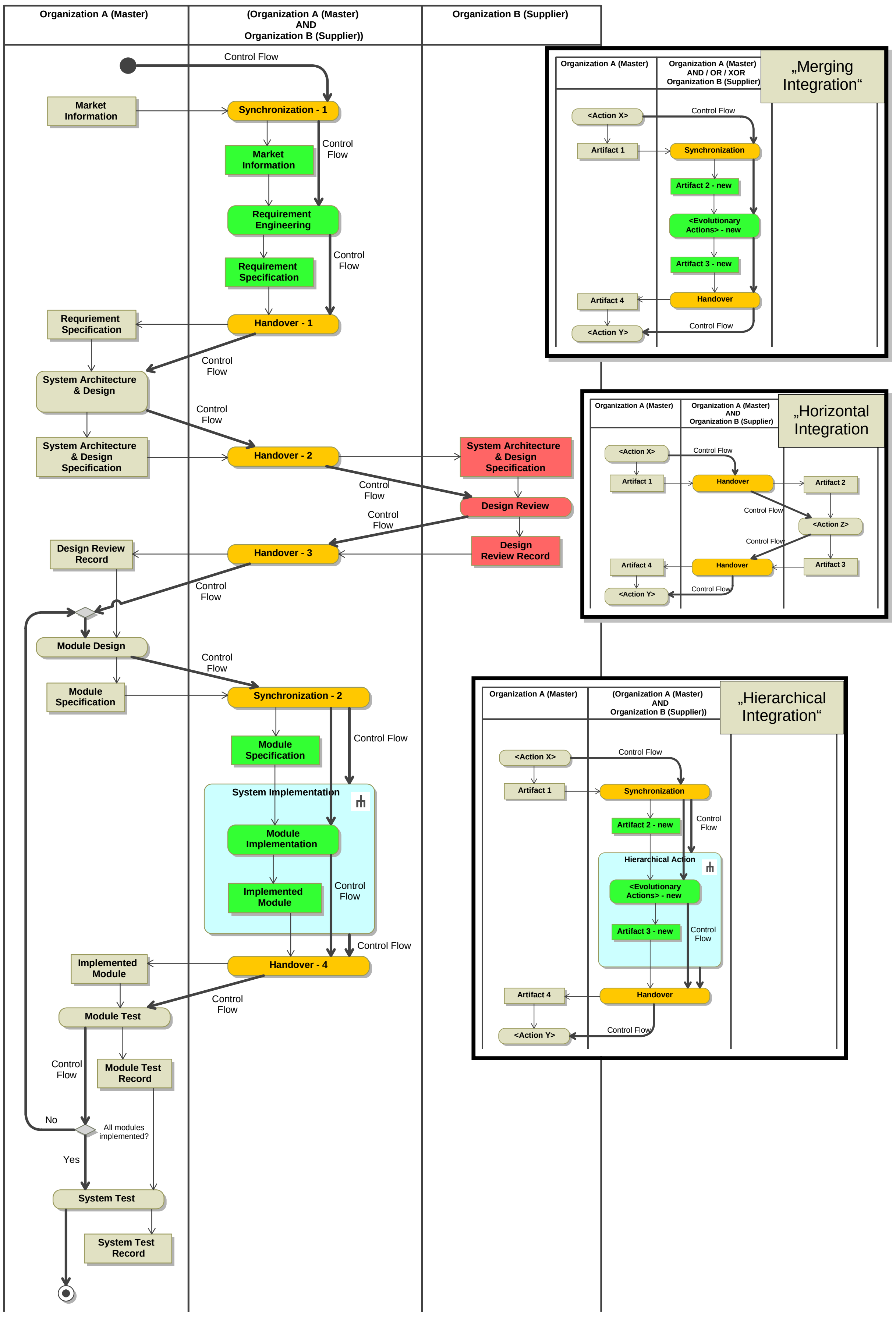


Figure 104: Collaborative Process for Development of "Driver Supporting System"

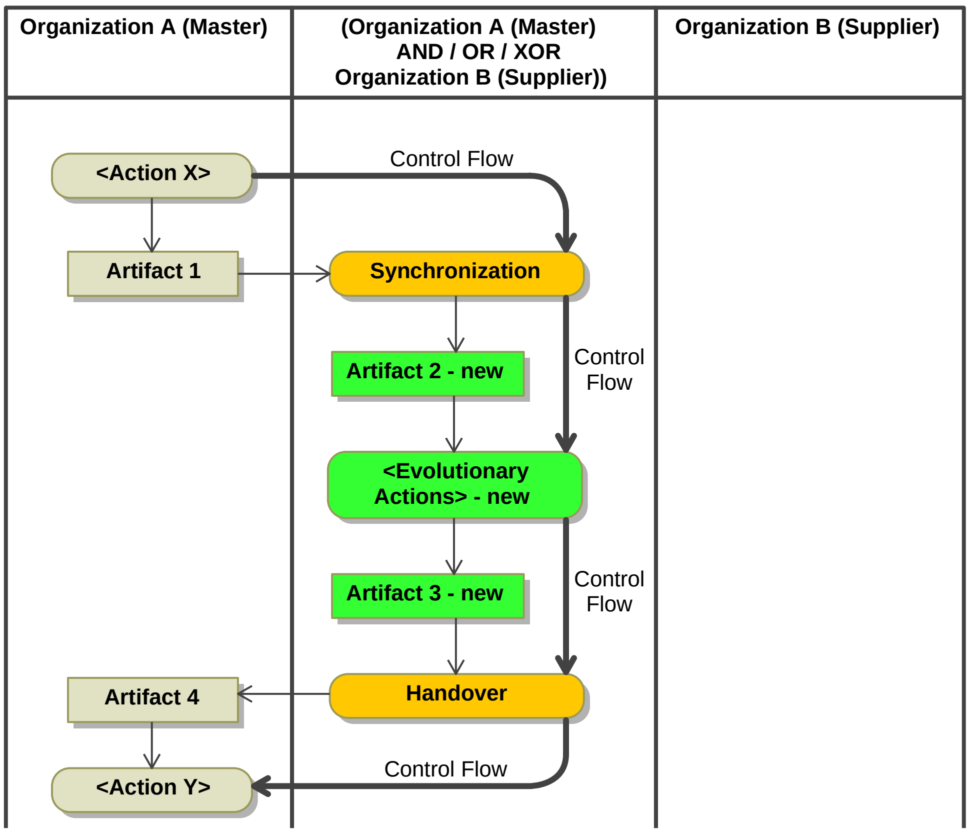


Figure 105: Mediator for 'Merging Integration'

The review record is in conjunction with *System Architecture & Design* input for the module implementation loop. The implementation loop starts with the Master that designs a module (*Module Design*). As already mentioned, Organization A decided to do module implementation together with Organization B. Since Organization B has its software implementation processes hierarchically defined, it is necessary to use a mediator that supports 'Hierarchical Integration' in a merging scenario. This mediator is depicted in Figure 107. Therefore, a *Synchronization-2* needs to be added to achieve a common understanding of *Module Specification* by both Organizations A and B. Figure 104 shows *Module Implementation* in "hybrid-view". The *Module Implementation* itself is not the one from Organization A as the labeling might suggest. Referring to the color code *Module Implementation* and the resulting *Implemented Modules* are newly defined process elements ("evolutionary" activity and artifact) based on *Module Implementation* from Organization A (Figure 99) and *Software Implementation* from Organization B (Figure 100).

After implementation of the module, *Handover-4* turns the results into the artifact *Implemented Module* on the Master's side. This is necessary to have the adequate input for the Module Test available; however, the adaptation/discussion will not be very comprehensive, if there is any at all, since implementation has been done by both organizations anyway. This means that especially Organization A will have those items be developed in *Module Implementation*, which are necessary for *Module Test*. *Module Test* results in *Module Test Record*. If there is still functionality to be implemented, the module implementation loop restarts. If the desired or sufficient functionality is created, *System Test* is conducted by Organization A. After *System Test* and the resulting *System Test Record*, the process terminates.

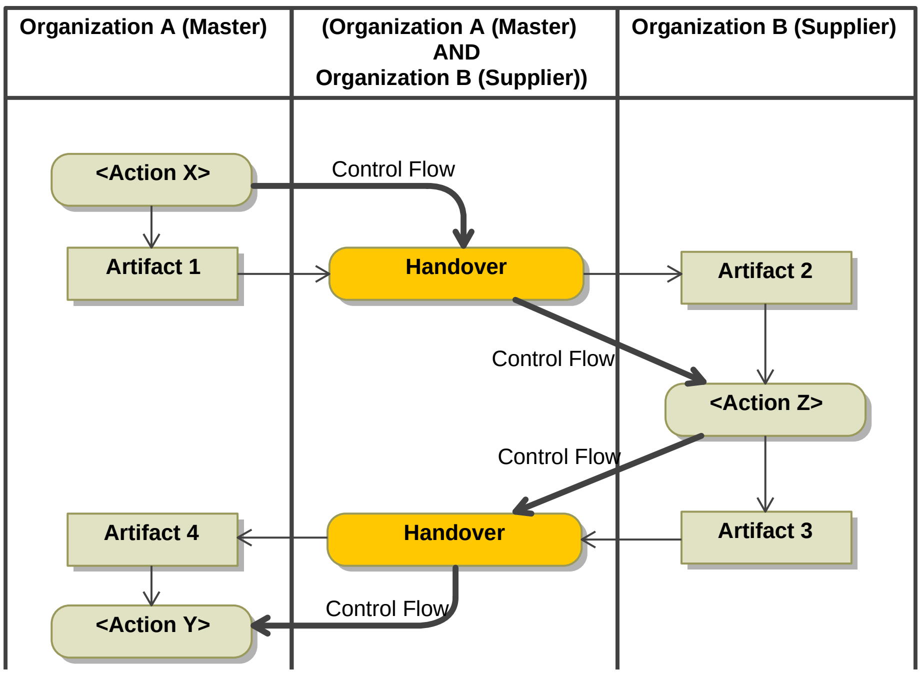


Figure 106: Mediator for 'Horizontal Integration'

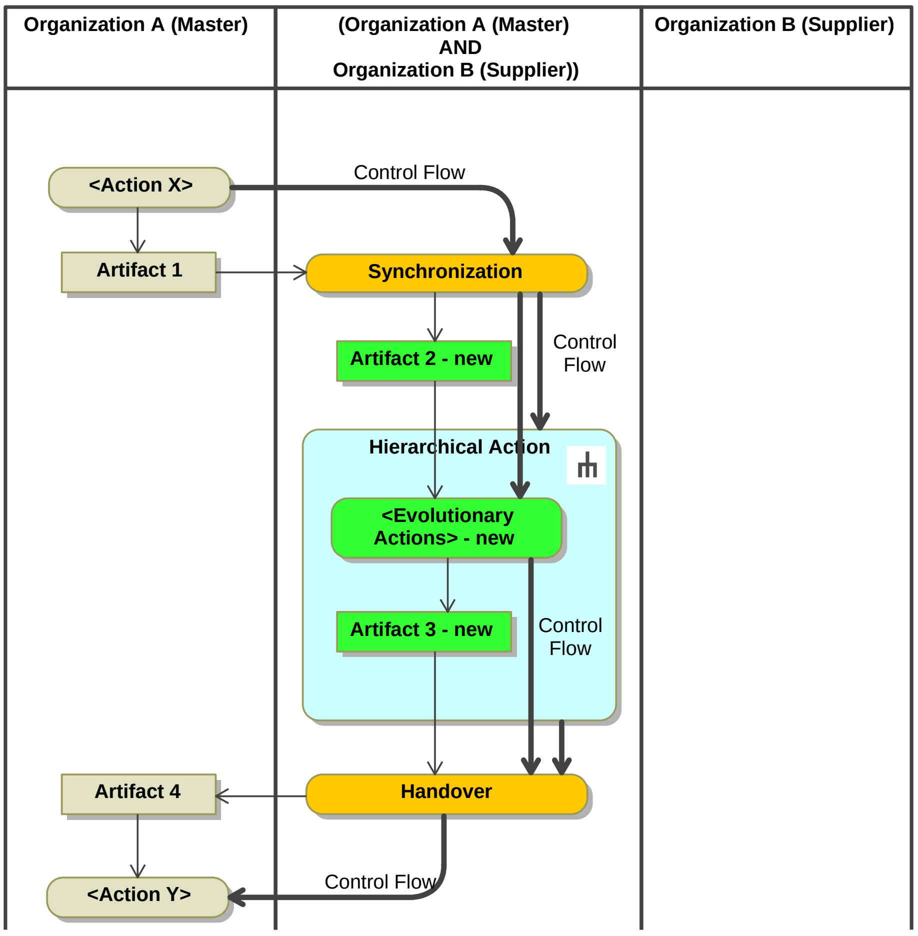


Figure 107: 'Hierarchical Integration': Mediator for 'Merging Integration'

5.2.5.2 Navigation System

The functionality of the Navigation System is rather straightforward; however, the system is a highly proprietary system, which needs to be more than others adapted to the system's interfaces in the automobile and its environment, e.g., Roadside Information System.

In this case, it is assumed that Organization A wants to outsource the design review and system development (Figure 108). For this purpose, not merely one single organization is considered for outsourcing of a sub-product's development, but two organizations, i.e., Organization C and Organization D.

Referring to the entire collaboration scenario (Figure 108) *Requirement Engineering* with the appropriate artifact (*Requirement Specification*), as well as *System Architecture and Design* and *System Architecture and Design Specification* remain at the Master's company's responsibility. For review, the *System Architecture and Design Specification* is handed over (*Handover-7*) to the collaboration partners Organization C and D. This makes sense, since these partners are foreseen to implement the system in the following. Process integration is done via the Mediator 'Horizontal Integration' as illustrated in Figure 106. The *Handover-7* - conducted by all three organizations A, B, and D - turns the *System Architecture and Design Specification* into that format Organization C and D can handle in the *Design Review*. Based on *Design Review* of Organization C (Figure 101), this activity is newly defined, since the two Organizations C and D are conducting this review conjointly. The *Design Review Record* is converted back to the Master's format by the *Handover-8* action. The reviewed *System Architecture and Design Specification* is now basis for *Module Design*, which results again in the *Module Specification*. Development of Module Specifications will be done by Organizations C and D. Organizations D, which has an amorphous process in place, is coerced to follow the development process of Organization C. This is done by using the Mediator for 'Merging Integration' from Figure 105. A *Module Specification* is converted with *Handover-9* into a *Software Design Specification*. Consecutively, Organization C and D start *Software Implementation* of appropriate software modules. The *Implemented Software* is tested (*Software Test*) and the *Software Test Record* is converted back to the format of the *Module Test Record* of Organization A. This is done by using the action *Handover-10*. Since this is an iterative process, it runs as long as there are still modules to be implemented. If all modules are implemented, *System Test* starts, which is documented in the *System Test Record*. This terminates the process.

