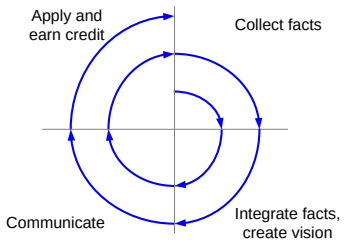
System Architecting



Gerrit Muller
University of South-Eastern Norway-NISE
Hasbergsvei 36 P.O. Box 235, NO-3603 Kongsberg Norway
gaudisite@gmail.com

Abstract

This book addresses the role of the system architect and the role of the system architecting process within the business:

- What is the role and the task of the (system) architect?
- How does the architect do his work?
- What are the main activities of the architect?
- How does the architect fit in the organization?
- What kind of person is the architect?
- How does architecting fit in the business?

Note: this book is likely to be refactored in smaller books in the future.

version: 1.8 status: preliminary draft April 6, 2021

Contents

List of Figures

List of Tables

Introduction

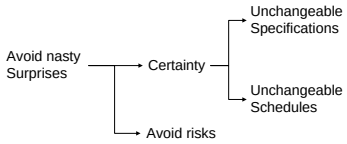
This book bundles the articles and intermezzo's produced by the Gaudí project.

At this moment the book is in its early infancy. Most articles are updated based on feedback from readers and students. The most up to date version of the articles can always be found at [16]. The same information can be found here in presentation format.

Chapters can be read as autonomous units. The sequence choosen here is more or less top down, hopping from one viewpoint to the next. On a regular base a sidestep ("Intermezzo") is being made, either to describe a more fundamental notion, or to propose a more challenging point of view.

Note: this book is likely to be refactored in smaller books in the future.

System Architecture: The Silver Bullet?



0.1 Introduction

The expectation level with respect to processes in general and the system architecture process in particular can vary from skeptical to blind faith. The skeptics have experienced that horrible specifications and designs can be pursued under the grand name of Architecture. The followers with blind faith are at the opposite end of the spectrum, their believe in processes inhibits them from seeing the limitations and constraints from the processes applied.

The central message of this Intermezzo is:
Silver Bullets do not exist.

This Intermezzo intends to set realistic expectation levels with respect to the System Architecture Process, and describes the ingredients for successful application.

0.2 Why System Architecture?

System Architecting is a means to create systems efficient and effective, by supplying overview, by guarding consistency and integrity, and by balancing. In other words the System Architect helps the development team to find its way in a rather complex, dynamic and uncertain world.

From psychological point of view people apply their own survival mechanisms, when they perceive a threat. One of the most common survival mechanisms is *The Quest for Certainty*, see subsection 0.2.1.

Unfortunately System Architecting will never remove all uncertainties, see subsection 0.2.2. The application of a system architecture process can help in the risk management, amongst others by prevention, and by minimizing impact.

Successful application of system architecture is far from trivial, section 0.3 describes **how** the System Architecture Process should be applied to meet the goals of efficiency and effectively.

0.2.1 The Quest for Certainty

This section provides a caricatural view on human behavior based on a free interpretation of the Maslow Hierarchy of Needs, as discussed for instance in [7]. This exaggerated view matches with the security needs in the lower Maslow Hierarchy. Note that less defensive behavior can be triggered by needs in the higher layers, were words such as adventurous en explorative are being used.

The majority of people, including managers and engineers, have a need for certainty. Their ideal is to have stable, unchangeable sets of specifications, schedules et cetera. This (hopefully) isolates them from the nasty surprises of reality see table 1.

- incompetent people
- human mistakes
- lack of collaboration or synergy
- misunderstanding or miscommunication
- changing markets
- fast moving competition
- unforeseen physical, chemical, mechanical properties
- mother nature (illnesses, floods)

Table 1: *Nasty Surprises of Reality*

Unfortunately these nasty surprises are a fact of life. Our human capability to control these phenomena is quite limited.

Risk management can help to be more robust. However risk management certainly does not remove these phenomena and it also does not reduce the consequences to zero. Risk management balances probability, effect, and cost.

People with a need for certainty are willing to accept any method or process which promises certainty. In other words *certainty* appears to be their personal key driver. It is better to rephrase this key driver as *to avoid nasty surprises*, which is closer to the internal motivation at the one hand and which gives a handle later on to manage the expectations. Figure 1 visualizes these drivers.

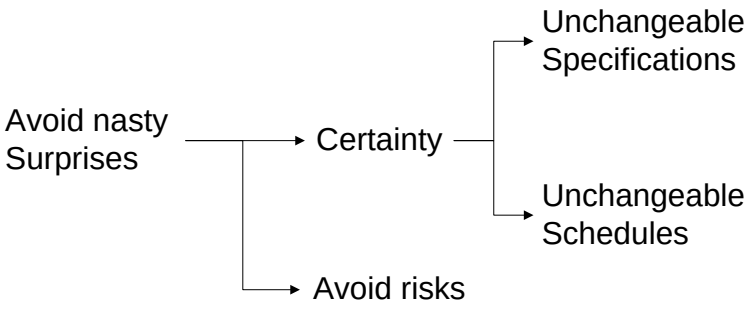


Figure 1: Personal key driver *to avoid nasty surprises*

0.2.2 Disclaimer; Setting the Expectations to a realistic level

The Gaudí Project will not deliver a Plug-and-Play System Architecture Process. System architects which have read all the articles and followed the course will not automatically be successful.

The Gaudí project will deliver a large set of consistent background material for system architects. This material ranges from process and architecture principles, providing insight and understanding, to more specific how-to's which provide more directly applicable guidelines.

The competent system architect will use the material by customizing it to the specific problem to be addressed. At the same time the system architect will have to interact with the environment to share this customized way of working.

Whenever the material is applied literal, this is a strong indication that the organization and the system architect do not work explicit enough on the way of working.

0.3 How: Critical Success Factors

Ingredients for an effective application of a system architecture process are shown in table 2.

No method or process will function without these critical success factors. A process can not be used as substitute for know how or common sense.

0.3.1 Know-How

The core of the system architecture work is know-how, ranging from pure technology know-how to application and business know-how. Active control on a broad basis is a prerequisite for a system architect.

- Know-How
- Common Sense
- Pragmatics
- Critical attitude
- Social skills
- Drive
- Vision

Table 2: *Critical Success Factors for an effective application of a System Architecture Process*

0.3.2 Common Sense

Most problems encountered during Product Creation require common sense to solve them. Mechanistic approaches severely limit the solution space, resulting in complex solutions. System architects are capable of "lateral" thinking, allowing solutions in previously unexpected directions.

0.3.3 Pragmatics

The holistic approach can easily derail in a sea of seemingly conflicting requirements and viewpoints. The system architect needs a significant amount of pragmatism to be selective and focused, while being holistic in the back of his mind.

0.3.4 Critical attitude

Clear diagrams, tables with facts and smooth presentations give the impression of high quality and increase the confidence. However these same diagrams, tables and presentations conceal the forgotten, misinterpreted, or underestimated facts. The system architect must always be alert, for instance by asking questions as shown in table 3.

0.3.5 Drive

A good system architect has a passion for his architecture, it has an emotional value.

An architect which is working entirely according to the book, obediently going through the motions, will produce a clinical architecture without drive or ownership.

Good architectures have an identity of themselves, which originate in the drive of the architect. Such an architecture is an evolving entity, which is appreciated by the stakeholders.

- Do we address the right problem or requirement?
- Is the customer/user on-board?
- Is this design adequate?
- Consists the input data from facts, wishes or ideas?
- Do we need so many people for the implementation?
- Does this process or organization fit the problem?

Table 3: *Critical Attitude: Examples of questions to be asked by the System Architect*

0.3.6 Vision

The system architect needs to have a vision to be able to provide direction. A vision enables an evolution of existing architectures to desired architectures. Having vision is not trivial, it requires a good understanding of needs (the problem) and means (the solution) plus the trends (opportunities and threats) in both needs and means.

0.4 Summary

The one sentence summary of this intermezzo is: *Silver bullets do not exist*. Table 4 gives a bullet-wise summary.

- Most people want to avoid nasty surprises
- Most people are looking for certainty
- Silver Bullets do not exist
- System Architecture is not a golden bullet
- Critical Success Factors: Know-How, Common Sense, Pragmatics, Critical attitude, Drive and Vision

Table 4: *Summary*

0.5 Acknowledgements

Hans Gieles suggested improvements to increase the cohesion and the red line in this Intermezzo.

Henk Obbink and Angelo Hulshout and many others pointed out that ”The Golden Bullet” should have been ”The Silver Bullet”, which has been changed

finally. Adriaan van den Brand pointed out the forgotten change of golden into silver.

Eugene Ivanov pointed out that evolution aspects were missing. The result is the addition of vision as critical success factor.

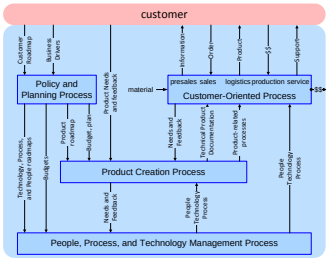
Steve R. Nanning indicated unclarity in the abstract and criticized the too negative phrasing of the *Quest for Certainty*. Stephen Boggess added a question to the list in Section ”Critical Attitude”. Ning Lu added more surprises and critical success factors.

Part I

Processes

Chapter 1

Process Decomposition of a Business



1.1 Introduction

This chapter positions the system architecting process in a wider business scope. The objective of this chapter is to provide system architects insight in the business processes and especially in the processes where system architects actively contribute.

The focus is on companies that create physical products. Other types of businesses, such as solution providers, services, courseware, also need systems architecting. The process structure will deviate somewhat from the structure presented here. See Intermezzo “Products, Projects, and Services” for a discussion on the processes in these other businesses.

1.2 Process Decomposition

The business process can be decomposed in 4 main processes as shown in Figure 1.1. We have on purpose ignored the supporting and connecting processes. This simplification will allow us to get a number of more fundamental insights in the main processes.

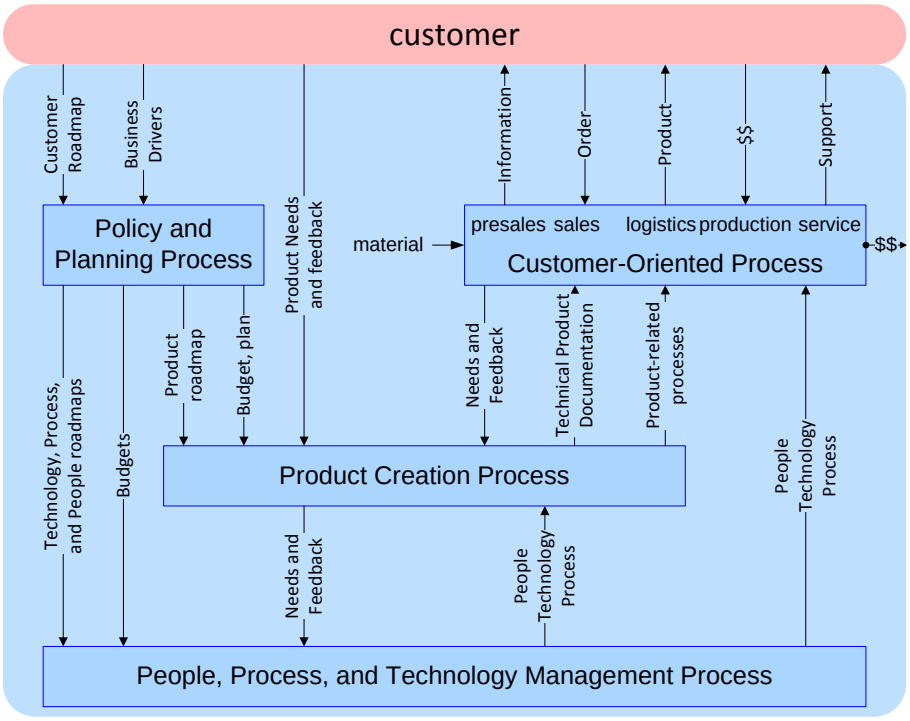


Figure 1.1: Simplified decomposition of the business in 4 main processes

The function of the 4 main processes is:

Customer Oriented Process performs in repetitive mode all direct interaction with the customer. This process is the cash flow generating part of the enterprise. All other processes only spend money.

Product Creation Process feeds the Customer Oriented Process with new products. This process ensures the continuity of the enterprise by creating products that keep the company competitive. In this way the Product Creation Process enables the Customer Oriented Process to generate cash flow in the near future as well.

People, Process, and Technology Management Process manages the competencies of the employees and the company as a whole. The competencies of the employees and the company are the main assets of a company.

Policy and Planning Process is the management process. The Policy and Planning Process defines the strategy, the long term direction of the company, and it balances the shorter term tensions between the three other main processes. The Policy and Planning Process uses roadmaps and budgets to define the

direction for the other processes. Roadmaps give direction to the Product Creation Process and the People, Process and Technology Management Process. For the medium term these roadmaps are transformed in budgets and plans, which are committal for all stakeholders.

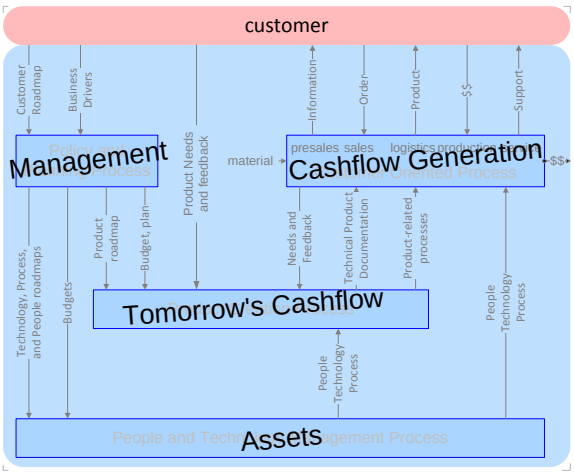


Figure 1.2: Decomposition of the business in 4 main processes, characterized by their financial meaning

The 4 processes as described here are different in nature. The Customer oriented process executes over and over a well defined set of activities. The system architect does not participate in active role in this process. However since the Customer Oriented Process is the main customer of the Product Creation Process, it is crucial that the system architect understands, or better has experienced, the Customer Oriented Process.

The system architect is in continuous interaction with many stakeholders, mostly about technical aspects. From this perspective the architect will generate inputs for the People and Technology Management Process. This might even result in participation in this process for instance by coaching, participation in the appraisal process, or participation in technology studies.

The number of instances of each process is related to different entities:

Customer Oriented Process: Depends on geography, customer base, and supply chain.

Product Creation Process: One per entity to be developed, where such an entity can be a product family, a product, or a subsystem.

People and Technology Management Process: One per "competence", where a competence is a cohesive set of technologies and methods.

Policy and Planning Process: One per business. This is the pro-active integrating process.

The evolutionary developments of product variants and new releases are seen as individual instances of the Product Creation Process. For example the development of a single new feature for an existing product is performed by following the entire Product Creation Process. Of course some steps in the process will be (nearly) empty, which does not cause any harm.

1.3 Process versus Organization

This process decomposition is not an organization, see Intermezzo "What is a Process". A single person can (and often will) fulfill several roles in different processes.

System architects specifically spend most of their time in Product Creation Process (circa. 75%), a considerable amount of time in the Policy and Planning Process (circa 20%), and a small fraction of their time in the People, Process and Technology Management Process.

Most engineers will spend a small amount of time in the People, Process, and Technology Management Process, working on technologies and capabilities, while the majority of their time is spend in the Product Creation Process.

1.4 Value Chain and Feedback

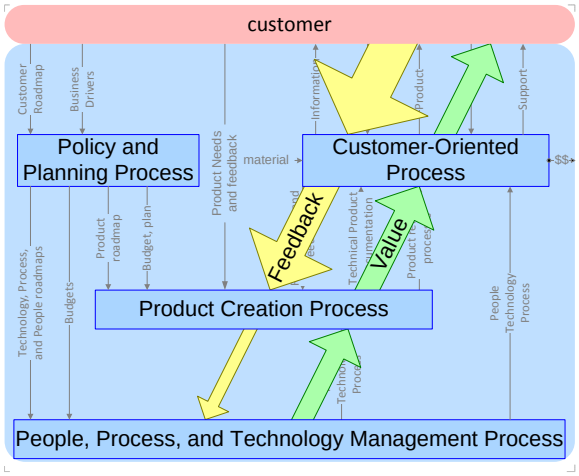


Figure 1.3: The value chain and the feedback flow in opposite direction

The value chain in these processes starts at the assets in the People, Process, and Technology Management Process. The assets are transformed into potential money by the Product Creation Process. The Customer Oriented Process finally turns it into real money. Figure 1.3 shows the value chain.

The feedback flows in the opposite direction, from customer via the Customer Oriented Process and the Product Creation Process to the People Technology and Process Management Process. Customer will communicate mostly with sales and service people. Needs and complaints are filtered by the reporting system before the information reaches Product Creation Teams. Only a small part of the customer feedback reaches the People, Process, and Technology management.

This simple model explains why the knowledge about the customer gets less deeper in the organization. The consequence is that internal technology and process provides show to little concern for urgent customer or business challenges; the sense of urgency seems to be lacking. We can take preventive measures, such as sending process and technology managers to customer sites, once we are aware of the gap caused by this natural information flow.

1.5 Decomposition of the Customer Oriented Process

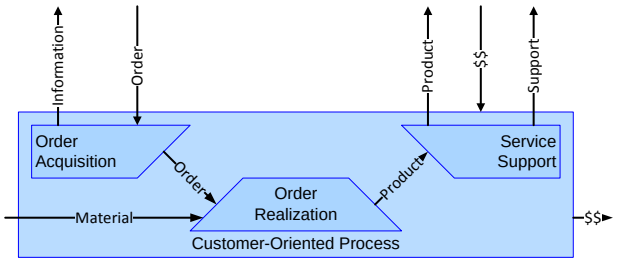


Figure 1.4: Decomposition of the Customer Oriented Process

The Customer Oriented Process is often the largest process in terms of money. From business point of view it is an oversimplification to model this as one monolithic process. Figure 1.4 shows a further decomposition of this process.

The Order Acquisition Process and the Service Support Process are operating quite close to the customer. The Order Realization Process is already somewhat distant from the customer.

The owners of all these three processes are stakeholders of the Product Creation Process. Note that these owners have different interests and different characteristics.

1.6 Extended Process Decomposition; Generic Developments

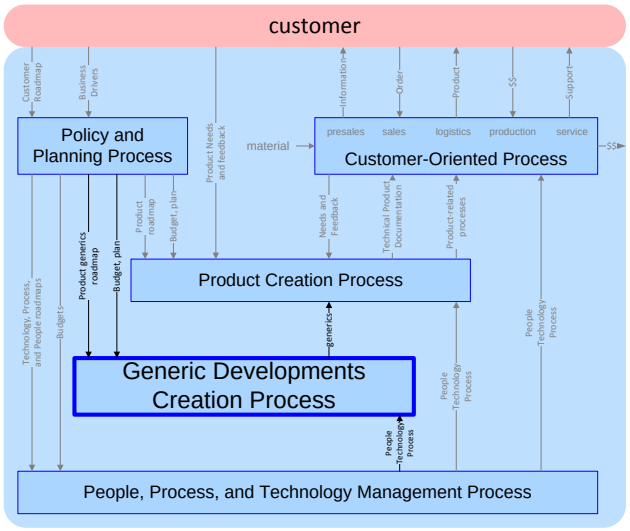


Figure 1.5: The Process Decomposition extended with a generic developments creation process

Companies which develop product families try to capitalize on the commonality between the members of the product family. This is often implemented by the development of common subsystems or functions. In the diagram 1.5 this is called **Generic Developments Creation Process**. A wide variety of names is used for this phenomena, such as re-use, standard design, platform et cetera.

1.7 Acknowledgements

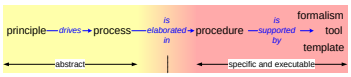
Discussions with and critical comments from Rard de Leeuw, Jürgen Müller, Henk Obbink, Ben Pronk and Jan Statius Muller helped to shape, to improve the structure and to sharpen the contents of the article "Positioning the System Architecture Process". This intermezzo is based on the first sections of this article. I am grateful for their contribution.

Discussion with Ab Pasman helped to remove some architect bias from the process decomposition, by providing a further decomposition of the Customer Oriented Process.

Jaap van der Heijden helped to improve the layout of the diagrams and with the document structure.

Chapter 2

What is a Process?



2.1 Introduction

We rely in this part heavily on the notion of a process. This intermezzo is defining "process" for the context of this book. We define "process", since this word is heavily overloaded in our daily world. We also discuss the relationship of processes with organizations and the drive for process improvement.

2.2 What is a process

We use process as an abstracted way of working. A process can be characterized the attributes shown in Figure 2.1

In [10] the following definition is given:

A process is an activity which takes place over time and which has a precise aim regarding the result to be achieved. The concept of a process is hierarchical which means that a process may consist of a partially ordered set of subprocesses.

This definition parallels the characterization above. It adds explicitly the potential hierarchical decomposition of the process itself.

The notion of a process can be seen as one step in an abstraction hierarchy, as shown in 2.2. The most abstract notion in this hierarchy is the "principle". A principle is a generic insight that can be used for many different purposes. An example of a principle is *decomposition*: Whenever we have something big, e.g.

Purpose What is to be achieved and why

Structure How will the goal be achieved

Rationale What is the reasoning behind this process

Roles What roles are present, what responsibilities are associated, what incentives are present, what are the criteria for these roles

Ordering What phasing or sequence is applied

Figure 2.1: Process Attributes

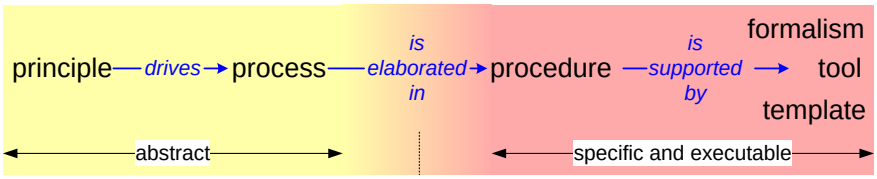


Figure 2.2: A process within an abstraction hierarchy

a problem or project, then we can decompose it in smaller pieces. These smaller pieces are easier to solve or create than the original big one.

A process is rather abstract. It describes the essentials of the purpose, structure, rationale, roles and timing, leaving plenty of implementation freedom. The power of a process is its abstraction, which enables its application in a wide range of applications, by tailoring its implementation to the specific application.