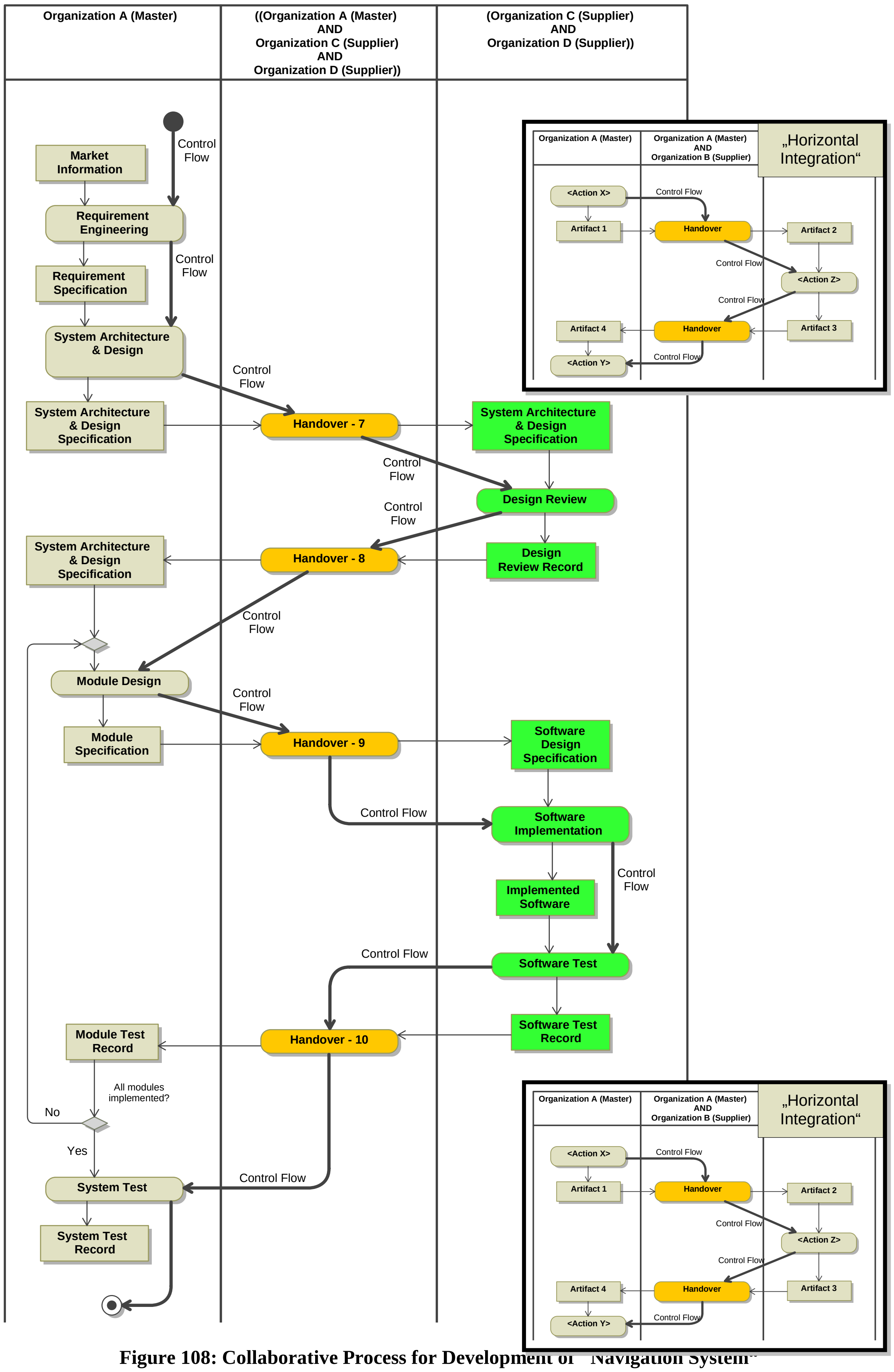


Figure 108: Collaborative Process for Development of „Navigation System“

5.2.5.3 Roadside Information System

The **Roadside Information System** that also includes software for running the **Road Condition Analysis** is constantly installed at the roadside and includes sensors for measuring weather and analyzing derived road conditions. For this reason, the development of an appropriate product requires not only software but also hardware that needs to be manufactured and programmed with parameters to be compatible with the **Driver Supporting System**.

The collaboration scenario for developing the Roadside Information System is depicted in Figure 109.

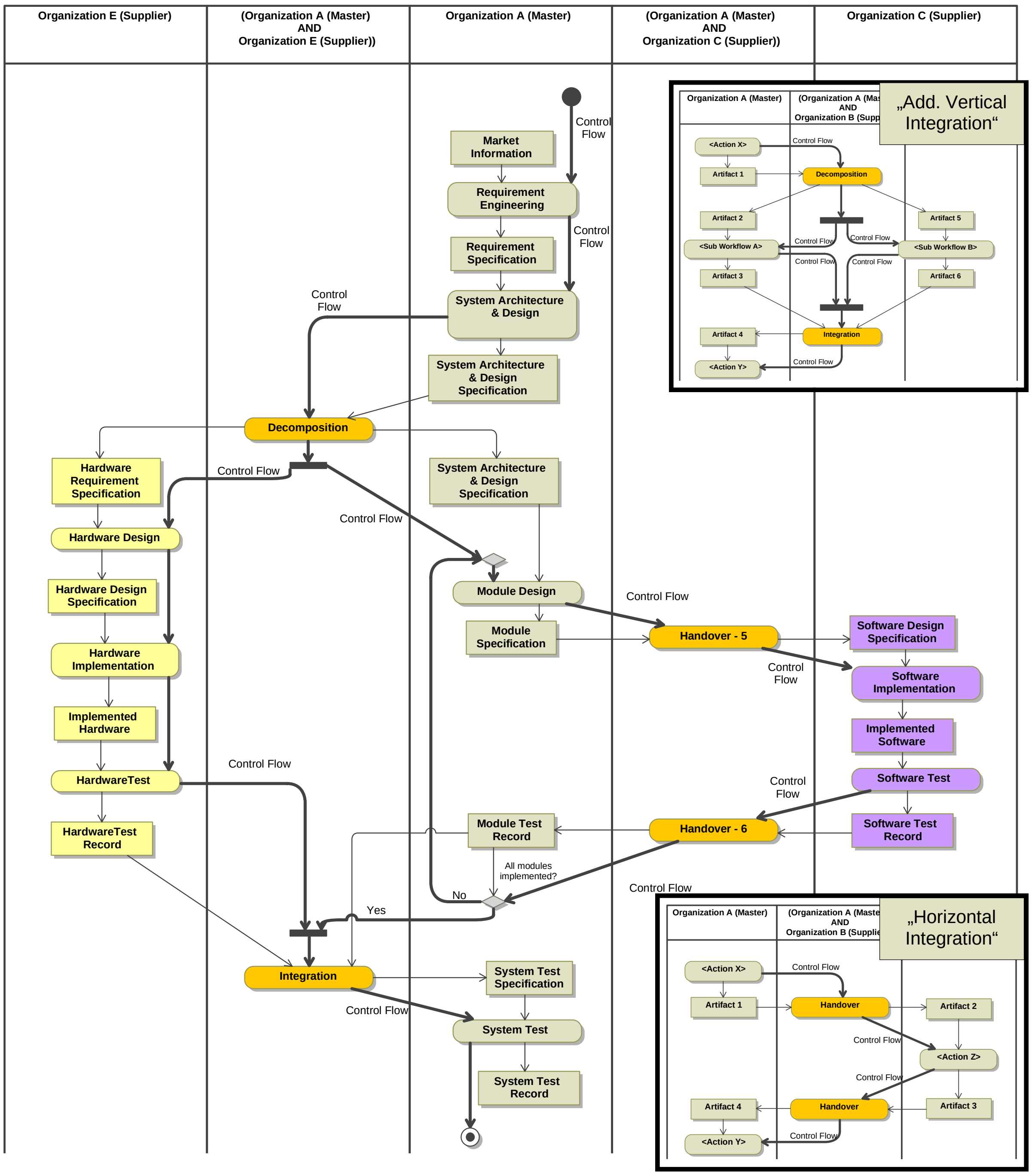


Figure 109: Collaborative Process for Development of Road Information System

For this product, Organization A decides to do the "early phase" of development by itself, i.e., *Market Information*, *Requirement Engineering* and *Requirement Specification*, as well as the *System Architecture* and the corresponding specification is all done by the Master organization.

The *System Architecture Specification* needs to be decomposed to identify those product parts that need to be allocated to hardware implementation and software implementation. Therefore, *Decomposition* shall produce a *Hardware Requirement Specification* and an appropriate *System Architecture Specification* that highlights those product parts to be implemented by the software process (Figure 109). For the hardware part, it is assumed that Organization E, as a hardware specialist, is nominated to deliver all the sensors and controllers for the **Roadside Information System**. This organization runs its process as described above at Figure 103. For this parallelizing process operation, the mediator 'Additive Vertical Integration' as shown in Figure 110 is used.

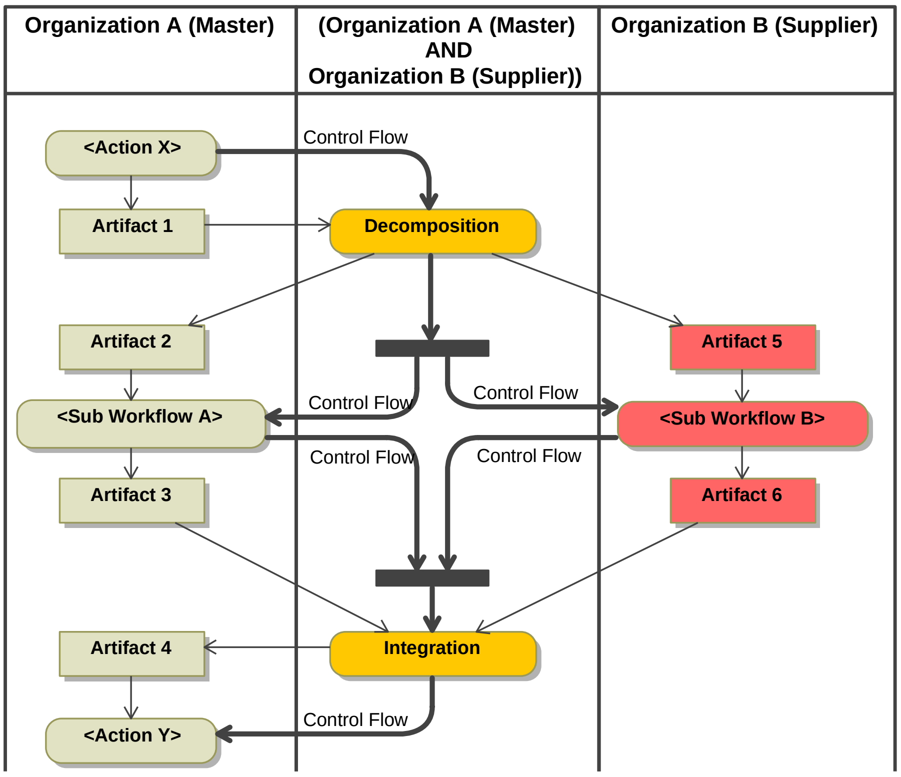


Figure 110: Mediator Pattern for 'Additive Vertical Integration'

As also stated before the entire module development of Organization A consists of an iterative process. As in the other scenarios, the Master keeps the *Module Design* and *Module Specification* in its own responsibility (Figure 109). For developing the **Roadside Information System,** Organization A takes advantage of another software supplier, which is Organization C. This concerns especially the software development. For process integration of the organization, the Mediator 'Horizontal Integration' (Figure 106) is used. This means that *Handover-5* turns the *Module Specification* of Organization A into the format (for *Software Specification*) required by Organization C. After implementing the desired software module the *Software Test Record* from *Software Test* (Organization C) is handed back to the Master (*Handover-6*), which is converted again to a *Module Test Record*. If all desired modules have been implemented, an integration of software and hardware has to take place.

This is done during *Integration*, which results in a *System Test Specification*. These process steps contain, e.g., creation and release of interface specifications, release of technical documentation, definition of packaging (if necessary), creation of a hardware prototype, set up of basic data for assembly part disposition (if necessary) etc. After *System Test*, which is documented in *System Test Record*, the process terminates.

5.2.6 Summary Case Study 1: Scenario from Automobile Industry

This scenario from automobile industry illustrates how the process integration approach works in general and how mediator patterns can be applied in order to set up a collaborative process environment for distributed development. Having defined an initial situation, it has been shown that the process integration approach is able to be used even in more complex scenarios like in Figure 109. Thereby, the set up procedure follows the content that needs to be developed by other organizations from the master organization's point of view. Whenever these contents are defined, the connection points for the mediators are identified and (sub-) processes are connected which each other. This routine is done sequentially and, therefore, points out the convenience of the approach's applicability even in complex scenarios, i.e., more than two involved organizations and more than one mediator usage for collaboration process set up.

Two case study questions that have been raised in section 5.1 are answered and reflected here:

1. How does a (new) collaborative process (control flow) look like that incorporates two or more different processes from cooperating organizations?

   This case study gives an impression that the process integration approach works within an application case. It has been shown that processes might be dependent from product to be developed since

   a) the content and requirements are different for every product and
   b) the participating organizations are also not the same for each product.