A process can be tailored and elaborated in one or more procedures that describe cookbook-like what needs to be done when and by whom. The why in a procedure has often disappeared, to be replaced by practical information for the execution.

The implementation of a procedure can be supported by tools, notations, templates and other means.

In practice managers and employees ask for tools (means) and procedures (what and how). However, without understanding of the thinking behind the procedure (why), as given in the process, these tools and procedures can be meaningless. The process captures the rationale behind procedures, tools, notations, templates, and other means.

2.3 The relation between Processes and Organizations

Traditional management is focused on “organizations”. Where organization are characterized by the attributes shown in Figure 2.3.

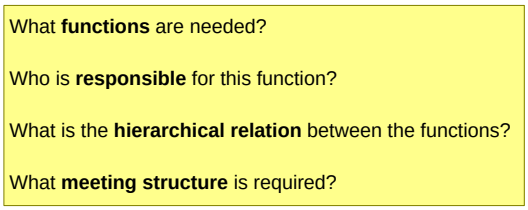


Figure 2.3: Organization Attributes

This management views is insufficient in today’s fast moving complex world. The weak spots of the organizational view are shown in Figure 2.4.

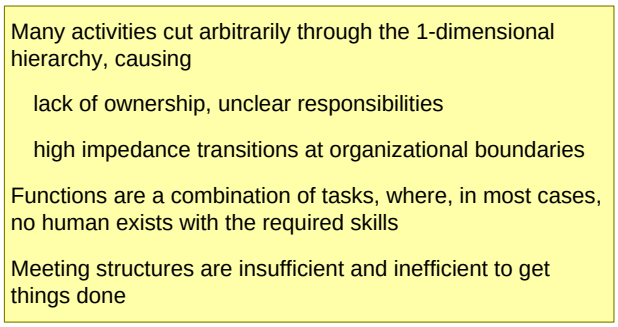


Figure 2.4: Weaknesses of the organizational view

Processes are more modern instruments for management. Many processes are required to ensure the effective functioning of an organization. These processes are interrelated and overlapping. Processes are non-orthogonal and don’t fit in a strict hierarchical structure.

Most complex product developments don’t fit in the classical hierarchical organization model, but require a much more dynamic organization model, such as the currently popular more chaotic network organization. Processes are the means which help to ensure the output of dynamic organization models such as a network organization.

Processes can be seen as the blueprint for the behavior of the people within the organization. People will fulfill multiple roles in multiple processes. The process description is intended to give them an hold on what is expected from them.

All important activities will be covered by a process, requiring the definition of

ownership, relation with other processes et cetera. The allocation of roles to people is much more dynamic than in conventional hierarchies. More dynamic allocation enables a better match between personal capabilities and required skills. In practice dynamic allocation leads to more distribution of responsibilities, making it more feasible to match capabilities and skills.

The 80/20 rule is also valid for processes: 80% of the behavior is covered by the processes, while 20% requires independent creative behavior. An organization without processes drowns in chaos, while an organization which blindly implements them will be killed by its own inertia, its inability to adapt to the fast changing world.

For reasons of continuity and stability an hierarchical organization will remain. The slowest evolving dimension is mostly used as a basis for this hierarchy. This hierarchy functions as anchor point for people in the continuously changing process world, but should play only a minor role in the entire operation.

The **Centurion** turn around operation within Philips, orchestrated by CEO Jan Timmer in the early nineties, urged the Philips managers and employees to change from an introvert organization point of view to an external result oriented process point of view.

2.4 Process Improvement

Urged by competitive pressure organizations look for ways to improve their efficiency. Many opportunities for improvement have a strong process component.

The 7S model by McKinsey gives a practical way to improve an organization in a balanced way. The message behind this model is that at least 7 views must be balanced when changing an organization. See Figure 2.5 for the 7 views.

The most common pitfall in improvement programs is the over-emphasis on the process component, or worse the isolation of the process improvement. Organizations assessing their maturity level, for instance by Maturity Models [22], quite often get too much process focus. The Process Improvement Officer[1] is focused on process issues only. Hence where the process view is introduced as an extrovert result oriented approach, it suddenly turns into an introvert improvement program, where business goals and drivers are unknown.

This is a quite sad situation: The opportunities for improvement are ample with a strong process component, however due to the wrong focus a negative effect is obtained (such as rigid procedures).

Recommendation: Process improvements should originate from the directly involved people, for instance project leaders, engineers, architects et cetera. Invite participation by this group in such a way that they feel the ownership.

[1] The existence of this function in itself is quite dangerous, it invites the unbalanced isolated "improvement" behavior

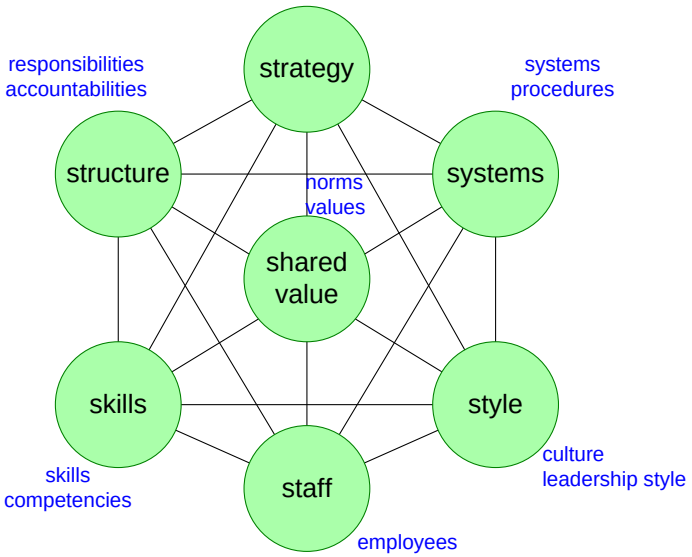


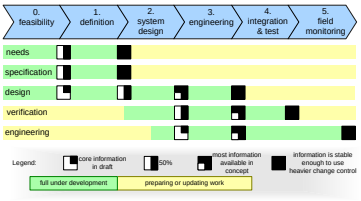
Figure 2.5: McKinsey 7S model

2.5 Acknowledgements

Discussions with and critical comments from Rard de Leeuw, Jürgen Müller, Henk Obbink, Ben Pronk and Jan Statius Muller helped to shape, to improve the structure and to sharpen the contents of the article "Positioning the System Architecture Process". This intermezzo is based on the first sections of this article. I am grateful for their contribution.

Chapter 3

The Product Creation Process



3.1 Introduction

The Product Creation Process described how an organization gets from a product idea to a tested system and all product documentation that is required for the Customer Oriented Process. System Architects spend most of their time in the Product Creation Process. This chapter describes the Product Creation Process, including organizational aspects and the roles of people within the process.

3.2 The Context of the Product Creation Process

Figure 1.1 shows the context of the Product Creation Process in the decomposition of the business in 4 main processes. From Product Creation Process point of view the Policy and Planning Process determines the charter for the Product Creation Process. The Technology and People Management Process supplies people, process and technology enabling the Product Creation. The Customer Oriented Process is the customer: it receives and uses the results of Product Creation.

The Product Creation Process has a much wider context than the conventional “Research and Development” or “Development and Engineering” departments. The Product Creation Process includes everything that is needed to create a new product, for instance it includes:

- Development of the production process

- Design of the logistics flow and structure
- Development of required services
- Market announcement
- Market introduction

In other words the Product Creation Process is a synchronized effort of nearly all business disciplines within a company.

The term Product Creation is not only used for the development of entirely new products, but applies also to the development of variations of existing products or the development of upgrades or add-on products. The implementation of the Product Creation Process can vary, depending on the product being developed; a small add-on product will use a different organization than the development of a large new complex product.

3.3 Phases of the Product Creation Process

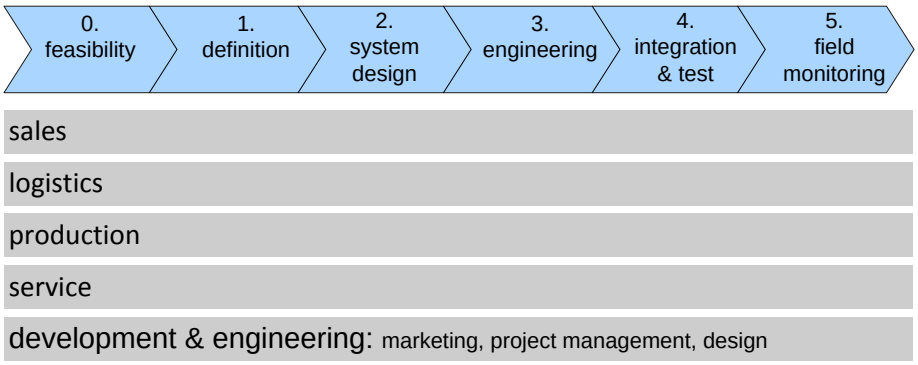


Figure 3.1: A phased approach of the Product Creation Process, showing the participation of all disciplines during the entire process

The Product Creation Process can be structures by using a phased approach. Figure 3.1 shows the phases as used in this book. The figure shows the participation of all business disciplines during this process.

These phases are used across all business functions which have to participate in the Product Creation Process. It is a means to manage the relations between these functions and to synchronize them. Note that sales, production, logistics and service people are involved in the Product Creation Process. Their participation is required to understand the needs of the Customer Oriented Process. A good understanding of these needs is required to develop the new procedures and processes for the customer oriented process, such as ordering, manufacturing, and installation.

Figure 3.2 zooms in on the expected progress for the design deliverables. We use the term work flows for the horizontal classes of activities: *needs analysis*, *product specification*, *design*, *verification and validation*, and *engineering*. Note that needs analysis, product specification, and design progress concurrently. Also note that the first review typically takes place long before any of the work flows is complete. The main question for the first review is: does it make sense to invest in the later phases?

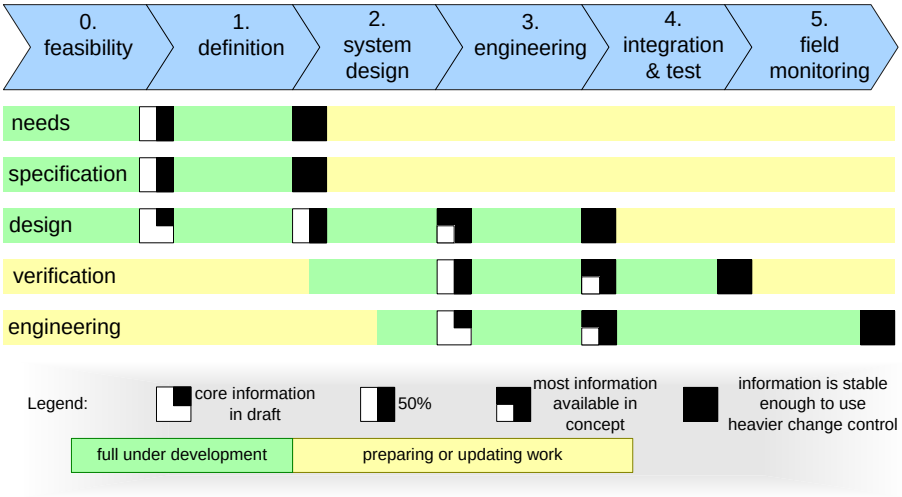


Figure 3.2: A phased approach of the Product Creation Process, showing the progress of the different design deliverables

The advantages of a phased approach are shown in Figure 3.3. The project members get guidelines from the phase model,: *who* does *what* and *when*. At the same time the check lists per phase provide a means to check the progress for the management team. The main risk is the loss of common sense, where project members or management team apply the phase model too dogmatic.