   Furthermore, it is illustrated how to integrate organizational processes that are not fully defined keeping in mind that at least some process fragments need to be available to make the process integration approach work.

   Last, the mediator functionality is shown by using several mediators in a row for each collaboration scenario provided (Figure 104, Figure 108, Figure 109). For better illustration, these UML activity diagrams depict which mediator has been used for setting up the collaborative scenario.

2. How do artifacts (data flow) look like? How can artifacts be handled if they are defined in different formats?

   Data flow is also depicted in the provided collaborative scenarios and goes very often hand in hand with control flow. If the mediator is properly defined, it is also very easy to use the mediator in whatsoever complex scenario. This should not simplify the problem of collaborative development processes, but underlines the ease of use of the process integration approach in general.

   The problem of having several and different artifact formats in each organization defined is handled by each mediator used and has not been explicitly addressed in this illustrative case study. The possible handling of this problem is described in the artifact and handover concept in Chapter 3.3.2.

5.3 Case Study 2: Scenario from Agile Development

This case study is basically defined as an “Exploratory Case Study” that addresses and measures the performance of the process integration approach. Results shown in the following have been published in [16].

The scenario investigates distributed agile software development [14] in a students' project, which was held as a joint course by the Technische Universität Clausthal (TUC) and the Leibniz Universität Hannover (LUH). A similar, non-distributed course was held at LUH many times before. One of the main goals in the non-distributed agile course was to teach eXtreme Programming (XP) in a realistic environment. The students should learn how the agile practices affect software development. Therefore, all twelve practices proposed by Beck [14] were tried to be fully implemented. An essential part of the course is a one- week block, encompassing most of the software development activities. Thus, it is possible to implement practices like Onsite-Customer, 40h - week, and Pair-Programming. For teaching purposes, iterations were kept very short, i.e., two-day iterations worked best [135].

5.3.1 Case Study Questions

This case study targets the performance assessment of a distributed process in comparison to a process conducted in one location. This distributed process has been set up with the approach of this thesis.

The assessment is based on the “Goal-Question-Metric” (GQM) paradigm, which helps to assess process improvements or at least changes in a defined and systematic way [10]. This approach deals with a specific type of improvement:

a) the improved process shall be applicable in distributed projects and
b) the improved process shall still incorporate the same advantages and properties of the original process.

In such comparable co-located projects, some advantages of the corresponding co-located XP process have been observed, which are also worthy to be included in a distributed development process. Therefore, the *Goal* of this case study is to evaluate within a distributed development project:

1. high developer commitment because of shared responsibility.
2. low risk of failure and low amount of rework because of good customer interaction.

Depending on the abstractness of the defined GQM goal, a refinement into sub goals might be required. Based on experience, goals should have a single quality goal and a single well-defined perspective for best results.

It needs to be investigated to what extent the advantages from co-located development could be retained. Such a comparison is specific for a given project situation and the type of processes involved, since we want to assess whether a specific advantage exists in the derived process.

The following research questions have been systematically derived, which will be evaluated in Chapter 5.3.4:

Question 1:
Does the derived distributed process explicitly allocate responsibilities in a way that leads to comparable commitment of developers as in the co-located case?

Question 2:
Does the continuous integration and the handover of new functionality to the rest of the team work as well in reducing the risk of failure as in the co-located process?

Question 3:
Does the distributed process support the development of a shared understanding of the design?

Question 4:
Despite the additional communication effort of the distributed process: is it still profitable to execute projects with this process?

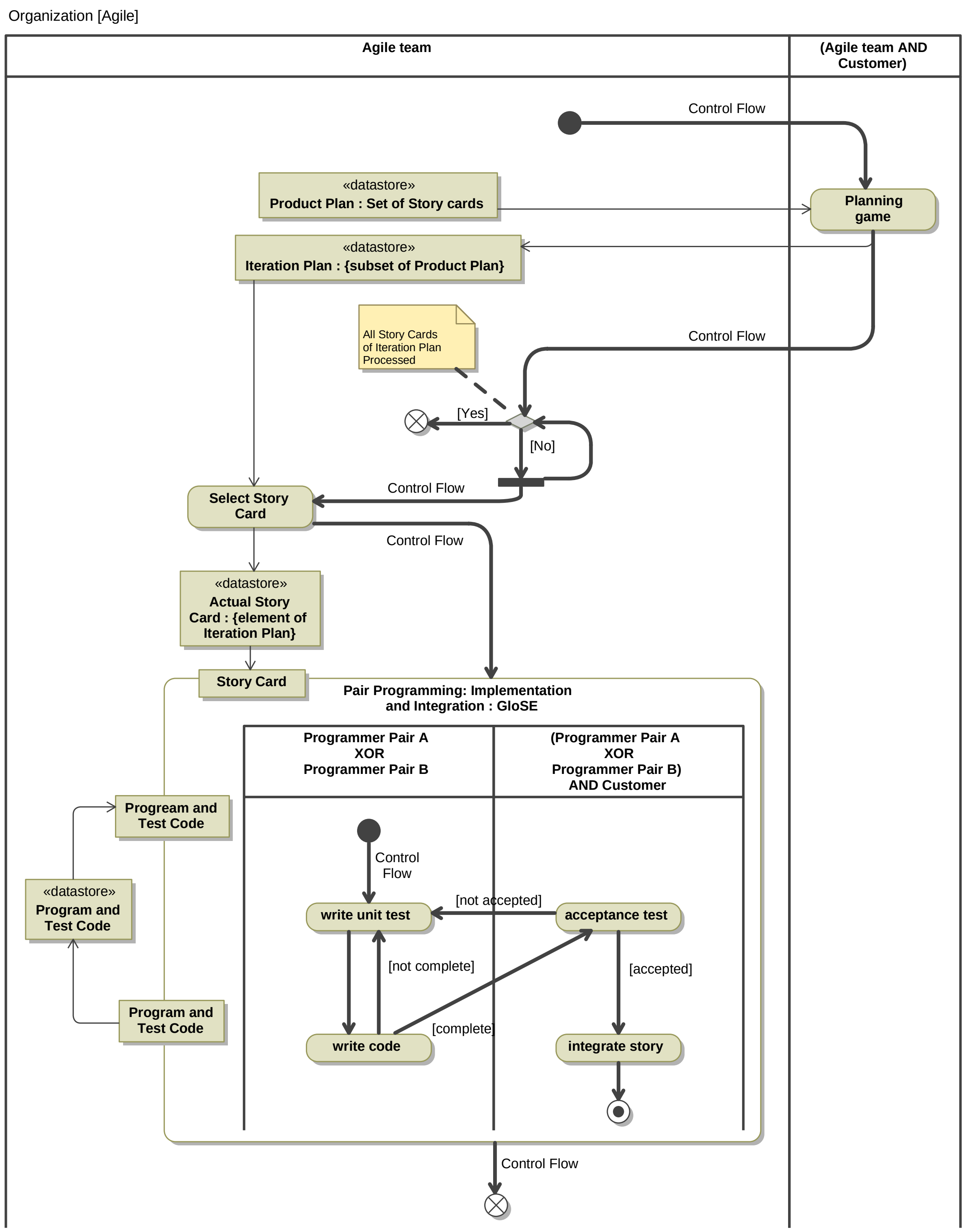
5.3.2 Organization's Process Definition

Figure 111 describes the development process in co-located XP classes, illustrating the initial process of LUH. This process is basically an agile process incorporating the major concept of story cards. These handwritten slips of paper contain a short description of how a future user will use the system that is still under development. Often, narratives (i.e., usage stories) are used. Onsite customers write these stories together with the agile team. New story cards can always be created and added to the Product Plan. The Product Plan contains all story cards the team and customer are aware. Referring to experience, most stories are created at the beginning or end of iterations.

Each iteration starts with a Planning Game. If the customer comes up with new stories, these are considered first. After that, each story card is estimated by the developers for its forecast effort. Then, the customer prioritizes the story cards by sorting them. The most important story card lies on top of the stack. Based on experience from past iterations, customer and team select as many story cards from this stack as one iteration allows to implement, which defines the iteration plan.

After the Planning Game, which is conducted in conjunction with the customer story cards from the iteration plan are implemented via Pair Programming. Any pair of two developers takes the topmost story card ("*Select Story Card*") and implements it by applying the test-first practice. The developers start with writing an automatic unit test ("*write unit test*"), then add just enough code to make the test run ("*write code*"), before writing the next test. This is depicted in the activity "*Pair Programming: Implementation and Integration*". As illustrated with turning arrows, improvement loops are included.

At any time, the developer pair thinks that a story card is implemented, they go to the onsite customer and present their results. If these results are acceptable for the customer ("*acceptance test*"), the developer pair integrates them into the system, and the story card is finished ("*integrate story*"). The pair takes the next story from the iteration plan and starts again. This iterative process loop ends if all story cards of iteration plan are processed and/or acceptance from the customer is gained.



Organization [Agile]
Agile team
(Agile team AND Customer)
Control Flow
«datastore» Product Plan : Set of Story cards
Planning game
«datastore» Iteration Plan : {subset of Product Plan}
All Story Cards of Iteration Plan Processed
Control Flow
[Yes]
[No]
Select Story Card
Control Flow
Control Flow
«datastore» Actual Story Card : {element of Iteration Plan}
Story Card
Pair Programming: Implementation and Integration : GloSE
Programmer Pair A XOR Programmer Pair B
(Programmer Pair A XOR Programmer Pair B) AND Customer
Program and Test Code
«datastore» Program and Test Code
Program and Test Code
Control Flow
write unit test
[not accepted]
acceptance test
[not complete]
[accepted]
[complete]
write code
integrate story
Control Flow

Figure 111: Co-located Agile Development Process

Prior to participation of the joint course, TUC followed a traditional waterfall development process depicted in Figure 112. This process was not able any more to address the up-to-date development issues, e.g., fast changing requirements, early releases for customers etc. Therefore, an agile approach like at LUH was intended for use in development projects. However, this approach did not consider an explicit design, which has been seen as very valuable at TUC, especially for distributed development projects. This made TUC consider an integration of the old waterfall design into a new development approach.

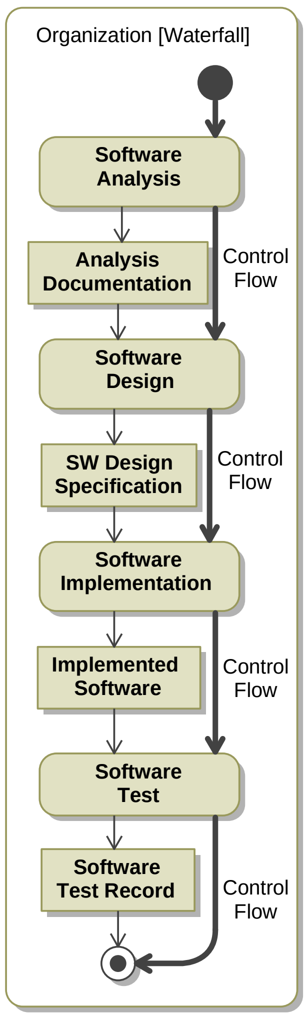


Figure 112: Initial Waterfall Process

5.3.3 Collaborative Process Definition

Referring to the collaboration scenarios defined in Chapter 4.2, the scenario ‘Additive Vertical Integration’ is taken into consideration to connect processes from LUH and TUC, based on agile development practices. The appropriate mediator for this scenario is again depicted in Figure 113.

In the following, it is illustrated how to apply the process integration approach to a students’ project having 11 participants at LUH and 4 participants at TUC. First, two teams are formed: a local team with 7 students from LUH who followed the original co-located process and a distributed team with 4 students from each site (two programmer-pairs per site) who followed the derived process for distributed XP. There was no barrier due to languages, time zones, and resulting communication problems, because LUH and TUC are both German universities. The project conducts 14 development runs with duration of 4 hours each. Thereby, five days included two development runs each and four days with one development run. The entire development project was based on XP. Although the project lasted only a little over two weeks (14 working days), all XP practices were applied, e.g., Planning Game, Refactoring, Pair Programming, Onsite Customer, Continuous Integration etc.

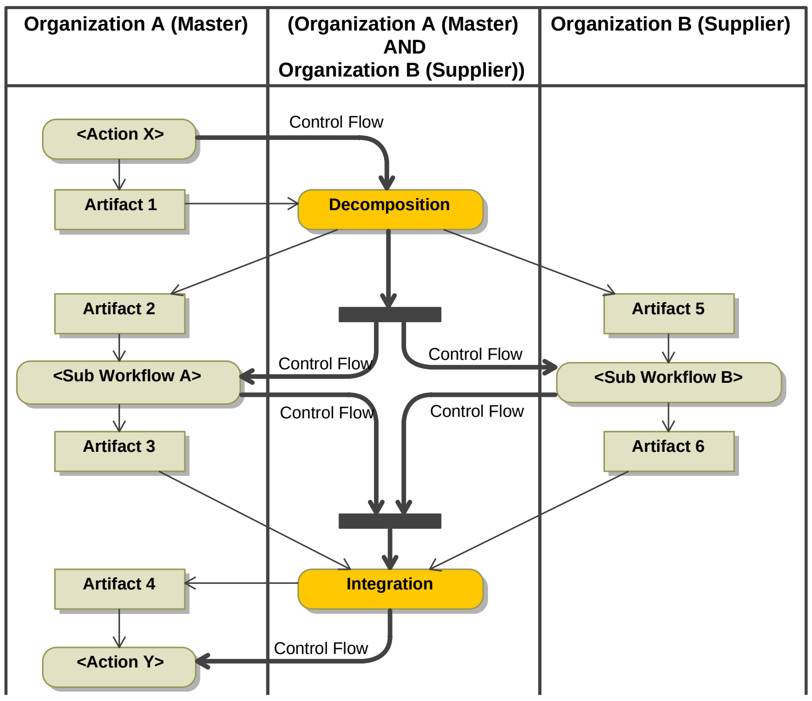


Figure 113: Mediator Pattern for 'Additive Vertical Integration'

Besides the XP practices discussed in Kircher et al. [79], special attention needs to be paid on the design activity, which has been done implicitly in the co-located agile scenario (Figure 111). This implicit approach is acceptable, since communication paths are very short and decisions can be taken informally without having anybody of the agile team ignored. In a distributed development environment, design discussions are even more important due to the distributed project character - even a simple design [14] needs to be known at all sites. Consequently, the main intention of TUC was to keep and conduct explicit design activities, even though an agile approach is followed. As depicted in Figure 111 programmer pairs continuously take story cards to implement. Design discussions and definitions have to be now explicitly defined and conducted in parallel to the implementation of story cards. This is necessary, because various features of story cards might affect the system design. The design adoptions in a distributed project environment need to follow a traceable approach in order to get commitment from the entire team.

Due to the fact that development activities have to be explicitly conducted in parallel, the 'Additive Vertical Integration' scenario is predestined to be applied. Figure 114 illustrates the collaborative scenario. Following this mediator pattern as illustrated in Figure 113, we have to identify the respective <Action X> first, that is the connection point for processes to be integrated.

Having chosen the action *Select Story Card*, a newly- defined action *Decomposition* has to be added right after a story card is selected. In this action, the agile teams decide on the dedicated developer pairs, which start discussions about design. A pair itself is not distributed over two locations, since this would make the XP practices even harder to fulfill by having too much communication effort. For this reason, a XOR role model definition is applied in the *Pair Programming* activity, which minimizes communication overhead. Going further in

Figure 114, the fork node starts parallelizing the process. The left path covers the design work, which typically leads to a design sketch on a whiteboard and documents the most important design decisions for the current iteration. The design sketch will be removed, if it is no longer useful.

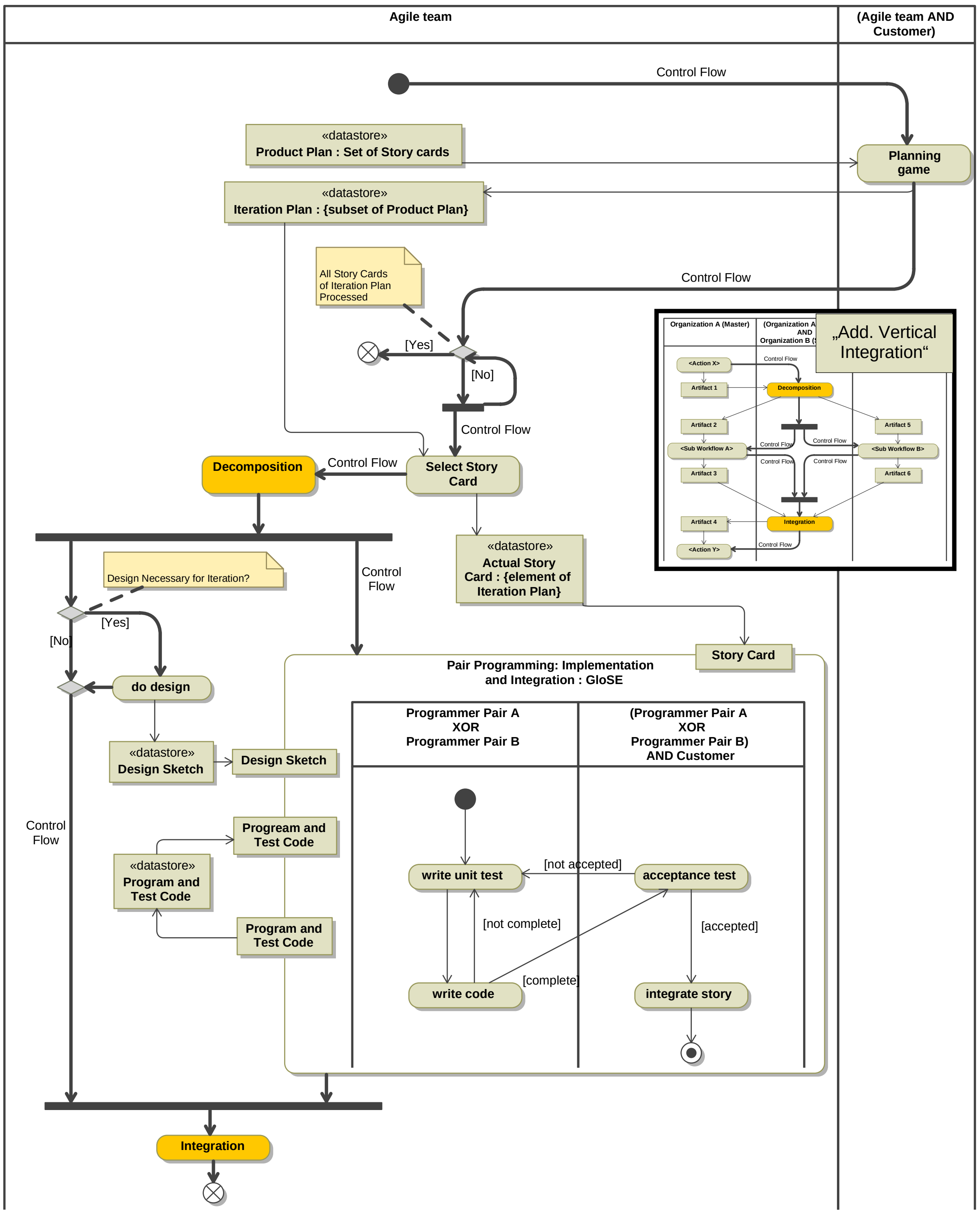


Figure 114: Distributed Collaborative Process

The right path leads directly to activity *Pair Programming: Implementation and Integration*. As depicted in Figure 114, a useful design sketch is input for this activity, which describes the implementation process itself. After implementation the story card's code is integrated into the existing software product. Following the scenario in Figure 114, the join node terminates the parallelism. At this point another overall integration needs to take place, which also includes the integration of various design approaches created on the whiteboard. This is represented by an additional *Integration* action at the end.

5.3.4 Assessment of the Derived Process

After having a process derived for distributed development, this section assesses whether it works as intended.

The GQM paradigm demands for a systematic approach for evaluation, i.e., the investigation goal has to be defined. Basili suggests a specific template to document such goals and to refine them to testable sub goals [10]:

Analyze:
A precise definition of the object under investigation should be given. In this case, it is the (formerly co-located) process that should be applied in distributed software development.

Purpose:
Basili presents a set of three pre-defined GQM purposes: understanding, improving, and controlling. Here, the purpose is to improve the process at hand in order to apply it in distributed software development.

Quality Focus:
High level description of the quality goals should be given. These goals are the basis for measurement. In this approach, these quality goals are the specific advantages and properties associated with a given co-located process.

Perspective:
Basili emphasizes the importance of the perspective from which a process is observed. In this case, it makes a difference, if a quality goal is observed from the perspective of a developer, a project manager, a customer, etc.

Context:
Basili's template asks for the context, where the investigation shall take place. In this case study, it is the (first) application of the derived process in a distributed project. Results of the evaluation are compared to experiences with comparable co-located projects (the baseline).

Following the GQM paradigm, *Questions* need to be derived from *Goals* defined in chapter 5.3.1. In GQM, this is typically done with Abstraction Sheets (Table 11), which is introduced in the following.

The **Quality Focus** from the goal is refined to Quality Aspects. This case focuses on *Communication Effort*, *Developer Commitment*, *Risk of Failure*, and *Amount of Rework*. These are the most important properties of the co-located process, which need to be investigate to what extent they are incorporated in the distributed process.

Next, the **Baseline Hypotheses** for each Quality Aspect need to be defined. These are based on experiences from a co-located environment. The assumption shows, which property of the collocated process is responsible for a respective quality aspect. These properties are documented as **Variation Factors**.

The main difference of this approach to the standard way of using Abstraction Sheets is the specification of important aspects this distributed process should have due to the Variation Factors. This is done in the lower right field of Table 11.

Table 11: Abstraction Sheet

Quality Focus	Variation Factors
▪ Commitment of developers ▪ Risk of failure ▪ Amount of Rework ▪ High Truck Factor ▪ Communication effort	▪ Shared responsibilities (e.g. Collective Codeownership) ▪ Integration and Handover after finishing Story Cards ▪ Synchronization of Design Sketches and Rationales ▪ Jointly Execution of Planning Game and Pair Programming ▪ All Variation Factors significantly add to the communication effort in distributed development
Baseline Hypothesis (co-located)	**Impact of variation factors** (on distributed process)
▪ Commitment of developers: high because of shared responsibilities ▪ Risk of failure: low (ca. 25%) because of continuous integration ▪ Amount of Rework is low because of daily synchronization ▪ xy% of work time is communication effort	▪ Explicit allocation and sharing of responsibilities allows comparable commitment of developers in distributed projects ▪ A strategy for integrating and handing over new functionality to the rest of the team leads to comparable low risk of failure and low amount of rework in distributed projects ▪ Synchronization of Design Sketches and Rationales from all sites decreases Rework ▪ Jointly execution of (distributed) Planning Game Activity increases the Commitment of Developers. ▪ The advantages of being able to execute the process in distributed projects justify 10 percentage points more communication effort

The Abstraction Sheet allows deriving questions for evaluation systematically. For each desired property of the distributed process we need to evaluate, whether the derived distributed process satisfactorily fulfills the associated quality aspects. As proposed by GQM, one hypothesis is given for each question, which allows evaluating whether a Quality Aspect is satisfactorily met.

5.3.4.1 Question 1: Commitment of Developers

Question 1:
Does the derived distributed process explicitly allocate responsibilities in a way that leads to comparable commitment of developers as in the co-located case?

Metrics:
The metric measures the additional time students invest into the course (M1: overtime [h]) as an indicator they are highly motivated by the project. For the same reason, it is counted how often students are late (M2: occurrences of being late).

Hypothesis 1:

There is no difference in metrics M1- M2 between co-located and distributed teams.

Measurement:
During the project, tutors and observers logged all peculiarities, especially if (M1) students stayed for longer discussions or did some work (e.g. reading tutorials) at home and (M2) if a student was late during the block course.

Findings:
Students from the global team were very interested in process issues. On three occasions, they stayed more than half an hour longer, discussing XP concepts and general software design. Such occasions were not observed with students from the local team, but slightly more volunteers were found from the local team when volunteers were needed to create a market-ready version of the software after class. Therefore, M1 can be rated to be indifferent or even in favor of the global team. On the first few days, distributed stand-up meetings were delayed, because some developers were late (5-15 minutes). Punctuality improved during the distributed project. In the co-located project, some of the developers were regularly late, so the co-local team performed worse with respect to M2. Similar projects in past terms show that the distributed project is more typical than the co-local here.

5.3.4.2 Question 2: Risk of Failure

Question 2:
Does the continuous integration and the handover of new functionality to the rest of the team work as well in reducing the risk of failure as in the co-located process?

Metrics:
New functionality should be added at a stable pace. This reduces the risk of not finishing in time with the most important requirements implemented. A stable pace is only possible, if new functionality can be integrated without major problems and reuses functionalities of existing increments. Therefore, variation of the implementation progress has been measured (M3: variation of velocity).

Hypothesis 2:
There is no difference in the variation of the implementation progress (M3).

Findings:
Figure 115 shows the implementation progress (x-axis: development runs (4h length), y-axis: estimated effort) of the distributed team in comparison to two co-local XP projects (Figure 116 and Figure 117).

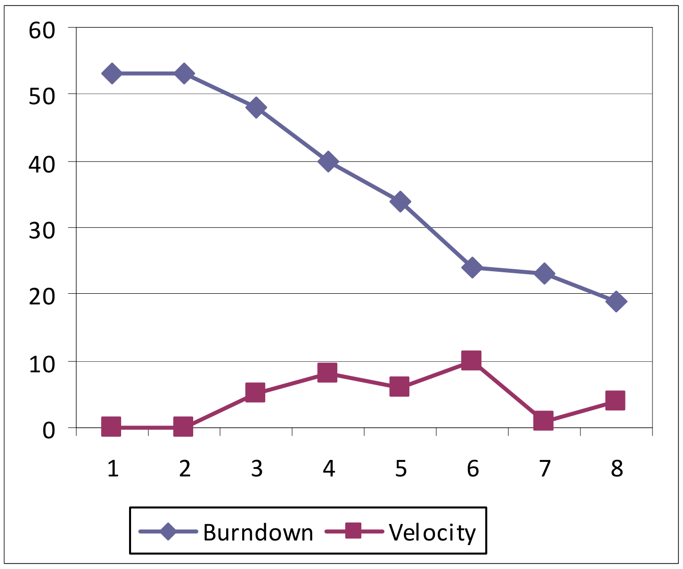


Figure 115: Velocity and Burn-down in the distributed XP project

The velocity shows how much of the estimated effort was implemented for each development run. The burn-down depicts how much of the initial amount of estimated work was left each day. This amount is reduced by the estimated effort of a story card, whenever it is integrated after acceptance. The burn-down should fall continuously, whereas, the velocity should ideally be constant.

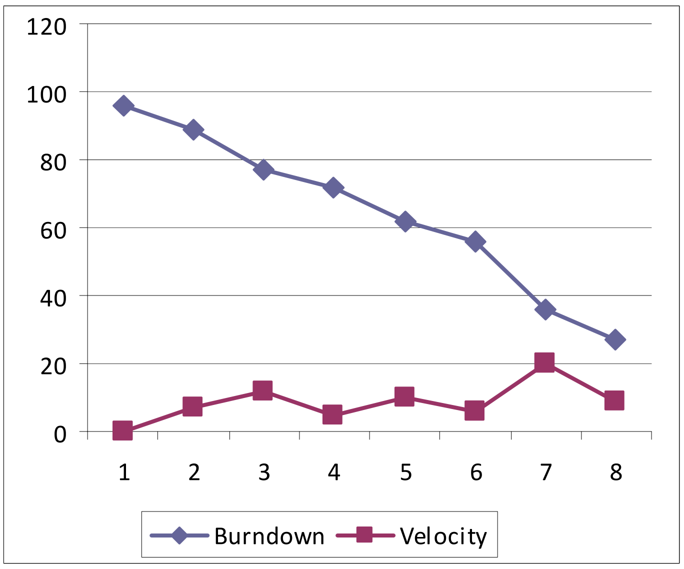


Figure 116: Velocity and Burn-down from a well conducted co-local XP project

If a project progresses well, the velocity is constant or even growing (Figure 117). If problems occur, the velocity drops (Figure 116). Compared with these two co-local projects, our distributed project seems to be fine (Figure 115).