Customization of the phase model to the specific circumstances is always needed. Keep in mind that a phased process is only a means.

The phase process is used as a means for the management team to judge the progress of the Product Creation Process. That can be done by comparing the actual progress with the checklists of the phase model, at the moment of a phase transition. The actual progress is measured at the moment of transition. Normally the development will continue after the phase review, even if some deliverables are behind schedule. In that case the problem is identified, enabling the project team to take corrective action. Some management teams misinterpret the phase transition as a milestone with mandatory deliverables. Based on this misinterpretation the management team might demand full compliance with the checklist, disrupting the

benefits	*disadvantages*
blueprint: how to work	following blueprint blindly
reuse of experience	too bureaucratic
employees know *what* and *when*	transitions treated black and white
reference for management	

Figure 3.3: Advantages and Disadvantages of a phased approach

project. This kind of interference can be very counterproductive. See section 3.5 for a better management method with respect to milestones.

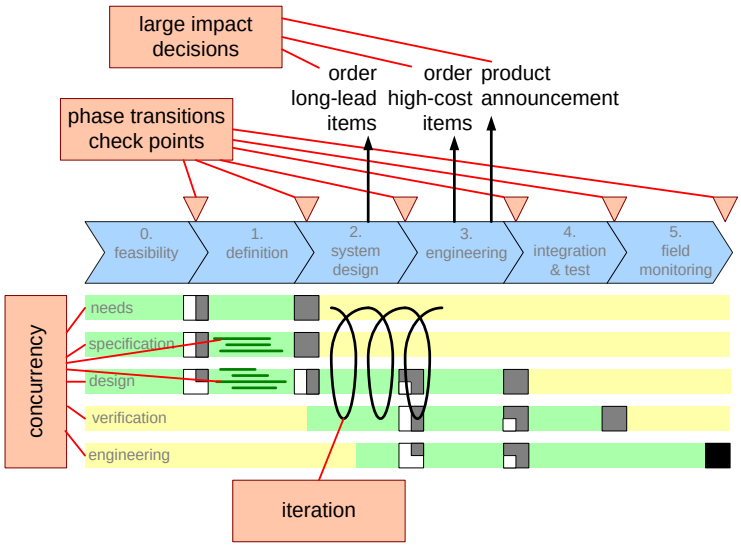


Figure 3.4: Characteristics of a phase model

Important characteristics of a phase model are shown in Figure 3.4:

Concurrency of need analysis, specification, design, and engineering, and concurrency between activities within each of these work flows.

Checkpoints at phase transition. Often more checkpoints are defined, for instance halfway a phase.

Iteration over the work flows and over activities within the work flows.

Large impact decisions that have to be taken, often long before the full consequence of the decisions can be foreseen.

3.4 Evolutionary models for Product Creation

The phase model stresses and supports concurrent activities, see also [8]. A common pitfall is a waterfall interpretation of a phased approach. Following a strict top-down approach can be a very costly mistake, because feedback from implementation and customers is in that case too late in the process. Early and continuous feedback both from implementation as from customer point of view is essential, see Intermezzo 4.

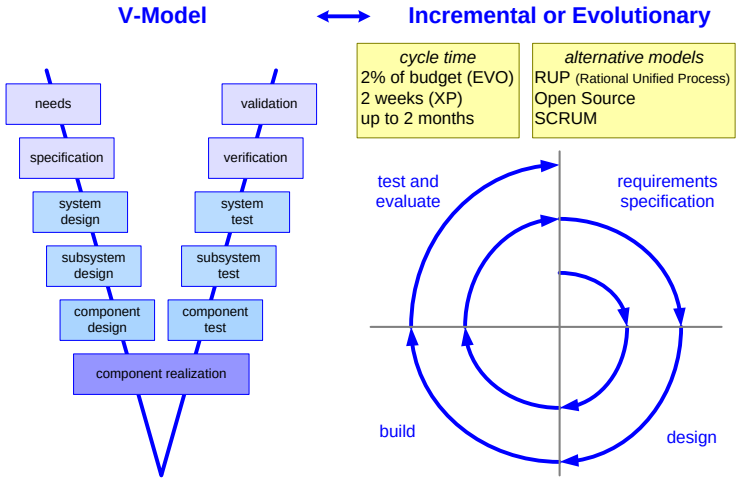


Figure 3.5: V-model versus Incremental or Evolutionary development models

High market dynamics exposes one weakness of the phased approach: market and user feedback becomes available at the end of the creation process. This is a significant problem, because most product creations suffer from large uncertainties in the specifications. Discovering at the end that the specifications are based on wrong assumptions is very costly.

Figure 3.5 show the V-model and evolutionary model side by side. Evolutionary methods focus on early feedback creation. EVO [6] by Gilb recommends to use evolutionary development steps of 2% of the total development budget. In every step some product feedback must be generated. Extreme Programming (XP) [2] by Beck is based on fixed duration cycles of two weeks. XP requires additional customer value in every increment.

The class of agile product creation approaches is struggling with the architecting process. The leaders of these communities dislike the “big design up-front”. However, running in a treadmill of small increments may cause the loss of the “big picture”. Architecting and short cycles, however, are not in conflict. The architecture also has to grow in incremental steps.

3.5 Milestones and Decisions

The project team is faced with a limited number of large impact decisions during the creation process. The decision in general engage the organization with a commitment somewhere in the future. For example:

Ordering of long lead items where changes in specification or design might obsolete ordered items. Re-ordering will cause project delay. Using the initially ordered items might decrease system performance.

Ordering of expensive materials where changes in plan, specification or design might obsolete the ordered materials.

Product announcement can not be reversed once the outside world has seen the announcement. Note that announcing a new product often impacts the order intake of existing products. Announcing a new product late might cause competitive risks.

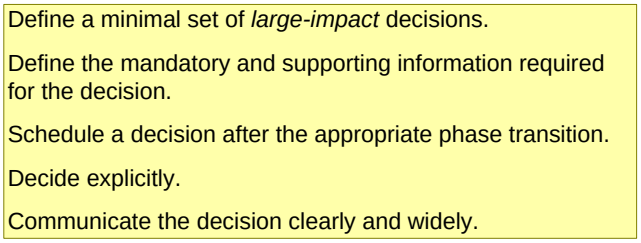


Figure 3.6: How to deal with large impact decisions

An explicit decision can be planned as a milestone in the project master plan. Information should be available to facilitate the decision: some of the information is mandatory to safeguard the company, some of the information is only supportive. In general the mandatory information should be minimized to prevent a rigid and bureaucratic process, causing the company to be unresponsive to the outside world. These decisions can be planned after the phase transition when most of the required supportive information will be available in an accessible way. Figure 3.6 shows the recommendations how to deal with large impact decisions.

3.6 Organization of the Product Creation Process

The Product Creation Process requires an organizational framework. The organizational framework of the Product Creation Process is independent of the Organizational frameworks of the other processes[1]

[1] Quite often a strong link is present between People and Technology Management Process and the Product Creation Process; Using similar frameworks can be quite counterproductive, because

3.6.1 Hierarchical decomposition

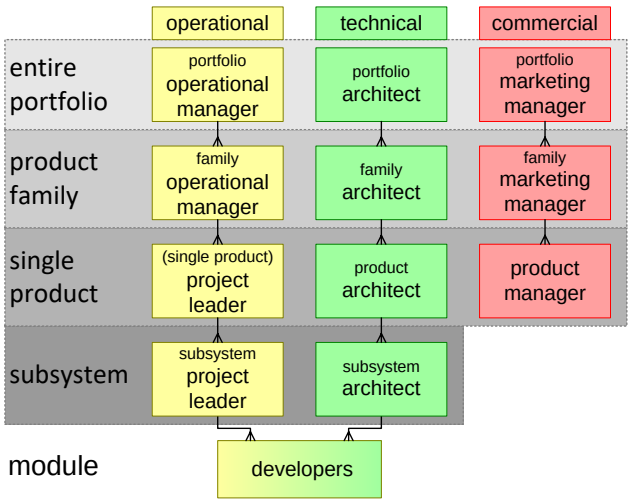


Figure 3.7: The simplified hierarchy of operational entities in the Product Creation Process form the core of this process.

The operational organization is a dominant organizational view on the Product Creation Process. In most organizations the operations of the Product Creation are decomposed in multiple hierarchical levels, at the highest level the entire product portfolio at the lowest level the smallest operational entity for instance a subsystem. Note that in figure 3.7 the hierarchy stops at subsystem level, although for large developments it can continue into even smaller entities like components or modules. The hierarchy is simply the recursive application of the decomposition principle.

Figure 3.7 is simplified by assuming that a straight forward decomposition can be applied. This assumption is not valid when lower level entities, e.g. subsystems, are used by multiple higher level entities, e.g. systems. For instance, if one subsystem is used in different products. In Chapter 23 we elaborate this aspect further.

3.6.2 Further decomposition of the Product Creation Process

The Product Creation Process can be decomposed in 3 processes as shown in 3.8:

Marketing: Defining how to obtain a sellable profitable product, starting with listening to customers, followed by managing the customer expectations, introducing the product at the customer and obtaining customer feedback.

these processes have quite different aims and characteristics. Of course, nearly all people are part of both organizational frameworks.

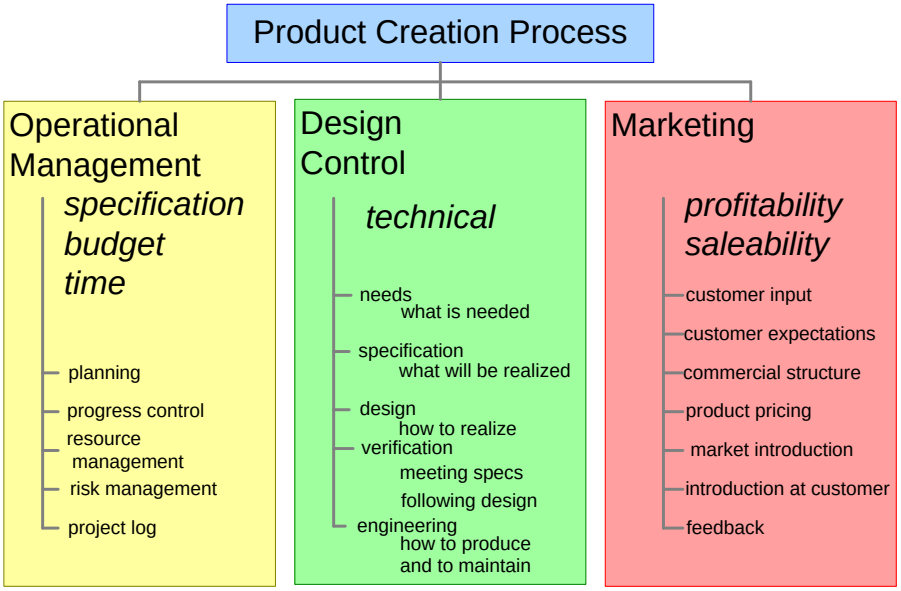


Figure 3.8: Decomposition of the Product Creation Process

Project Management: Realizing the product in the agreed triangle of

- specification
- resources
- amount of time

Design Control: Specifying and designing the system. The Design Control Process is that part of the Product Creation Process that is close to the conventional R&D activities. It is the content part of the Product Creation Process.