Remarks:

For Figure 116 and Figure 117 only limited data has been available, which makes statistical comparisons difficult. Nevertheless, the major trend of Burn-down and velocity is made clear.

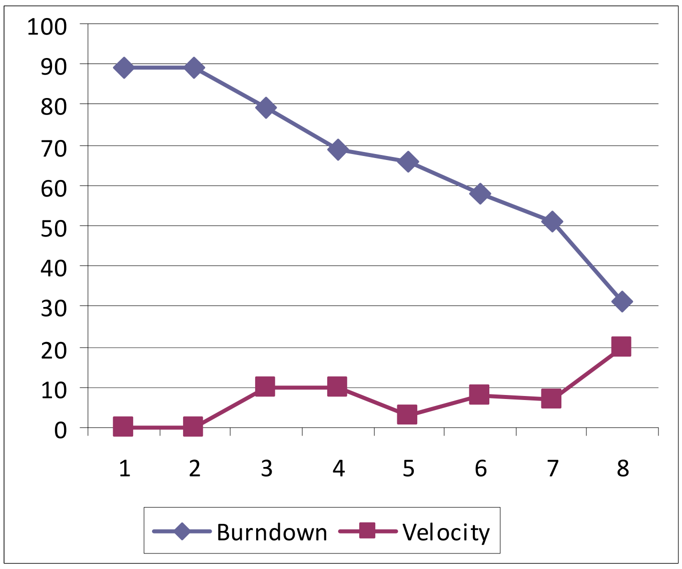


Figure 117: Velocity and Burn-down from a bad co-local XP project (x-axis: 8h day)

5.3.4.3 Question 3: Amount of Rework

Question 3:
Does the distributed process support the development of a shared understanding of the design?

Metrics:
The amount of rework was measured in two ways. Firstly, the number of story cards with bugs ("bug cards") in relation to all story cards (M4). Secondly, the time spent on these bugs (M5).

Hypothesis 3:
Basili and Boehm stated in [23] that the typical amount of rework is about 40-50% of the overall project effort. Our projects should be better than that.

Findings:
Number of bug cards shows a relative high value for M4 of about 37% bugs. However, none of these bugs took a long time to correct (M5, Figure 119): the amount of rework in terms of time [minutes] is determined at about 15%. Again, these values are comparable to the co-located projects.

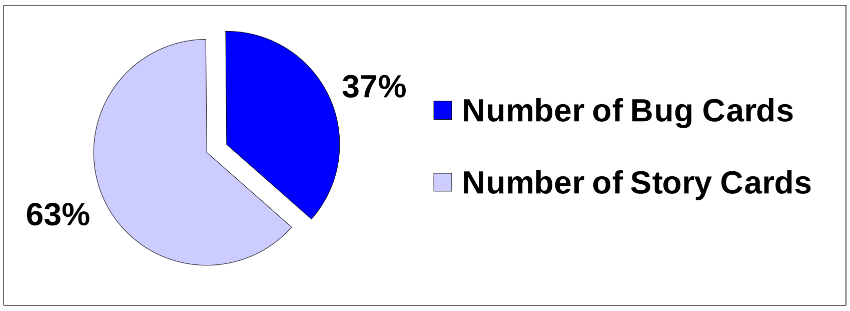


Figure 118: Number bug cards in relation to story cards

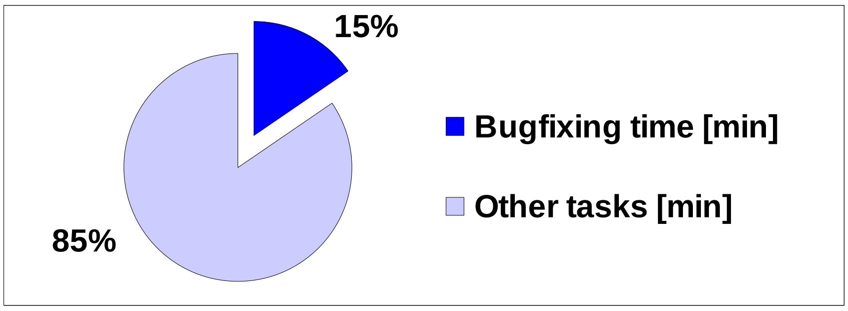


Figure 119: Relation of bug fixing time to other tasks

5.3.4.4 Question 4: Communication Effort

Question 4:
Despite the additional communication effort of the distributed process: is it still profitable to execute projects with this process?

Metrics:
Working time spent in regular communication (i.e. Planning Game, Stand-Up Meeting, Design Synchronization / Handover) by

a. the local team (M6)
b. the global team (M7).

Hypothesis 4:
Communication effort is expected to be higher in the distributed project, but it should not be more than twice as high: 1.5 * M6 < M7 < 2 * M6.

Findings:
The communication effort of this project has been investigated in detail in [96]. Accordingly, the communication effort for the daily stand-up meeting was 28:32 minutes on average per meeting for the distributed team. Additionally, three planning games were conducted lasting 1:36 hours on average (M7). In comparable co-located projects, the stand-up meetings lasted 15 minutes on average. The planning games were usually shorter, too.

The communication effort is about twice as high as in co-located projects. The problem is even more severe, if additionally set-up costs are considered [96]. As discussed in [9], strategic considerations can make this additional effort acceptable. In this case, it is more effective to stay more agile, and therefore to be able to react faster, and still distributing development between two sites. This way, both sites learn to work with each other by forcing knowledge sharing simultaneously.

5.3.5 Validity of Results

The investigation in this case study is subjected to certain threats to validity. Wohlin et al. introduced a well- accepted classification of typical threats [149]. Based on the case study 2, the validity in general is shown in the following.

Basically, two different topics are discussed:

1. First, the selected type of the case studies itself and their representativeness.

2. Second, how results of the case study support the process integration approach of this work.

Following Wholin’s classification of threats, four types of validity are differentiated.

5.3.5.1 Internal Validity

In this project students were very motivated in the XP laboratory, because the course is voluntary and very well accepted among students. Thus, results concerning the motivation of the students might be independent from dispersiveness of the project. This threat is somewhat leveraged by the relative comparison of two projects in the same laboratory. In addition, students might be especially motivated to participate in a distributed software engineering class. For this reason, the two teams have been chosen randomly. It is possible that supervisors tried to mitigate discouraging effects in the global team to avoid annoying students.

The constant velocity and the low amount of rework we observe in our data could result from

a) a good global team or
b) a very simple task with only a few requirement changes instead from the good derived process.

The randomly selected teams are very likely to have similar strengths. However, it is very difficult to evaluate team strengths in a similar assignment, since objective criteria that document the successfulness are very challenging to define.

5.3.5.2 External Validity

Threats to external validity affect the generalizability of the results.

The evaluation scenario is a student's project, i.e., it remains an open question, whether the assessment results would hold for industrial projects, too. Although this was not the goal of our evaluation, it is very certain that the process integration approach will be also feasible and successful in the industrial context: The complexity of the investigated process is comparable to industrial processes, and most of the metrics for assessment will be measured as part of the normal management of industrial projects. However, an industrial case study remains future work.

5.3.5.3 Construct Validity

In applied research, construct validity (the theoretical constructs and their representation in the experiment) is less important than internal and external validity.

The applied metrics for *Risk of Failure* and *Amount of Rework* are not fully covering these constructs. Nevertheless, it is important to get a feeling for the performance of the derived process in these fields. For evaluation of this approach, the construct validity is more important. For this reason, it has been focused on a students' project, in which the realistic mapping of the important agile practices has been already proven [135].

5.3.5.4 Conclusion Validity

Conclusion validity deals with the question, whether a repetition of the study will lead to similar results.

For this investigation, this is the least important validity class. Understanding the implications of executing a given process in a distributed setting is more important than statistical significant results. In practice, the assessment should yield good results before a large number of projects has been assessed - especially if the process does not perform as well as expected.

5.3.6 Summary Case Study 2: Scenario from Agile Development

The case study 2 applied the collaboration scenario ‘Additive Vertical Integration’ to derive a process for distributed development in a students’ XP project conducted between LUH and TUC. This development project was basically set upon an agile basis with all typically used agile methods. Agile projects are usually less process-oriented; nevertheless, it is crucial, especially for small projects, to make use of the ability to systematically derive a process for distributed development from a well understood co-local development process. The reason for this is due to the fact that more and more software is developed in distributed teams.

The performance of the process integration approach is complemented by empirical techniques to derive the benefits and drawbacks associated with the original co-local process. This is a prerequisite for assessment, whether these properties are represented in the derived distributed approach.

This example focuses on the properties *developer commitment*, *risk of failure*, *amount of rework*, and *communication effort*. Basically, the derived process performs as well as the original co-located process at the cost of twice as high communication costs. The reasons for that are manifold; however, the use of an explicit design activity might essentially contribute to a project's success, since design activities are typically the foundation of any software product.

The results provided give process managers one example of the potential successfulness of a collaborative scenario supporting the understanding of dependencies. Others may find these metrics and data points useful, when evaluating distributed projects.

Since this case study provides only one quantified example (data point) concerning the effects of the use of the ‘Additive Vertical Integration’, it cannot be concluded that each and every software development project using ‘Additive Vertical Integration’ is equally successful.

However, currently, project managers are limited to ad-hoc adjustments for making a process fit for distributed development. Even if the correctness of this data point will not be confirmed by another example case, this data point provides “First Aid” towards getting a feeling of parameter sensitivity and having more effective distributed processes in place.

6 Discussion

After demonstration of the process integration approach on a concrete scenario in Chapter 5.3, the applicability of the approach is reviewed and discussed here.

6.1 Major Issues of Process Integration Approach

6.1.1 Accuracy of Mediator Definition

A basic concept of the introduced process integration approach is the usage of mediators. These mediators connect the collaborating parties' processes together and create a new executable process as final result. Figure 120 illustrates as a reminder the scenario 'Horizontal Integration', which encompasses the *Handover* mediators for transferring *System Design Specification* and *Implemented System*. On this level of granularity regarding process definition, it is obvious that 'something' must happen within the mediator. As already stated, a mediator in general represents any kind of meeting, either in person (face-to-face) or virtual, in which appropriate parties get together clarifying process interfaces, which is basically a mapping of artifacts. In Figure 120, this is the middle swim lane which describes "WHAT" needs to be done towards a working collaborative set-up. The remaining issue is on this level of granularity; it is very challenging to define and address "HOW" the work within a mediator needs to be done. This means for the example (Figure 120) that it cannot be generally defined what (e.g. UML diagrams, pre-defined variables etc.) needs to be changed in *System Design Specification* of Organization A to end up with a *System Design Specification* that Organization B can handle and use for its own processes.

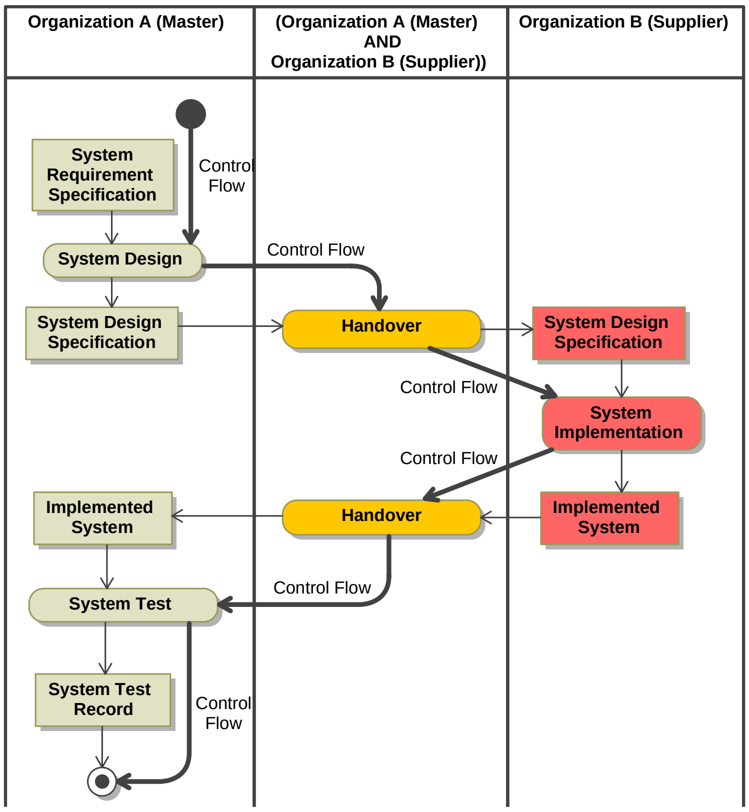


Figure 120: Reminder: Collaborative Process for 'Horizontal Integration'

In order to support the problem, two possibilities are provided in this work:

- Table 4 contains crucial artifacts for software development. It gives guidance of what concrete artifacts might be transferred by using a mediator.
- Chapter 3.3.2 provides an "Artifact Synchronization and Handover Concept" that deals with differences of provided and needed artifacts during hand over. This might have its origin in different methods used to generate different output-formats. These output formats need to be converted so that it can be smoothly used by the collaborative organization(s) or partners.

However, there is still a portion of the problem that depends on the concrete project set-up and the content of the initial processes for collaboration. This is not solved by the Process Integration Approach, but must be solved by the concrete collaboration project.

6.1.2 Interfaces: Different Inputs and Outputs

The examples that illustrate the functionality of defined integration scenarios are consistently defined based on practical experience. The purpose of the scenarios is to illustrate the basic idea of how processes for collaborations can be connected. Nevertheless, these scenarios have character of models, which itself are only simplified illustrations of reality. This means that that cases might occur, which cannot be obviously solved using the Process Integration Approach. As substantiation, please refer to Figure 121 that illustrates the initial processes for 'Merging Integration' scenario.

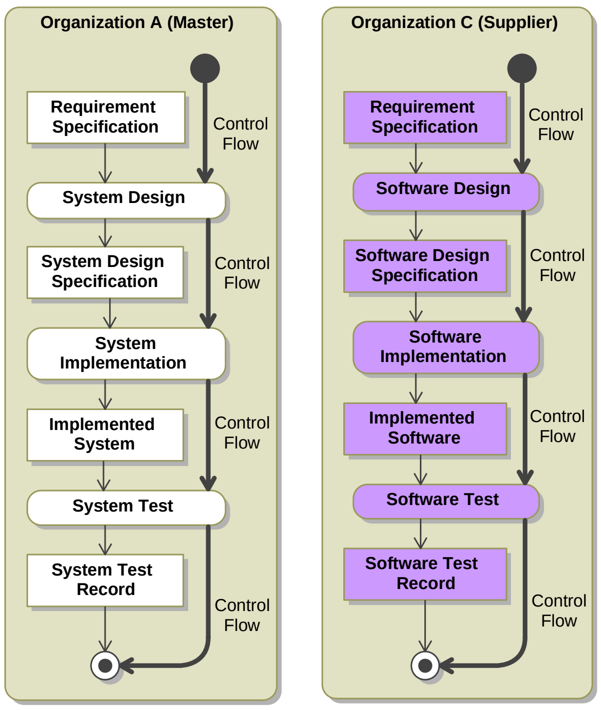


Figure 121: Processes defined on different granularity levels (system vs. software)

If Organization A wants to outsource its *System Implementation* according to 'Horizontal Integration' scenario in Figure 120, it has to transfer its *System Design Specification* to Organization C. However, Organization C (Figure 121) has no action/activity defined in its own processes to be able to handle and process the transferred *System Design Specification*. The problem here is the different definition level of processes (system level vs. software level). This example describes an application case in which straightforward process integration, according to defined mediators of this work, cannot easily be applied. The solution of these types of problems is done by developing a sequence of combined scenarios, which has been conducted in chapter 4.2.5 ('Merging Integration with on single organization' & 'Horizontal Integration').

6.1.3 Empirical Validation Possibilities

A further issue concerning process integration is the conduction of an empirical validation on the level of detail used in the dissertation. Some basic evaluations on the functionality and benefits of the process integration approach has been already shown in [16] in conjunction with agile development.

This integration approach has been defined on such a granularity level, that one has to deal with artifacts or artifact types. Referring to the 'Four Layer Architecture of the Meta-Object Facility' in Figure 17, this would be allocated on action level M2 or M3.

Evaluating the capability of the process integration approach requires the availability of two, almost identical projects to receive data that allow for comparison on the quantitative performance of this approach. In order to get such comparative results the first project needs to run (almost) without any process definition, the second by following the process integration approach. The availability of having two identical projects is very rare in an industrial environment.

In comparison, other evaluation approaches deal with more general topics, like *culture* and *communication* and draw valid conclusions, that distributed development projects are more complicated and, therefore, take more time than other projects conducted in one location [62], [55], [62], [30]. These evaluations would be allocated on a Meta level (M0 or M1 in Figure 17), which makes it more feasible to compare two or even more projects based on these criteria.

6.2 Benefits of the Process Integration Approach

Beside the challenges described in the previous chapter, the process integration approach also comes up with some major advantages that are described in the following.

Velocity of Process Set-up
The illustrated approach is set up in a way which allows process engineers for a quick process set-up at the beginning of any collaboration project. This is due to the fact that the mediators are already pre-defined and guide the respective roles through the set-up by showing what connections in terms of control- and object flows are necessary to define a valid diagram for all participating organizations.

Consistency of Process Integration
The pre-defined mediators support process engineers to create valid process diagrams or workflows. Besides control flows, the approach also provides object flows for each mediator, which shows and gives at least valuable hints for what artifacts need to be included. The object flow is also consistently defined and avoids artifacts that run into dead- ends or missing inputs for defined activities. This means that there are reliable interfaces throughout all participating organizations in any collaboration, which is crucial for its serviceability.

Applicability of Approach
The process integrative approach is not only dedicated and limited to software development process but can also be used for hardware, i.e. mechanical or electrical engineering, although it originates in the software domain. Every development discipline is dependent on collaboration with other parties. Therefore, this approach supports every type of development domain and every type of process, no matter whether it is an iterative or waterfall approach. Even the agile methodology is applicable [16].

Adaptability of Process Integration Approach
This approach covers all potential development scenarios that occur and provides appropriate mediators for it. However, if any project has the need to define another mediator necessary, this approach is also open for further mediator and scenario definition. Although much investigation through "direct observations" have been conducted, this could be required in scenarios that combine one or more development types.

Second, each defined scenario needs to be slightly adapted to every collaborative development case, since every project encompasses different numerous parties and, therefore, requires different artifacts.

Role Concept
During set-up of collaborative processes by using this approach, every activity and process step has a responsible, organizational role assigned. This contributes to effectiveness of the collaborative process during execution, since every participating organization is aware of what needs to be done.

7 Overall Summary and Further Research

Development organizations have had to face a tremendous challenge in the last years: moving from a co-local towards a distributed development environment. This trend is still ongoing and coerces companies to decrease cost while requirements on innovations are constantly growing, and competition increases. In order to stay competitive, development organizations take a chance and “going global” regardless of the fact that challenges are undiminishedly high.

The reasons for that are manifold. Required expertise is typically not readily available from the local market when needed at a certain site. Therefore, development organizations have to expand their search on a global basis to get the right people hired from the job market worldwide. In addition to that, outsourcing of business resources or infrastructure in low cost countries often forces organizations into globalization. Furthermore, globalization may allow organizations to penetrate desired markets to gain market share and organizational growth. This applies especially for emerging countries, where expected growth rates are significantly higher than in already saturated markets.

Additional external business sites around the world need to be included into the organizational core business. This is crucial to efficiently control and make use of any satellite businesses. The inclusion of additional sites works best by having them coordinated via organizational processes. However, these globally- defined processes need to fulfill certain requirements to be capable for distributed development collaborations. On one hand, processes have to stay organization- specific, even if included in a broader net of globally-defined processes. This is important since organizational culture is strongly interrelated to processes. If processes are changed ad hoc culture will turn organization’s work inefficient. On the other hand, processes have to be easy to establish and intuitively to use, which is a necessary aspect to get acceptance from the workforce on processes, especially if they are newly defined.

Therefore, globally distributed process set ups need a structured concept that allow for connecting two or more semantically equal or different processes, while keeping the initial process. Semantically equivalent processes can be connected straightforwardly. Linking two or more processes together that are semantically not equivalent requires at least an interface. Such an interface is called *Mediator* in this work and takes the task to establish a connection that meets all necessary process and cultural requirements mentioned above.

This dissertation defines several *Mediators* that can be used for several collaborative process scenarios:

1. Process Integration with semantically equivalent processes
2. Process Integration without semantically equivalent processes
    a. Horizontal Integration
    b. Additive Vertical Integration
    c. Alternative Vertical Integration
    d. Merging Integration
    e. Hierarchical Integration

These defined scenarios cover all potential collaborative process constellations within distributed development environments. For visualization, UML activity diagrams are used a basis.

Especially in distributed business environments, clearly defined responsibilities are essential to avoid double work and to make sure that no crucial task has been forgotten. Therefore, this process integration approach provides a role concept based on activity/action level. That enables an organization’s process managers to define all responsible persons in charge for any task prior to process execution. Within UML activity diagrams, these responsibilities are defined using the operators “AND”, “OR”, and “XOR”.

Process definition and modeling work need to be done very efficiently, especially in a distributed development context. Increased productivity can be gained by having appropriate tool support available. Chapter 4.3 provides mathematical formalization of processes, which is defined on a graph- based notation. The formalization provides an ideal basis for implementation of a process integration tooling, supporting any process integrator during set up of collaborative and distributed processes. Additionally, formalization helps to understand accurately and communicate the functionality of process integration.

Performance of the process integration approach has been assessed in two case studies:

First, an illustrative case study has been introduced to get familiar with this process integration approach, to get a feeling how the defined approach works in practice, and to identify further questions. The case study makes use of several defined mediators: ‘Horizontal Integration’, ‘Additive Vertical Integration’, and ‘Merging Integration’.

Second, the explorative case study addresses performance measurement of this approach. By use of the ‘Additive Vertical Integration’ mediator, this case brings together one co-located agile development project approach and a waterfall development process in a scientific environment. Based on GQM methodology, four hypotheses (quality factors) have been evaluated that encompass:

- Commitment of Developer
- Risk of Failure
- Amount of Rework
- Communication Effort

Results show that based on the quality factors a distributed development project can perform as well as a comparable project in one location. However, the statistical significance of the result can be challenged since this has been a student's project only.

The communication effort was measured twice as high as in a comparable co-located project. This comes along with Herbsleb’s studies that show that communication effort in globally distributed projects might take 2.5 times longer than comparable projects conducted in one location. However, this weakness is no reason to omit process management in general. As Figure 7 depicts, process management comes with many advantages for development organizations and the corresponding business, e.g., faster processing of orders or cost reduction. These advantages are able to compensate increased communication effort if defined and set up precisely and with care. Therefore, it is better to have any process in place than no process at all. This fact counts especially for collaborative development, which might be distributed all over the world.

Further Research

This dissertation provides the basic structure for collaborative processes. Based on this work, further research may be done in several fields and concerning the following aspects.

Process definition and documentation are very time-consuming if done manually. This means that the procedure of process creation and documentation should be optimized by having tooling in place. The provided formalization is an ideal basis for implementing a tooling prototype for process integration. This supports process set up significantly.

Having one case study in this work as basis, the process integration approach needs to be further approved towards reliability based on empirical data. In order to achieve this target more empirical projects in both scientific and industrial environments need to be conducted that provide a basis to measure the performance of this approach. Performance measurement needs to be done on both hard and soft factors and facts. Hard facts include results from appropriately defined metrics applied, e.g., effort for communication, lead time etc.

Criteria like convenience of handling, visualization, etc., are counted among the soft performance factors, which need to be provided by process definition group. This personal feedback gives a feeling how the entire approach works in practice.

The provided approach for process integration will help development organizations to handle the complexity of globally distributed development projects. The approach makes them analyze interfaces and identify potential roadblocks, prior to process execution, which, in turn, improves the project efficiency regarding time, cost, and quality.

Bibliography

[1] Aggarwal, A., Aspray* W., Berry* O., Lenway S.A., Taylor V.: “Offshoring: The Big Picture”, World Investment Report 2004, p. 148, http://www.scribd.com/doc/19464482/internationalbusiness

[2] Altmann, J., Pomberger G.: “Kooperative Softwareentwicklung: Konzepte, Modell und Werkzeuge”, 4. Internationale Tagung Wirtschaftsinformatik 1999, Hrsg.: August-Wilhelm Scheer; Markus Nüttgens. – Heidelberg: Physica-Verlag, 1999

[3] Al-Ani B., Edwards H. K.: “A Comparative Empirical Study of Communication in Distributed and Collocated Development Teams” International Conference on Global Software Engineering (ICGSE) 2008, p. 35-44, ISBN 978-0-7695-3280-6

[4] Ambler, S. W.: ” Process patterns: building large-scale systems using object technology”, Cambridge University Press/SIGS Books, July 1998, ISBN: 0-521-64568-9

[5] Arnold V., Dettmering H., Engel T., Karcher A.: „Product Lifecycle Management beherrschen“, Springer Verlag 2005, ISBN-13 978-3-540-22997-1

[6] Association for Computing Machinery (ACM): “Globalization and Offshoring of Software - A Report of the ACM Job Migration Task Force”, 2006, http://www.acm.org/globalizationreport/pdf/fullfinal.pdf, October-09, 2010

[7] Avritzer, Hasling, Paulish: “Process Investigation for Global Studio Project Version 3.0”, Second IEEE International Conference on Global Software Engineering, 2007, p. 247-251, ISBN 0-7695-2920-8

[8] Bach N., Biemann T.: “Geschäftsprozessmanagement in Deutschland – Ergebnisse einer Befragung”; in: Ellringmann, H., Schmelzer, H.J.: „Geschäftsprozessmanagement inside“, Hanser Verlag, München 2004, ISBN 978-3446229921

[9] Bartelt, Ch. et al.: “Orchestration of Global Software Engineering Projects (Position Paper)”, Proceedings of the Third International Workshop on Tool Support Development and Management in Distributed Software Projects, collocated with the Fourth IEEE International Conference on Global Software Engineering ICGSE 2009, July 13-16 2009, Limerick, Ireland.

[10] Basili, V. R.; Caldiera, G. & Rombach, H. D.: “The Goal Question Metric Approach - Encyclopedia of Software Engineering”, Wiley, 1994, 646-661

[11] Bass, M., Paulish, D.: “Global Software Development Process Research at Siemens” Third International Workshop on Global Software Development, International Conference on Software Engineering (ICSE) 2004

[12] Ban Al-Ani B., Redmiles D.: “In Strangers We Trust? Findings of an Empirical Study of Distributed Teams”, IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp. 121 - 130, ISBN 978-0-7695-3710-8

[13] Bauer, B., Müller, J. P., Roser, S.: “Adaptive design of cross- organizational business processes using a model-driven architecture” p.103-121, In: Ferstl O. K. et al.: “Wirtschaftsinformatik 2005: Eeconomy, Egovernment, Esociety”, Physica Verlag 2005, ISBN 3-7908-1574-8

[14] Beck, K.: “Extreme Programming Explained”, Addison-Wesley, 2000

[15] Beck, K. et al.: "Manifesto for Agile Software Development". Agile Alliance, 2001, Retrieved 2010-06-14.