The functions mentioned in figure 3.7 map directly on the processes in figure 3.8:

- The *operational* or *project leader* is responsible for the *operational management*
- The *architect* is responsible for the *design control*
- The *marketing* or *product manager* is responsible for the *commercial* aspects

3.6.3 Design Control

The ISO 9000 standard has a number of requirements with respect to the *design control* process. The design control process is a core content oriented process, it is the home base of the system architect. The system architect will support the project management and the commercial process.

The design control process itself is further decomposed, also shown in figure 3.8:

- Needs
- Specification
- Design
- Engineering
- Verification

The needs express what the stakeholders of the system need, not yet constrained by business or technical considerations. Most development engineers tend to forget the original needs after several iterations of commercial and technical trade-offs.

The specification describes what will be realized, in terms of functionality and performance. This specification is the agreement with all stakeholders. The difference between the needs and the specification is that in the specification all trade-offs have been made. See also Chapter 15 where we elaborate the process of needs analysis and requirements management.

The design is the description how the specification will be realized. For instance, the physical and functional decomposition and the budgets for critical technical resources belong to the design.

Needs, specification and design are documented in development documents. The main function of these documents is to streamline the Product Creation Process. During this process these are living documents fulfilling an important communication function, while at the same time they play an important role in the control aspect of the design process.

The verification process verifies that the implementation meets the specification in the way it is specified in the design.

The engineering process provides the foundation upon which the Customer Oriented Process works for the entire life-cycle of the product. The documentation generated in the engineering process is the output of the Product Creation Process.

3.6.4 Operational Management

The operational management is governed by a simple set of rules, see Figure 3.9. These rules combine a number of very tightly coupled responsibilities in one function, to enable a dynamic balancing act by the operational leader. These responsibilities form the operational triangle as shown in figure 3.10.

The rules ensure that the operational leader takes ownership of the timely delivery of the specification within the agreed budget, with the “standard” quality level. Transfer of one of these responsibilities to another person change the system in an open loop system[2].

[2] Many conventional development organizations have severe problems with this aspect. The

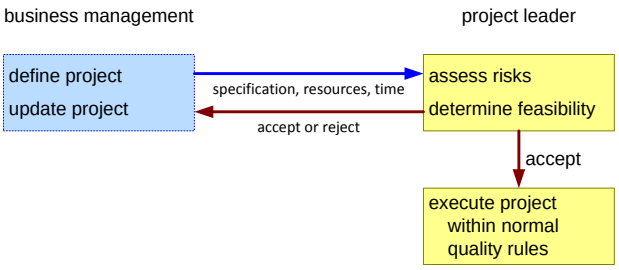


Figure 3.9: Commitment of the operational leader to the project charter

Figure 3.10: The Operational Triangle of responsibilities; The operational leader commits to the timely delivery of the specification within the agreed budget, with the "standard" quality level

3.6.5 Marketing

The marketing manager knows the market: who are potential customers, what are their needs, what is of value in the market, what are commercial partners, what is the competition. This knowledge is “future” oriented and is used to make choices for future products. What are feasible products, what are the features and performance figures for these products, based on choices where value and cost are in a healthy balance. Hence the marketing manager is involved in packaging and pricing of products and options. A good marketing manager looks broader than the current products. Most innovations are not “more of the same”, but are derived from new opportunities, technical or in the application.

Note that most sales managers are much more backward oriented: we sell what we have to customers who know existing systems. Good sales persons are often not good marketing persons!

---

most common mistake is that either the quality responsibility or the resource(budget) responsibility is transferred to the People and Technology Management Process. The effect is that excuses are present for every deviation of the commitment, for instance *I missed the timing because the people were working on non project activities.*

3.6.6 Product Creation Teams

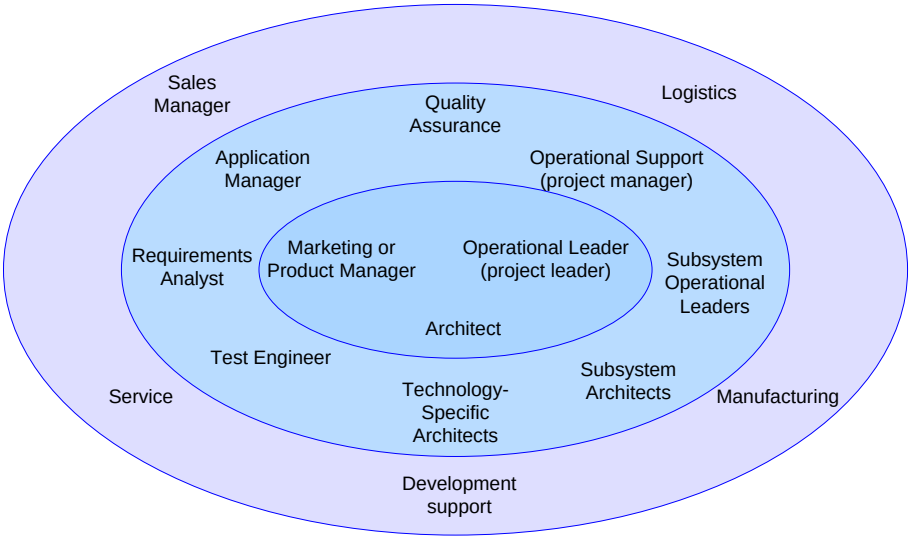


Figure 3.11: The operational teams managing the Product Creation Process

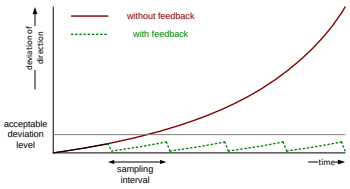
So far we have discussed *Operational management*, *Design Control* and *Marketing*. However, in the Product Creation Process more specialized functions can be present. Figure 3.11 shows a number of more specialized functions as part of a number of concentric operational teams. The amount of specialization depends on the size of the operation. In very small developments none of the specializations exist and is even the role of project leader and architect combined in a single person.

3.7 Acknowledgements

Rahim Munna suggested to add a short description of Marketing.

Chapter 4

The Importance of Feedback for Architecture



4.1 Introduction

Feedback is a universal principle that is applied in highly technical domains such as control engineering, but also in social sciences. This Intermezzo discusses feedback as part of the Systems Architecting Process and explains its importance.

4.2 Why Feedback?

4.2.1 Control

Feedback is used in control systems to ensure that the actual direction corresponds to the desired direction. In general the deviation from the desired direction grows exponentially in time, see Figure 4.1.

Many control systems implement a feedback loop to force the system back in the desired direction. Figure 4.1 also shows the effect of a discrete feedback system over time. It will be clear that the sampling interval is determined by the time constant of the deviation and the acceptable deviation level.

Product development can be seen as an ordinary system that can be controlled analog to technical control systems. Product developments without feedback result

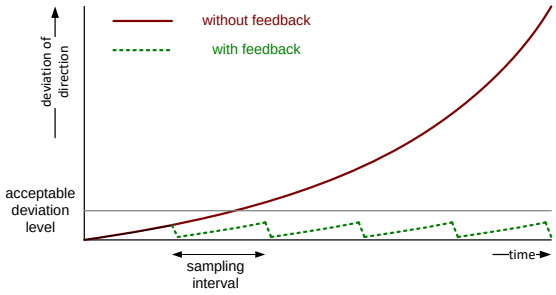


Figure 4.1: The deviation of the actual direction of product development with respect to the desired direction as function of the time

in products that are out of specification (too late, too slow, too expensive, too heavy et cetera). Sound development processes contain (often multiple) feedback loops.

4.2.2 Learning

Human beings learn from their mistakes, *provided that they are aware of them*. Feedback is the starting point of the learning process, because it provides the detection of mistakes. Efficiency of individuals and organizations can be increased by learning. Without learning similar mistakes are repeated: a waste of resources.

4.2.3 Applicability

The principle of feedback can be applied on **any** activity. The higher the uncertainty or the larger the duration of an activity is, the more important feedback becomes.

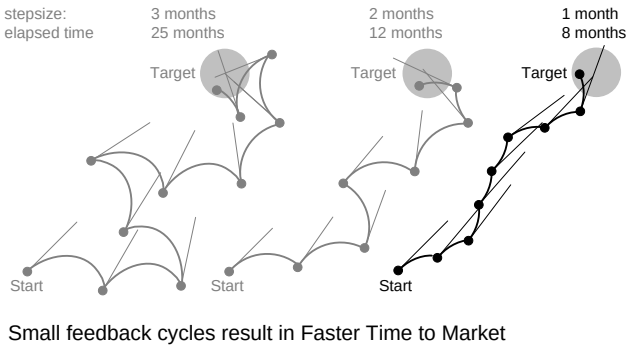


Figure 4.2: Example with different feedback cycles (1, 2, and 3 months) showing the time to market decrease with shorter feedback cycles

Figure 4.2 shows an example of a development with three different feedback cycle times, respectively three, two, and one months. The three month feedback cycle results in an project duration of 25 months. Decreasing the feedback cycle to 2 months brings the total project duration down to 12 months. One month feedback cycles give a total time of only 8 months. This simple model ignores the cost of obtaining feedback, but it clearly illustrates the essence of short feedback cycles.

4.3 Theory versus Practice

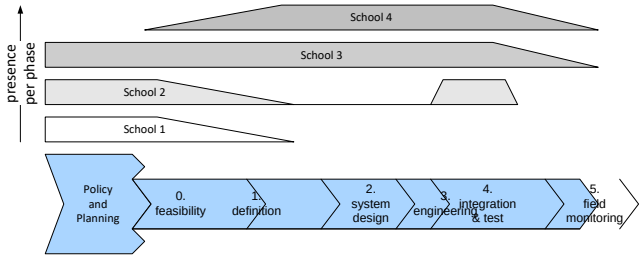


Figure 4.3: Four different schools of architecting, showing the presence of the architect in relation to the policy and planning process and the product creation process

Systems architecting is partially a very conceptual activity. The concepts are theoretical as long as they are part of presentations or specifications. Some architecting schools promote the system architecting function as strategic, providing direction, without being drowned in operational problems. A second school promotes an architect who is active in the definition phase of a product as well as in the verification phase. We argue a third direction: architecting has to be done during the entire development life cycle. In practice many architects function still in a fourth way: entirely in the technical domain. Figure 4.3 visualizes the 4 different schools as function of the process phase.

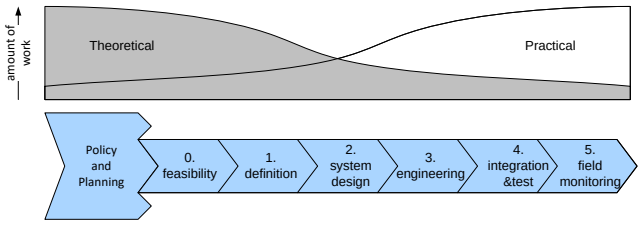


Figure 4.4: Theoretical versus Practical system architecture work in relation to the development life cycle

Figure 4.4 shows the amount of "theoretical" work and the amount of "practical" work also as function of the process phase. Where we use the term "theoretical" for concepts in presentations or specifications that have not been exposed to the physical world. Similarly, "practical" is used for work where the design is realized and tested.

A number of feedback loops can be closed during the Product Creation Process. Normally the next phase in the process provides feedback to the previous phase in the process. This phase transition feedback is often applied. However, feedback from the next phase is a rather indirect measure for the desired direction. The next step provides feedback on the usefulness of the input to continue the work, but the user satisfaction and market success can not be measured by the next step.

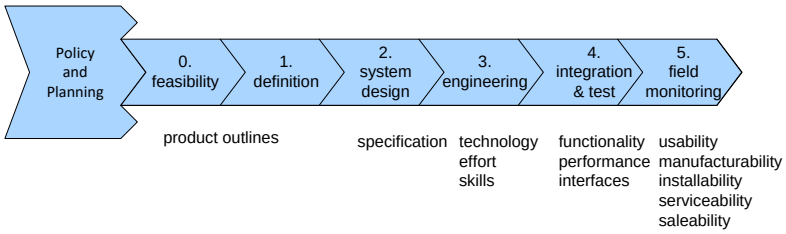


Figure 4.5: Feedback per development phase

The feedback for theoretical work comes from the practical work. Figure 4.5 shows the feedback per development phase. This figure makes it immediately clear that the amount of feedback is proportional to the amount of practical work going on.

4.4 Conclusions

The conclusions of this paper are given here as a set of position statements:

1. For the education of system architects it is essential that they participate in the entire feedback loop.

2. The education of system architects is never finished.

3. System architects must participate in the entire product creation lifecycle for most of their carrier.

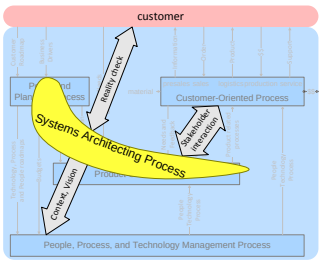
4. The value of system architects in the policy and planning process stems from the practical feedback during the product creation process.

5. Feedback can never come too early.

6. System architects can have fantastic dreams, feedback is required to prevent that dreams turn into a nightmares.

Chapter 5

The System Architecture Process



5.1 Introduction

This chapter positions the systems architecting process in a wider business scope. This positioning is intended to help understanding the process itself and the role of the system architect (or team of system architects).

We focus on systems architecting within organizations that create and build systems consisting of hardware and software. Although other product areas such as solution providers, services, courseware et cetera also need system architects, the process structure will deviate from the structure as presented here. See Intermezzo 6 for an elaboration of these other architecting models.

5.2 System Architecture in the Business Context

Figure 5.1 shows the main activities of the System Architecting Process as an overlay over the business decomposition.

Processes are goal oriented, as discussed in Intermezzo 2. The process decomposition is not orthogonal, several processes are overlapping. The System Architecting Process is a clear example of such non-orthogonality. Figure 5.2 shows a map of the System Architecture Process and neighboring processes. Many processes,

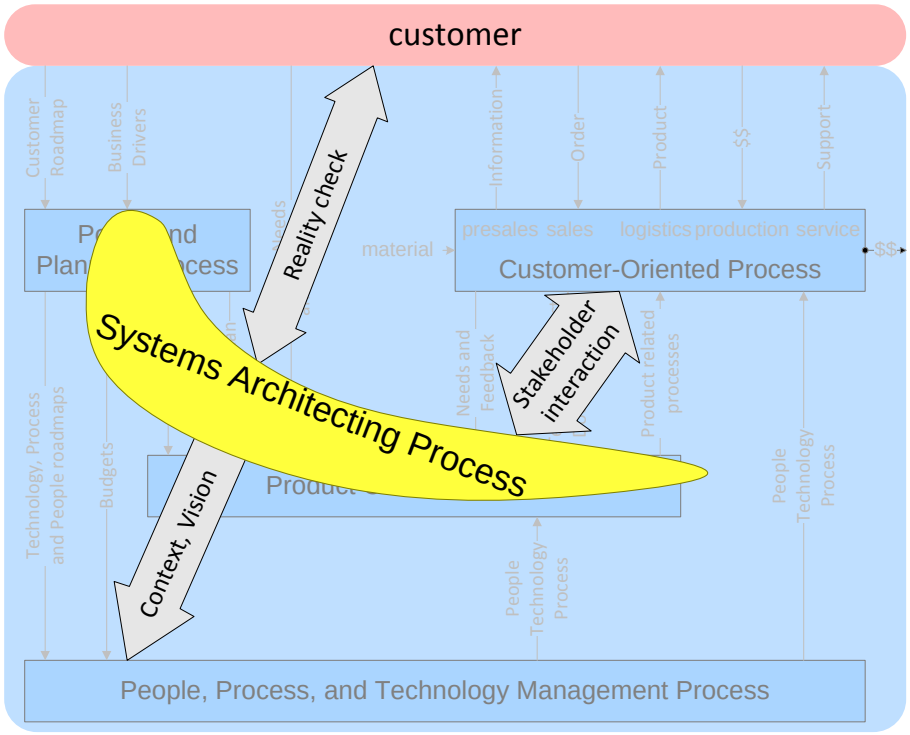


Figure 5.1: The main System Architecture activities in the Business Context

such as manufacturing engineering, service engineering, have been left out of the map, although these processes also have a high architecture relevance.

Both figures make it clear that the System Architecting Process contributes heavily to the Product Creation Process, while it plays also an essential role in the Policy and Planning Process. Both contributions are strongly coupled, see figure 5.3

The System Architecture Process bridges the gap between Product Creation Process and the Policy and Planning Process. In many organizations this link is missing. The absence of this link results in:

- re-inventing a (different) product positioning during the Product Creation Process, with a limited context view
- policies which are severely handicapped by a lack of practicality or realism

The overview created by the System Architecting Process also helps in establishing a technology policy.

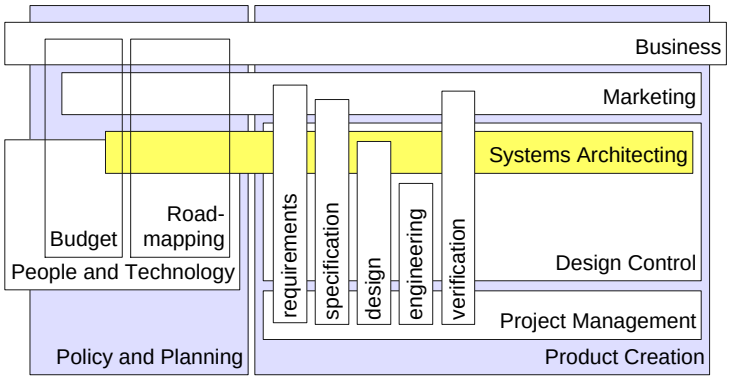


Figure 5.2: Map of the System Architecture Process and neighboring processes

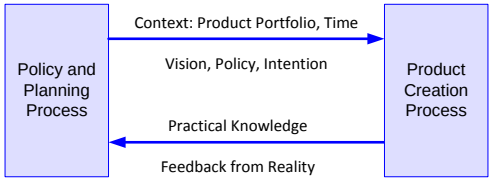


Figure 5.3: Contribution of System Architecting to the the Coupling of Policy and Planning Process and the Product Creation Process

5.3 Purpose of the System Architecting Process

Every business exceeding a few people enables the efficient concurrent work of these people by dividing the tasks in smaller more specialized jobs, the *decomposition principle* in action. This decomposition of responsibilities requires an opposing force integrating the activities in a useful overall business result. Several integrating processes are active in parallel, such as project management, commercial management et cetera.

The System Architecting Process is responsible for:

- the Integral Technical aspects of the Product Creation Process, from requirement to deployment.
- the Integral Technical Vision and Synergy in the Policy and Planning Process.

The System Architecting Process is striving for an optimal overall business result, by creating and maintaining the key issues, such as a balanced and consistent design, selection of the least complex solution, and satisfaction of the stakeholders.

The System Architecture Process is balancing amongst others:

- External and internal requirements

- Short term needs and long term interests
- Efforts and risks from requirements to verification
- Mutual influence of detailed designs
- Value and costs

Such a balance is obtained by making trade-offs, for example *performance* versus *qualities* versus *functionality*, or *synergy* versus *specific solution*

It is the purpose of the System Architecting Process to maintain the consistency throughout the entire system, from roadmap and requirement to implementation and verification. On top of this consistency the integrity in time must be ensured.

An enabling factor for an optimal result is *simplicity* of all technical aspects. Any unnecessary complexity is a risk for the final result and lowers the overall efficiency.

5.4 The System Architect as Process Owner

The owner of the System Architecting Process is the System Architect or the System Architecting Team. Many other people are involved in the System Architecting Process.

The system architect or the team members spent the majority of their time, about 80%, in the Product Creation Process. From the remaining time the majority is spent in the Policy and Planning Process. In 5.2 it is explained that these processes are strongly coupled. This coupling is for a large part implemented by employing the same people in both processes. A small amount of time is spent in People, Process, and Technology Management.

5.5 System Architecting in Product Creation Context

The Systems Architecting Process is striving for consistency and balance from requirement to actual product.

The amount of people working in product creation can vary from a few to tens of thousands of people. All people working on the creation of a new product have only knowledge of a (small) subset of the information. Inconsistencies and local optimal solutions pop up all the time, caused by lack of knowledge of the broader context.

The Systems Architecting Process has to prevent this natural degradation of the system quality. Systems Architecting acts pro-active by clear and sharp requirements, specification and system design as well as reactive by following up the feedback from detailed design, implementation and test.

During the Product Creation Process many specification and design decisions are taken. Quite often these decisions are taken within the scope of that moment.

Consecutive decisions can be in contradiction with previous decisions. For instance, a decision is taken to add memory to the product to increase performance, while one month later the amount of memory is decreased to lower the cost. The Systems Architecting Process maintains the integrity over time, by looking at decisions from a broader perspective.

5.6 Acknowledgements

Discussions with and critical comments from Rard de Leeuw, Jürgen Müller, Henk Obbink, Ben Pronk and Jan Statius Muller helped to shape, to improve the structure and to sharpen the contents of the article "Positioning the System Architecture Process". This article is based on the last sections of this article. I am grateful for their contribution.

Jürgen Müller spotted hiccups in the flow of the new article, enabling a streamlining and extension of this article. Robert Deckers analyzed the text and pointed out many inconsistencies and poor formulations.

An inspiring presentation by Bud Lawson helped me to make a more complete and balanced list of System Architecture key issues.

Chapter 6

Products, Projects, and Services; similarities and differences in architecting

logo
TBD

6.1 Introduction

We have focused on the product creation of "box" like products: products that have a clear physical part; the product is a box that is created and the products are sold as boxes by sales. In the twentieth century this was one of the dominating models in industry. Another business model is *project* delivery: customers order a turn-key solution to be delivered by the supplier.

At the end of the century, several other types of systems and related business models became increasingly important. An increase of interoperating systems has opened a world of *services*, e.g. traffic information for navigation systems. *Services* are also systems, but these systems tend to be less tangible, while these service systems often include people, processes and organizations.

Similarly, System of Systems emerge everywhere. We have become dependent on the interoperation of multiple systems, the system of systems.