[16] Biffl, S., Winkler, D., and Bergsmann, J. (Eds.): SWQD 2012, LNBIP 94, Klein H., Knauss E., Rausch A.: “Scaling Software Development Methods from Co-located to Distributed”, pp. 71–83, 2012, Springer-Verlag Berlin Heidelberg 2012

[17] Explained.At: The Information and Knowledge Portal: “Bitwise operation explained”, http://everything.explained.at/bitwise_operation/, Download November 2011

[18] Boczanski M., Muth M., Scheer A.-W., Segelbacher U., Schmitz W.-G.: „Prozessorientiertes Product Lifecycle Management“, Springer Verlag 2006, ISBN 3-540-28402-8

[19] Boden A., Avram G., Bannon L., Wulf V.: “Knowledge Management in Distributed Software Development Teams – Does Culture Matter?”, IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp.18-27, ISBN 978-0-7695-3710-8

[20] Boehm, B.: "A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes", pp. 14-24, August 1986, DOI: 10.1145/12944.12948

[21] Boehm, B.: “Guidelines for Verifying and Validating Software Requirements and Design Specifications”, Technical Report 1979, published in Journal IEEE Software Volume 1 Issue 1, January 1984, IEEE Computer Society Press Los Alamitos, CA, USA

[22] Boehm B.: „Some Future Trends and Implications of System and Software Engineering Processes“, Wiley InterScience 2006

[23] Boehm B, Basili V. R.: “Industrial Metrics Top 10 List”, IEEE Software, Sept.1987, pp. 84-85

[24] Booch Grady, Rumbaugh James, Jacobson Ivar: “The Unified Modeling Language User Guide”, Addison-Wesley Longman, Amsterdam, July 1998, ISBN 978-0201571684

[25] Bronstein, Von I. N., Semendjajew K. A., Musiol G., Muehlig H.: „Taschenbuch der Mathematik“, Verlag Harri Deutsch 2008, ISBN 978-3-8172-2007-9

[26] Bronwyn Becker, Patrick Dawson, Karen Devine, Carla Hannum, Steve Hill, Jon Leydens, Debbie Matuskevich, Carol Traver, and Mike Palmquist. (2005). *Case Studies.* Writing@CSU. Colorado State University Department of English. Retrieved [2011-08-02] from http://writing.colostate.edu/guides/research/casestudy/.

[27] Carey J., Brent C.: “Framework Process Patterns – Lessons Learned Developing Application Frameworks”; Addison Wesley, April 2002, ISBN 0-201-73132-0

[28] Casey V.: “Leveraging or Exploiting Cultural Difference?”, IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp. 8 - 17, ISBN 978-0-7695-3710-8

[29] Cataldo, M.: “Dependencies in Geographically Distributed Software Development: Overcoming the Limits of Modularity,” PhD Thesis, 2007, School of Computer Science, Carnegie Mellon University: Pittsburgh, PA

[30] Cataldo, M. & Herbsleb, J.D.: “Communication networks in geographically distributed software development” Proceedings, ACM Conference on Computer-Supported Cooperative Work, San Diego, CA, Nov 2008, pp. 579-588

[31] Champy, J.: “X-Engineering the Corporation”, New York: Warner Books 2002, ISBN 978-0446528009

[32] Colette R.: “Modeling the Requirements Engineering Process”, 3rd European-Japanese Seminar on Information Modelling and Knowledge Bases. Budapest, Hungary, June 1993

[33] Damian, D., Izquierdo, L., Singer, J. and Kwan, I.: “Awareness in the wild: why communication breakdowns occur”, IEEE International Conference on Global Software Engineering (ICGSE), 2007, pp. 81-90, Germany, ISBN 978-0-7695-2920-2

[34] Damian, D., Marczak, S., Kwan, I.: “Collaboration Patterns and the Impact of Distance on Awareness in Requirements-Centred Social Networks.” International Requirements Engineering Conference 2007, New Delhi, India, Oct 2007.

[35] Damian, D., Moitra, D.: “Global Software Development: How Far Have We Come?” IEEE Software, Vol. 23, No. 5, 2006

[36] Dasberg, J.: „Product Lifecycle Management: Innovation umsetzen“, 2008, Accenture Broschure

[37] Davenport T.: “Process Innovation: Reengineering work through information technology”, 1992, Harvard Business School Press, Boston, ISBN 978-0875843667

[38] Day, M.: „What is PLM“, April-15, 2002, http://www.caddigest.com/subjects/PLM/select/day_plm.htm; Download September 2010

[39] Deutsches Institute für Normierung (e.V.), ISO 9000:2000 Kapitel 3.4.1

[40] Diaz, M.: “Petri nets: fundamental models, verification and applications”, Wiley 2009, ISBN 978-1-84821-079-0

[41] Ebert Christof, Bvs Krishna Murthy, Namo Narayan Jha: “Managing Risks in Global Software Engineering: Principles and Practices”, pp.131-140, IEEE International Conference on Global Software Engineering (ICGSE) 2008, ISBN 978-0-7695-3280-6

[42] Eigner M., Stelzer R.: „Produktdatenmanagement-Systeme: Ein Leitfaden für Product Development und Lifecycle Management“, 2nd Edition, Springer Verlag 2009, ISBN-13 978-3-540-68401-5

[43] Fink C.A.: „Prozessorientierte Unternehmensplanung. Analyse, Konzepte und Praxisbeispiele“, Wiesbaden 2003

[44] Floyd, C., Züllighoven, H.: „Softwaretechnik“, In: Rechenberg/Pomberger (Hrsg.) Informatik-Handbuch, Hanser Verlag, München, Wien, S. 641-667, 1997

[45] Flvybjerg, B.: “Five Misunderstandings about Case-Study Research”, Sage Publications, Qualitative Inquiry Vol. 12, Number 2, April 2006, pp. 219-245

[46] Forbath T., Brooks P., Dass A.: “Beyond Cost Reduction: Using Collaboration to Increase Innovation in Global Software Development Projects”, IEEE International Conference on Global Software Engineering (ICGSE), 2008, pp.205-209, Bangalore, India, 2008, ISBN 978-0-7695-3280-6

[47] Gadatsch A.: „Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis: Eine Einführung für Studenten und Praktiker“, Vieweg+Teubner 2007, ISBN 978-3834803634

[48] Gadatsch, Andreas: “Grundkurs Geschäftsprozess-Management: Methoden und Werkzeuge für die IT-Praxis”, Vieweg+Teubner, 2009, ISBN 978-3834807625

[49] Gadatsch A., Knuppertz T., Schnägelberger S.: „Geschäftsprozessmanagement - Eine Umfrage zur aktuellen Situation in Deutschland“, Schriftreihe des Fachhandels Wirtschaft der Fachhochschule Bonn-Rhein-Sieg. Bd. 9 St. Augustin 2004, http://www.ifs.tuwien.ac.at/gpm-studie/2003/GPM-Studie-2003_Ergebnisse-Deutschland.pdf, 2010-08-28

[50] Girault,C., Valk, R.: “Petri nets for systems engineering: a guide to modeling, verification, and Applications”, Springer-Verlag Berlin Heidelberg New York, 2003, ISBN 3-540-41217-4

[51] Gotel O., Kulkarni V., Say M., Scharff1 C., Sunetnanta T.: “Quality Indicators on Global Software Development Projects: Does “Getting to Know You” Really Matter?”, IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp. 3-7, ISBN 978-0-7695-3710-8

[52] Grady B., Rumbaugh J., Jacobson I.: “The Unified Modeling Language User Guide” Second Edition, Addision-Wesley 2005, ISBN 0-321-26797-4
[53] Grady B., Rumbaugh J., Jacobson I.: “The Unified Software Development Process, Addison-Wesley, 1999, ISBN 978-0-2015-7169-1
[54] Hack, S.: “Collaborative Business Scenarios –Creating Value in the Internet Economy” http://www.worldinternetcenter.com/Think_Tanks/TTS_Short_List/Stefan_Hack_e_doc.pdf, download September-16, 2010
[55] Handel, M., Herbsleb, J.D.: “What is Chat doing in the workplace?”, Proceedings of ACM Conference on Computer-Supported Cooperative Work (CSCW), New Orleans, LA, 2002, pp. 1-10
[56] Hammer M., Champy, J.: “Business Reengineering”, 1993, Campus, ISBN 978-3593350172
[57] Haskell 98 Language and Libraries The Revised Report: “Pattern Matching”, December 2002, chapter 3.17, http://haskell.org/onlinereport/index.html, July-05, 2011
[58] Havey, M.: “Essential business process modeling”, O’Reilly 2005, ISBN 978-0-596-00843-7
[59] Herbsleb, J.D.: “Global software engineering: The future of sociotechnical coordination.” International Conference on Software Engineering, Fundamental Approaches to Software Engineering, Minneapolis, USA, 2007.
[60] Herbsleb, J.D., Atkins, D.L., Boyer, D.G., Handel, M., & Finholt, T.A.: “Introducing Instant Messaging and Chat into the workplace” Proceedings of ACM Conference on Computer-Human Interaction, Minneapolis, MN, 2002, pp. 171-178
[61] Herbsleb, J.D., Mockus, A.: “An Empirical Study of Speed and Communication in Globally Distributed Software Development”, IEEE Transactions on Software Engineering, 29, 6, June 2003, pp. 1-14
[62] Herbsleb, J.D., Mockus, A., Roberts, J.A.: “Collaboration in Software Engineering Projects: A Theory of Coordination”, 2006, International Conference on Information Systems, Milwaukee, WI
[63] Herbsleb, J.D, Moitra, D.: “Global Software Development”, IEEE Software, March/April (2001), pp. 16-20.
[64] Herbsleb, J. D., Paulish, D. J., Bass, M.: “Global Software Development in Practice: Experience from Nine Projects”, 2005, 27th International Conference on Software Engineering, St. Louis, USA, pp. 524-533.
[65] Hofstede G.: “Culture's consequences: comparing values, behaviors, institutions, and organizations across nations”, Sage Publications, 2001, ISBN 0-8039-7323-3
[66] Humphrey, Watts S.: “Introduction To the Personal Software Process”, Addison Wesley Pub Co Inc., December 1996, ISBN-13: 978-0201548099
[67] Hung, C., Dennis, A., Robert, L.: “Trust in Virtual Teams: Towards an Integrative Model of Trust Formation. Hawaii International Conference on System Sciences (HICSS), Track 1, Volume 1 Jan. 5 - 8, 2004.
[68] IBM news releases: “The Growing Ecosystem of the Automotive Industry”, June-07, 2010, http://www-03.ibm.com/press/us/en/pressrelease/31826.wss, Download: Dec-11, 2011
[69] IDS Scheer: Business Process Report 2005, Saarbrücken 2005
[70] IEEE Computer Society: “12207.0-1996 - IEEE Standard for Information Technology - Software Life Cycle Processes”, http://standards.ieee.org/findstds/standard/12207.0-1996.html

[71] International Standardization Organization (ISO): ISO/IEC 15504 Information Technology, http://www.iso.org/iso/iso_catalogue.htm, Download 2012-04-12

[72] International Standardization Organization (ISO): ISO 9000 Quality Management, http://www.iso.org/iso/iso_9000_selection_and_use.htm, Download 2012-04-12

[73] International Standard ISO/IEC 14977, 1996(E): Information technology – Syntactic metalanguage - Extended BNF, http://standards.iso.org/ittf/licence.html. Download: Jan-14, 2012.

[74] Jacob, A., Brauns, C.: „Der Industrieanlagen-Konsortialvertrag“ Carl Heymanns Verlag, 2006, ISBN 978-3452257147

[75] Jacobson Ivar, Booch Grady, Rumbaugh James: “The Unified Software Development Process”, Addison-Wesley Longman 1999, ISBN 978-0201571691

[76] Jochem, R., Mertins K., Knothe T.: “Prozessmanagement – Strategie, Methoden, Umsetzung” 1. Auflage, Symposium Publishing GmbH, Düsseldorf 2010, ISBN 978-3-939707-56-1

[77] Johansson H., Johansson H. J., Pendlebury A. J.: “Business Process Reengineering: Breakpoint Strategies for Market Dominance”, John Wiley & Sons 1993, ISBN 978-0471938835

[78] Key Highlights of the IT-BPO sector performance in FY 2007/2008 (of India), NAXXCOM report, www.nasscom.org, download: Septmber-29, 2010

[79] Kircher, M., Jain, P., Corsaro, A., Levine, D.: “Distributed eXtreme Programming”, Second international conference on eXtreme Programming and Agile Processes in Software Engineering”. (2001) 66–71

[80] Klein, H.: „Collaborative Processes of Enterprises“, Proceedings 36. Jahrestagung der Gesellschaft für Informatik, 2006 Dresden, pp.683-683, ISBN 978-3-88579-187-4

[81] Klein H., Rausch A., Künzle M., Fischer E.: „Application of Collaborative Scenarios in a Process-Based Industrial Environment“, 36th EUROMICRO Conference on Software Engineering and Advanced Applications, Lille, France, pp.327-330, 01-03 Sept 2010, ISBN 978-0-7695-4170-9

[82] Klein Harald, Rausch A., Fischer E.: “Collaboration in Global Software Engineering Based on Process Description Integration, Lecture Notes in Computer Science: Cooperative Design, Visualization, and Engineering, pp. 1-8, 2009

[83] Klein Harald, Rausch A., Fischer E.: "Towards Process-Based Collaboration in Global Software Engineering," SEAA, 2009 35th Euromicro Conference on Software Engineering and Advanced Applications, 2009, pp.263-266

[84] Kruchten, P.: “The Rational Unified Process - An Introduction”, Addison Wesley 1999, ISBN 0-321-19770-4

[85] Kuhrmann, Niebuhr, Rausch: “Application of the VModell XT – Report from a Pilot Project”, Springer Berlin / Heidelberg, 2006, ISBN 978-3-540-31112-6; http://www.bit.bund.de/nn_388050/BIT/DE/Standards__Methoden/V-Modell_20XT/node.html?__nnn=true

[86] Lamersdorf A., Münch J., Rombach D.: “A Survey on the State of the Practice in Distributed Software Development: Criteria for Task Allocation” IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp. 41-50, ISBN 978-0-7695-3710-8

[87] Lamersdorf A., Münch J., Rombach D.: “Towards a Multi-criteria Development Distribution Model: An Analysis of Existing Task Distribution Approaches”, International Conference on Global Software Engineering (ICGSE) 2008, p. 109 – 118, ISBN: 978-0-7695-3280-6

[88] Lehner F., Scholz M., Wildner S.: „Wissensmanagement. Grundlagen, Methoden und technische Unterstützung, Hanser 2008, ISBN 978-3446219335,

http://www.amazon.de/Wissensmanagement-Grundlagen-Methoden-technische-Unterst%C3%BCtzung/dp/3446219331

[89] Li Xitong et.al.: “A Pattern-based Approach to Protocol Mediation for Web Services Composition”, Proceedings of the Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008) - Volume 00, p.137-146, IEEE Computer Society 2008

[90] Mayer S., Knauss E., Schneider K.: “Distributing a lean organization: Maintaining communication while staying agile”, International Conference on Lean Enterprise Software and Systems (LESS), Springer, pp. 99-103, LESS 2010, Helsinki, DOI: 10.1007/978-3-642-16416-3_14

[91] Marczak, S., Kwan, I., Damian, D.: “Social Networks in the Study of Collaboration in Global Software Teams”, IEEE International Conference on Global Software Engineering (ICGSE), 2009, Munich, Germany, 2007.

[92] Messmer, W.: “Working with India: The Softer Aspects of a Successful Collaboration with the Indian IT & BPO Industry”, Springer Berlin Heidelberg, 1. Auflage, Nov 2008, ISBN-13: 978-3540890775

[93] Meyer B.: “Object-oriented software construction”, Prentice Hall PTR, 1997, ISBN 978-0-136-29155-8

[94] Mistrík I., Grundy J., v. d. Hoek, A. Whitehead J.: „Collaborative Software Engineering“, Springer Verlag 2010, ISBN 978-3642102936

[95] Mockus, A., Weiss, D.M., Bell Labs.: “Globalization by chunking: a quantitative approach”, IEEE Software 2001, pp. 30-37, ISSN 0740-7459

[96] MODAF Meta Model (M3), Version 1.2.004, March 2010, http://www.mod.uk/DefenceInternet/AboutDefence/CorporatePublications/InformationManagement/MODAF/ModafMetaModel.htm, download 2011-08-25.

[97] Muhammad Ali Babar M., Niazi M.: “Implementing Software Process Improvement Initiatives: An Analysis of Vietnamese Practitioners' Views”, International Conference on Global Software Engineering (ICGSE), 2008, ISBN 978-0-7695-3280-6

[98] Nicklisch G. et al.: “IT-Near- und Offshoring in der Praxis”, dpunkt Verlag, 2008 1.Auflage ISBN 978-3-89864-553-1

[99] Niinimäki T., Lassenius C.: “Experiences of Instant Messaging in Global Software Development Projects: A Multiple Case Study”, International Conference on Global Software Engineering (ICGSE), 2008, ISBN 978-0-7695-3280-6

[100] Object Management Group: „Meta Object Facility (MOF) CoreSpecification“, Version 2.4 (Beta), August 2010, http://www.omg.org/spec/MOF/2.4/Beta2/PDF/, 2011-08-25

[101] Object Management Group (OMG): “Unified Modeling Language^{TM} (OMG UML), Superstructure”, Version 2.3, May-05,2010, http://www.omg.org/spec/UML/2.3/Superstructure/PDF, download October-10, 2010

[102] Osterloh M., Frost J.,: „Prozessmanagement als Kernkompetenz: Wie Sie Business Reengineering strategisch nutzen können“, Gabler 2006, ISBN 978-3834902320

[103] Pervez N. Ghauri, P. N., Usunier, J.-C.: „International Business Negotiations (International Business and Management Series), Emerald Group Publications”, 2003, ISBN 978-0080442921

[104] Piri A., Niinimäki T., Lassenius C.: “Descriptive Analysis of Fear and Distrust in Early Phases of GSD Projects”, IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp.105-114, ISBN 978-0-7695-3710-8

[105] Pohl K., Böckle G., van der Linden, F.: „Software product line engineering: foundations, principles, and techniques“, Springer 2005, ISBN 978-3-540-24372-4

[106] Prikladnicki et al.: “Distributed Software Development: Practices and Challenges in Different Business Strategies of Offshoring and Onshoring”, Second IEEE International Conference on Global Software Engineering (ICGSE) 2007, p.262 - 271, ISBN 0-7695-2920-8

[107] Prikladnicki R.: “Exploring Propinquity in Global Software Engineering”, IEEE International Conference on Global Software Engineering (ICGSE), 2009, pp. 133-142, ISBN 978-0-7695-3710-8

[108] Prikladnicki, R., Audy, J. L. N., Evaristo, R.: “A Reference Model for Global Software Development: Findings from a Case Study,” 2006, 1st International Conference on Global Software Engineering (ICGSE), Florianopolis, Brazil, pp. 18-25.

[109] Prikladnicki, R., Audy, J. And Evaristo, J.R.: “Distributed Software Development: Toward an Understanding of the Relationship between Project Team, Users and Customers, Proceedings of the 5th International Conference on Enterprise Information Systems (ICEIS’03), 2003

[110] Prikladnicki R., Damian D., Audy J. L. N.: “Patterns of Evolution in the Practice of Distributed Software Development in Wholly Owned Subsidiaries: A Preliminary Capability Model”, International Conference on Global Software Engineering (ICGSE) 2008, p. 99 – 108, ISBN: 978-0-7695-3280-6

[111] Ravichandran, Von D.: „Programming with C++“, 2nd edition, Tata McGrawhil 2003, ISBN 0-07-049488-6

[112] Royce, W.W.: “Managing the Development of Large Software Systems”, Proceedings of IEEE WESCON 26, August 1970, pp. 1-9.

[113] Rumbaugh J., Jacobson I., Booch G.: “The Unified Modeling Language Reference Manual”, Addison-Wesley 1999, ISBN 978-0201309980

[114] Rummler, G. A., Brache A. P.: „Improving Performance: How to manage the white space on the organizational chart”, John Wiley & Sons 1995, ISBN 978-0787900908

[115] Rupp C., Queins S., Zengler B.: UML 2 Glasklar, Praxiswissen für die UML Modellierung, 3.Auflage; Carl Hanser Verlag 2007, ISBN 978-3-446-41118-0

[116] Sääksvuori A., Immonen A.: “Product Lifecycle Management”, 3rd Edition, Springer Verlag 2008, ISBN-13 978-3-540-78172-1

[117] Samek M.: „Practical UML statecharts in C/C++: event-driven programming for embedded systems“, Elsevier 2009, ISBN 978-0-7506-8706-5

[118] Sangwan R., Bass M., Mullick N., Paulish D.J., Kazmeier J.: “Global Development Handbook”, Auerbach Publications, Taylor & Frances Group, 2007, ISBN 0-8493-9384-1

[119] Scacchi, W.: „Process Models in Software Engineering“ in J.J. Marciniak (ed.), Encyclopedia of Software Engineering, Second Edition, John Wiley and Sons, Inc, New York, December 2001

[120] Schäling, B.: „Der moderne Softwareentwicklungsprozess mit UML“, Kapitel 3: Das Aktivitätsdiagramm, http://www.highscore.de/uml/titelseite.html, 2010-08-28

[121] Scheer A.-W.: ”ARIS - vom Geschäftsprozess zum Anwendungssystem”, Springer 2002, 4. durchgesehene Auflage, ISBN 3-540-65823-8

[122] Schmelzer, H.J., Sesselmann W.: „Geschäftsprozessmanagement in der Praxis“, 6. bearb. Auflage, Carl Hanser Verlag München, 2008

[123] Schuh, G.: “Enzyklopädie der Wirtschaftsinformatik”, Definition Product Lifecycle Management (PLM), September-08, 2010, http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/informationssysteme/Sektorspezifische-

Anwendungssysteme/Produktionsplanungs--und--steuerungssystem/Product-Life-Cycle-Management

[124] Schulte-Zurhausen M.: “Organisation”, Vahlen 2010, ISBN 978-3800637362

[125] Schwaber, Ken: “Agile Project Management with Scrum”, Microsoft Press 1st Edition, March 2010, ISBN 978-0735619937

[126] Sendler U.: „Das PLM-Kompendium: Referenzbuch des Produktlebenszyklus Managements“, Springer 2009, ISBN 978-3540878971

[127] Serce F.C., Alpaslan F.-N., Swigger K., Brazile R., Dafoulas G., Lopez V., Schumacker R.: “Exploring Collaboration Patterns among Global Software Development Teams” International Conference on Global Software Engineering (ICGSE), 2009, pp. 61-70, ISBN 978-0-7695-3710-8

[128] Siemens AG Quarterly Business Report, 4th Quarter 2008, http://www.siemens.com/press/pool/de/events/2008-q4/2008-q4-eckdaten-d.pdf Download: September-29, 2010

[129] Siemens AG, Business Report, Fiscal Year 2009, http://www.siemens.de/ueberuns/Documents/d09_00_gb2009.pdf, Download: September-29, 2010

[130] Siemens AG – Das Unternehmen 2010, as of May-10, 2010, http://www.siemens.de/ueberuns/Documents/das_unternehmen_2010.pdf, Download: September-29, 2010

[131] Siemens AG – Annual Report 2011, 2012, Jan-24, http://www.siemens.com/annual/11/_pdf/Siemens_GB_2011.pdf, Download, 2012-04-26

[132] Software Engineering Institute (SEI), CMMI Product Team: “CMMI® for Development, Version 1.3”, TECHNICAL REPORT, CMU/SEI-2010-TR-033, November 2010, http://www.sei.cmu.edu/downloads/cmmi/10tr033.docx, 31.05.2011

[133] Sooraj P, Pratap K.J. Mohapatra: “Developing an Inter-site Coordination Index for Global Software Development”, International Conference on Global Software Engineering (ICGSE) 2008, p. 119-128, ISBN 978-0-7695-3280-6

[134] Spillner A.: „W-model – test process parallel to the development process”, Proceedings of Jornada sobre Testeo de Software (JTS 2004), ITI Instituto Tecnológico de Infomática, Universidad Politécnica de Valencia, Spain, March 25-26th 2004

[135] Stapel, K.; Lübke, D. & Knauss, E.: “Best Practices in eXtreme Programming Course Design”, Proceedings of the 30th International Conference on Software Engineering (ICSE 2008), ACM Press, 2008, 769-776

[136] Steingart, G.: „Der Erfolgsfilm läuft rückwärts“, 15.09.2006, http://www.spiegel.de/wirtschaft/0,1518,436480,00.html, download September-09, 2010

[137] Töpfer A.: “Geschäftsprozesse analysiert & optimiert”, Luchterhand 2006, ISBN 978-3472027539

[138] Trikkula, V.: „Globalization of R&D and Product Development Set to Grow“ June-10, 2010, http://www.globalservicesmedia.com/IT-Outsourcing/Product-Development/Globalization-of-RandD-and-Product-Development-Set-to-Grow/22/4/9710/GS100610968441, download Sep-10, 2010

[139] Trompenaars A., Hampden-Turner C.: “Riding the waves of culture: understanding cultural diversity in global business”, B&T, 1998, ISBN 978-0786311255

[140] Tyler C. G., Baker S. R.:” Business genetics: understanding 21st century corporations using xBML”, Wiley 2007, ISBN 978-0-470-06654-6

[141] United States Government Accountability Office: Report to Congressional Committees, OFFSHORING U.S. Semiconductor and Software Industries Increasingly Produce in China and India, September 2006, GAO-06-423, http://www.gao.gov/new.items/d06423.pdf, download October-09, 2010

[142] Wang Yingxu, King Graham: „Software Engineering Processes: Principles and Applications",Crc Pr Inc, 2000, ISBN 978-0849323669 (http://www.amazon.de/Software-Engineering-Processes-Principles-Applications/dp/0849323665/ref=sr_1_1?ie=UTF8&s=books-intl-de&qid=1283362352&sr=1-1#reader_0849323665)

[143] Weske, M.: „Business Process Management: Concepts, Languages, Architectures", Springer Verlag, Berlin; 1st Edition 2007, ISBN-13: 978-3540735212

[144] White S. A.: „Process Modeling Notations and Workflow Patterns", March 2004, Future Strategies Inc., Pages: 1-25, Mendeley - Computer and Information Science, ISBN 0-970-35096-1

[145] Wichmann Klaus-Peter: "Offshore Zusammenarbeit erfolgreich etabliert: Ein Praxisbericht über ein Migrationsprojekt im Maschinenbau", SIGS DATACOM Gmbh, ObjektSpektrum Mai/Juni 2008 Nr.3, p.50-55

[146] Wikipedia – The Free Encyclopedia: "Diagram", http://en.wikipedia.org/wiki/Diagram, 2011-06-24

[147] Wikipedia – The Free Encyclopedia: "State Diagram", http://en.wikipedia.org/wiki/State_diagram, 2011-06-24

[148] Wilson, J. M., O'Leary, M. B., Metiu, A., Jett, Q. R.: "Perceived Proximity in Virtual Work: Explaining the Paradox of Far-but-Close", 2008, Organization Studies, 29(07), pp. 979-1001.

[149] Wohlin, C.; Runeson, P.; Höst, M.; Ohlsson, M. C.; Regnell, B. & Wesslén, A.: „Experimentation In Software Engineering: An Introduction.", Kluwer Academic Publishers, 2000

[150] Yin, Robert K.: "Case study research: design and methods Band 5 von Applied social research methods series", Sage Publications, 2009, ISBN 9781412960991

[151] Yin, Robert K.: "CASE STUDY METHODS", COSMOS Corporation, REVISED DRAFT, January 20, 2004, download June 10, 2011, http://www.cosmoscorp.com/Docs/AERAdraft.pdf

[152] Yongchareon S., Liu C.: "A Process View Framework for Artifact-Centric Business Processes", Springer-Verlag Berlin Heidelberg 2010, R. Meersman et al. (Eds.): OTM 2010, Part I, LNCS 6426, pp. 26–43, 2010

[153] Zimmermann, E.: „Siemens nutzt EU-Osterweiterung zur Verlagerung von Arbeitsplätzen und zum Lohnabbau", http://www.trend.infopartisan.net/trd0504/t140504.html, 06.September 2010, mirrored from http://www.wsws.org/de/2004/apr2004/siem-a23_prn.html (release 23.April 2004)

List of Figures

List of Tables