6.2 Products and Projects

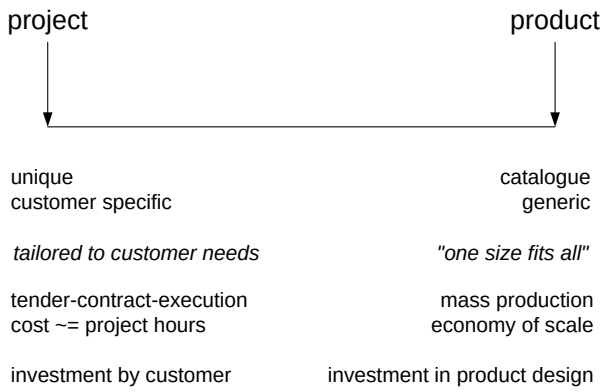


Figure 6.1: Projects versus Products

Figure 6.1 shows an axis with on the left hand extreme *projects* and on the right hand extreme *products*. We can characterize the extremes as:

Projects are unique for a specific customer. The solution is tailored to the customer needs. The sales starts with a tender phase, the execution phase starts when the contract has been signed. Cost is typically proportional with the number of project hours. In project business the customer is the investing party and carries most of the risk.

Products are standardized as part of the sales catalogue. Products are designed to be generic, i.e. to serve multiple customers. The standardization in extremis assumes that "one size fits all". At the same time standardization enables mass production, while the increased volume of multiple customers provides an *economy of scale*. Product companies typically invest themselves in new product designs

In practice business models are less black and white. Figure 6.2 shows a number of forces that lead to convergence between these two extremes. Project organizations see opportunities to increase their margin by harvesting and re-using standardized components or products. Product organizations adapt their standard products more to specific customer needs by making their products customizable and configurable. Customer support can adapt the product at the customer site to customer specific needs.

Figure 6.3 shows a simplified process diagram for project business. The Customer Oriented Process is replace by a triplet of processes:

Tender process where the specification and price are negotiated with the customer.

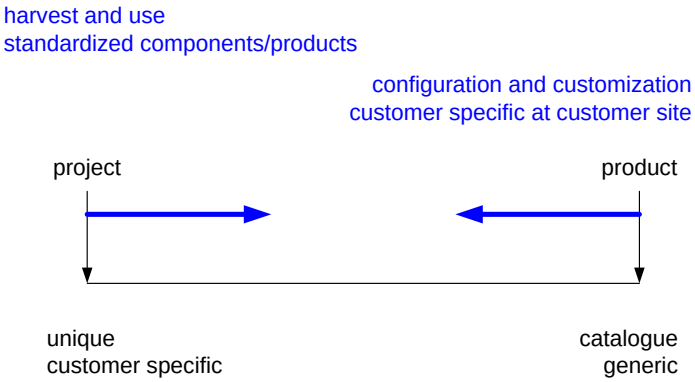


Figure 6.2: Convergence of Projects and Products

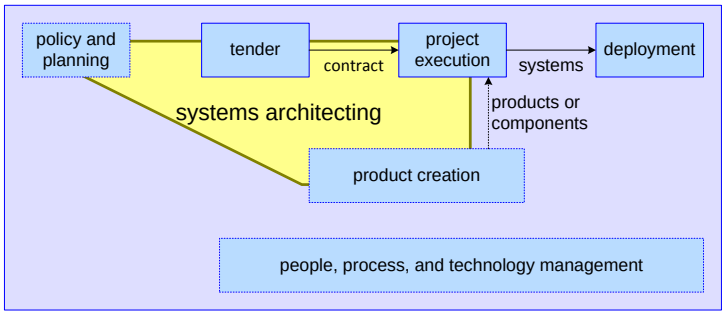


Figure 6.3: Simplified process diagram for project business

Execution process where the solution is created.

Deployment process where the systems are installed at the customer site and the operation is started.

6.3 Services

Figure 6.4 shows an example of a smart phone context. The smart phone as device contains hardware, operating system and software. The device offers an application infrastructure for many applications that are created by many different parties. The application creation probably will be supported by tools.

The applications on the device and telecom services facilitate content services in the broader world. E.g. a location service based on position, map, and directory information.

Device builders have to cooperate with the telecom world and the content world

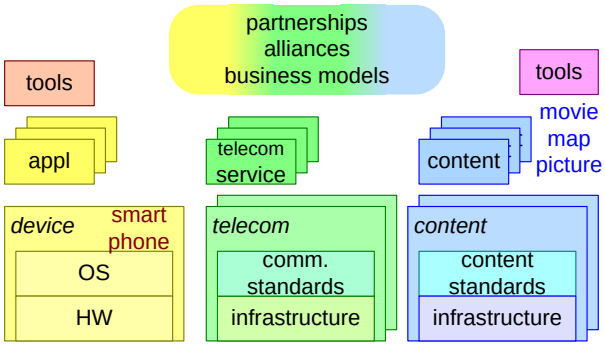


Figure 6.4: Example of extensive complex of services for smart phone type of device

to create a sellable device. Developing telecom services and developing content services can also be seen as the creation of systems. However, this world has many less technical aspects. Forging and nurturing partnerships and alliances is crucial, as well as the development of business models.

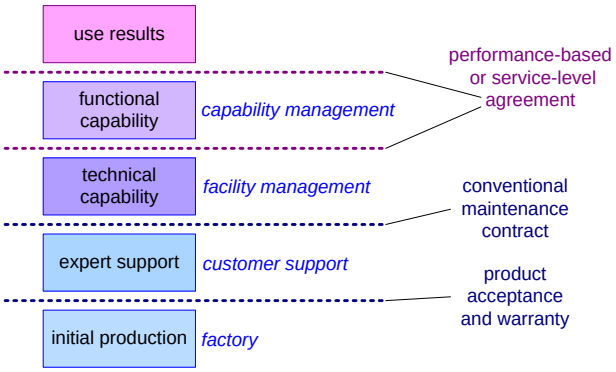


Figure 6.5: Model of operational services showing that the boundary between provider and customer can be defined at different levels

The type of deliverable and the related business model is also shifting. The conventional model is that the supplier delivers a product according to specification. The relation with the customer stops once he product has been delivered. In many business to business segments the relation is extended by offering maintenance contracts. However, in the conventional model, the customer takes ownership of the system. The bottom two layers in Figure 6.5 represent the conventional business models.

In business to business situations the system that is delivered will be managed

by a facilitation or technical department. E.g. in hospitals the radiology equipment is supported by technical hospital staff. The actual operation of the system is done by application experts, in the hospital example the radiology equipment is run by dedicated clinical staff and radiologists. The radiology department provides an imaging and diagnosis capability to the referring physicians.

The equipment manufacturer can shift their support "upwards" to offer:

Facility management a technical working and prepared system.

Capability management where the whole capability, such as diagnostic imaging, is offered.

The consequence of this shift is that the supplier creates a recurring revenue stream. The integral consequence for customer and supplier is that incentives are changing.

For example, when the supplier is responsible for a constant performance, then the supplier might decide to upgrade the equipment much more regular. The supplier also gets an incentive to minimize down time and maintenance costs.

The process structure might be adapted to facilitate the service development. Service development, both for the content type as well as for the operational type, require many less technical, more political, social, and economical development activities.

6.4 System of Systems

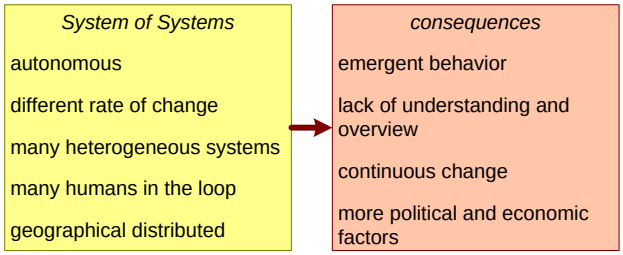


Figure 6.6: System of Systems and the consequences of this approach

Today's society depends heavily on the interoperability of many systems. We recognize that the solution can be created by interoperability of multiple systems, the so called *System of Systems*. See Figure 6.6 for the characteristics of System of Systems and the consequences of this approach. The System of Systems can be seen as a super system.

Examples of system of systems are:

Military capabilities , where amongst others planes, tanks, guns, officers, soldiers, and sensors are interconnected.

Health care treatment room , e.g. operating theater or catherization laboratory, where respiratory and physiology monitors, surgical tools, clinical support systems, nurses, surgeons, et cetera collectively perform the treatment function.

The individual systems in a System of Systems can operate autonomously. Most often these systems have not been created with this specific super-system in mind. The individual systems follow their own life cycles, with different rates of change. The systems can be quite heterogeneous (large, small, expensive, low cost, re-usable, disposable, fragile, robust, et cetera). Every system has its own human machine interface and its own control paradigm. The geographical location of the systems can be distributed and may change.

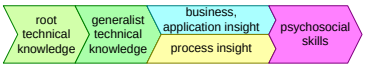
These characteristics have several consequences. The most dominant consequence is that the super system is so complex that nobody has the understanding and the overview of the whole. Hence nobody can predict what will happen and we get so-called *emergent behavior*. The amount of systems with their different change rates and the amount of humans create a super system that is never exactly the same: it changes continuously. In the larger scope of the System of Systems many non technical factors play a role, e.g. economical or political.

Part II

The System Architect as a Person

Chapter 7

The Awakening of a System Architect



7.1 Introduction

System architects are very rare commodity. This chapter describes the observed general growth pattern of system architects. We hope that by analysis of the the characteristics of existing system architects will facilitate the training of new system architects. Reference [21] contains a good description of a system architect.

7.2 The Development of a System Architect

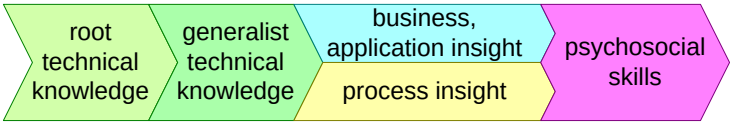


Figure 7.1: Typical Development of a System Architect

System architects need a wide range of knowledge, skills and experience to be effective. Figure 7.1 shows a typical development of a system architect.

The system architect is rooted in technology. A thorough understanding of a single technological subject is an essential underpinning. The next step is a

broadening of the technical scope. Section 7.3 describes the path from a mono-disciplinary specialist to a multi-disciplinary system architect with broad technological knowledge.

When the awakening system architect has reached technological breadth, then it will become obvious that most encountered problems have a root cause outside of technology. The system architect starts to develop along two main parallel streams:

The business side: the market, customers, value, competition, logistics, service aspects

The process side: who is doing what and why, necessitated by the amount of involved stakeholders

During this phase the system architect will broaden in these two dimensions. The system architect will view these dimensions from a technological perspective. Again when a sufficient level of understanding is attained an awareness starts to grow that people behave much less rationally than technical designs. The growing awareness of the psychological and the sociological aspects is the next phase of growth.

7.3 Generalist versus Specialist

Most developers of complex high tech products are specialists. They need an in-depth understanding of the applicable technology to effectively guide the product development. The decomposition of the development work is most often optimized to create a work breakdown enabling these specialists to do their work with as much autonomy as possible.

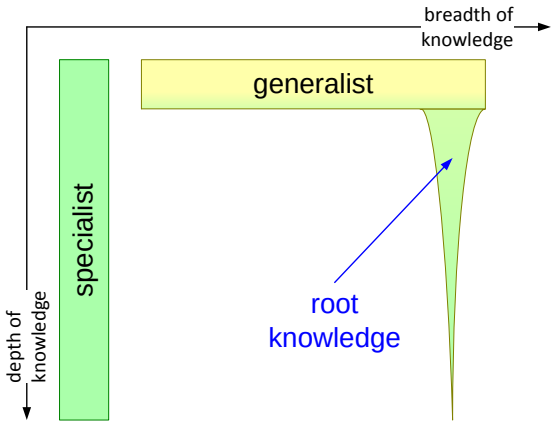


Figure 7.2: Generalist versus Specialist; depth versus breadth

Figure 7.2 is a visualization of the difference between a specialist and a generalist. Most generalists are constrained in the depth of their knowledge by normal human limitations, such as the amount of available time and the finite capacity of the human mind. The figure also shows that a generalist has somewhere roots in detailed technical knowledge. These roots are important for the generalist self, since it provides an anchor and a frame of reference. It is also vital in the communication with other specialists, because it gives the generalist credibility.

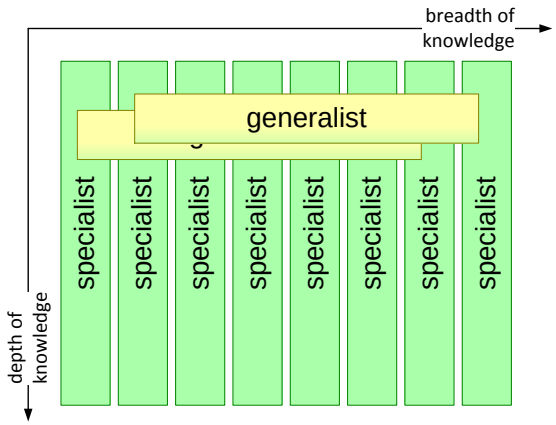


Figure 7.3: Generalists and Specialists are both needed in complex products, they have complementary expertise

Figure 7.3 shows that both generalists and specialists are needed. Specialists are needed for their in depth knowledge, while the generalists are needed for their general integrating ability. Normally there are much more specialists required than generalists.

There are more functions in the Product Creation Process that benefit from a generalist profile. For instance the functions of project-leader or tester both require a broad area of know how.

Architects require a generalist profile, since one of their primary functions is to generate the top-level specification and design of the system. The step from a specialist to a generalist is of course not a binary transition. Figure 7.4 shows a more gradual spectrum from specialist to system architect. The arrows show that intermediate functions exist in larger product developments, forming natural stepping stones for the awakening architect.

Examples of aspect architects are:

subsystem architects subsystems are the main organizational decomposition. In hardware intensive systems subsystems tend to be physical, e.g. loader or generator. Typical number of subsystems is between 5 and 15.

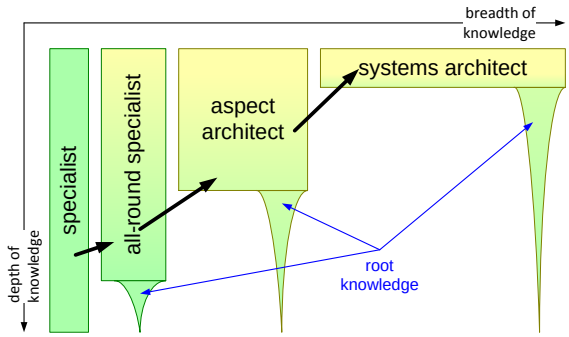


Figure 7.4: Growth in technical breadth, intermediate functions from specialist to system architect

SW, mechanics or electronics architects or discipline oriented architects. The architects ensure consistency across physical subsystems

function architects take responsibility for one system function, ensuring the soundness of that function.

quality architects take responsibility for one quality, e.g. safety, reliability, security.

For instance a software architect needs a significant in-depth knowledge of software engineering and technologies, in order to design the software architecture of the entire system. On the other hand a subsystem architect requires multi-disciplinary knowledge. The limited scope of one subsystem reduces the required breadth for the subsystem architect to a hopefully realistic level.

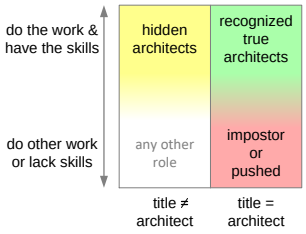
Many products are becoming so complex that a single architect is not capable of covering the entire breadth of the required detailed knowledge areas. In those cases a team of architects is required, where the architects are complementing each other in knowledge and skills. It is recommended that those architects have complementary roots as well; as this will improve the credibility of the team of architects.

7.4 Acknowledgements

Chuck Kilmer suggested a new title and offered many textual improvements.

Chapter 8

Systems Titles and Roles



8.1 Introduction

The following questions are asked frequently during and after the courses:

- What is the difference between *systems engineers* and *systems architects*?
- Why do all these people have the title systems architect, while they actually don't do the work?

The first questions is also posed in other variants, using titles such as *system designer* or *systems manager*. To complicate matters more there are people who do part of the systems level work, for example *requirements analyst*, *systems analyst*, *system integrator* or *system tester*, complementing the systems architect.

8.2 Cultural differences in terms

Exactly the same titles are used differently in different companies (or even divisions or product groups within one company), in different domains (e.g. defense, automotive, consumer electronics, IT), and geographic regions. No single unified standardized definition is used across companies, domains, and geographies. We do recommend

to *calibrate* terminology when entering new territory, and to be continuously alert for differences in interpretation even after calibration.

Throughout this book we use the term architecture for the combination of two crucial aspects:

Depth understanding of the system-of-interest including product specification, decomposition in subsystems and components, interface management, and function and resource allocation, to create a sound and fitting system that fulfills all qualities (e.g. safety, reliability, performance).

Breadth understanding of the context including the customer context and the stakeholders in the value chain , and the life cycle context from conception to decommissioning and all related business aspects.

Be aware that the term architect is used often for the *system-of-interest* part only. We use the term *system design* for this subset of architecting work.

A major professional society in the *systems* world is INCOSE, the International Council of Systems Engineering. The *Systems Engineer* as depicted by most of INCOSE documents is a very broad function, including work of the project leader, the requirements analyst, the systems architect, the configuration manager, and quality assurance.

Another extreme for the definition of systems engineer was in the medical domain, where this job of the systems engineer was solely the electro-mechanical design of cables and cabinets.

8.3 Title versus skills and actual job

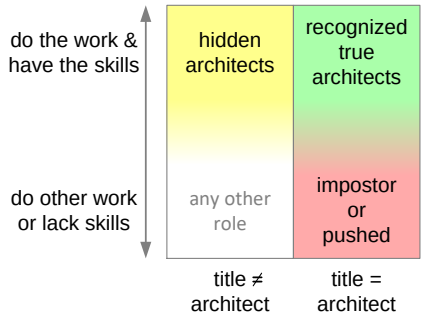


Figure 8.1: Four quadrants to classify architect and titles

First of all we have to distinguish what role or function someone performs and the title that is being used by the people in the context. Figure 8.1 shows the

four quadrants that you get by using title as horizontal axis and competence level as vertical axis. Note that the title axis is discrete, while the competence level is continuous. The figure shows four quadrants:

Any other role (bottom left) for these persons that don't do the job of an architect and do not have the title.

Imposters or pushed persons (bottom right) are people lacking the skills or actually not doing the work of an architect that nevertheless have the title architect. Note that impostors are these people that actually pursued the title, for example because of status or payment. Another category are those people that are pushed by their management into this job, but lack the capability to do it. People do not become true architects by declaration.

Hidden architects (top left) can be found in many organizations. These organizations might use different titles or they might not be aware of the *systems* discipline.

Recognized true architects (top right) are these architects that actually do the architecting job skillfully and got he title in recognition.

8.4 Systems roles and titles

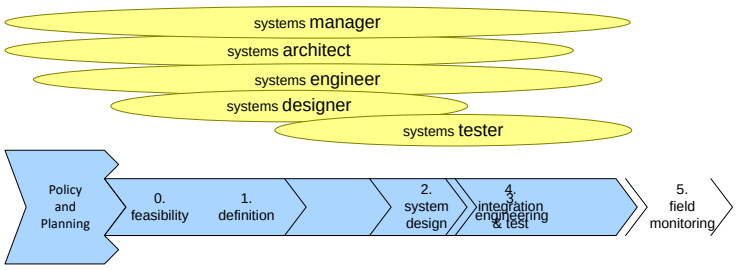


Figure 8.2: System Roles mapped on the development life cycle

In this section we provide a set of roles and relate these to the development life cycles. As explained in the previous sections these roles can be allocated in different ways and different terms can be used than shown here. However, the conceptual roles as shown here are quite universal.

Figure 8.2 shows the following roles:

Systems manager is the overall responsible for all systems aspects, ranging from strategically positioning in the portfolio and the time to final operational performance in the field. Note that such broad definition does not leave

much room for in-depth understanding. An alternate term for this role can be *program manager*.

Systems architect who combines understanding of the context with in depth understanding of the solution to create an appropriate system. Note that the architect role combines some perceptive and creative modes of operation with more analytical modes. This mixture limits how far the architect can go in the real engineering.

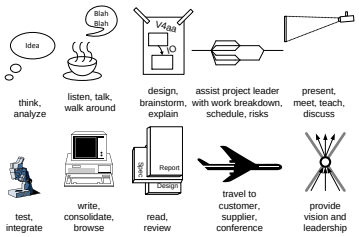
Systems engineer is very close to the systems architect, but the emphasis shifts from perceptive and creative more to engineering. With engineering we mean the capability to finalize and document all details required for the later processes such as logistics, manufacturing, sales, and customer support. Note that also the systems engineer has limits and will depend on specialized engineers (e.g. mechanical, electrical, or software) to finish the last details of the technical product documentation.

Systems designers take the product specification as starting point and work on (potential) solutions. System designers are "inward" focused, where system architects connect the *outward* and *inward* perspectives.

System testers verify that the solution performs as specified. In practice system testers need also trouble shooting capabilities to diagnose the cause of lacking performance.

Chapter 9

The Role and Task of the System Architect



9.1 Introduction

Architects and organizations are often struggling with the role of the system architect (or software architect or any other kind of architect). This struggle is partially caused by the intangible nature of the responsibilities of the architect. At the other hand (good) architects are highly appreciated, even if their quantifiable output is low.

This article starts with specific deliverables, then discusses the more abstract responsibilities and, finally, discusses the day to day activities of an architect.

The role of the software architect is nicely discussed in [3].

9.2 Deliverables of the System Architect

We start at looking for the tangible output that is expected from architects. Project leaders and program managers do expect deliverables to be finished at appropriate milestones. Most Product Creation Processes define the deliverables of a System Architect to be artifacts such as documents or models. These artifacts are symbolized by the stack in Figure 9.1.

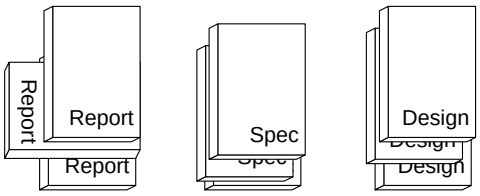


Figure 9.1: Deliverables of a system architect consists of artifacts forming a stack of paper when printed

Figure 9.2 shows the main deliverables of a System Architect more specific. Quite often the System Architect does not even produce all deliverables mentioned here, but the architect does take the responsibility for these deliverables by coordinating and integrating contributions of others. Note that some of these deliverables are part of the Policy and Planning Process.

Customer and Life-Cycle Needs *(what is needed)*

System Specification *(what will be realized)*

Design Specification *(how the system will be realized)*

Verification Specification *(how the system will be verified)*

Verification Report *(the result of the verification)*

Feasibility Report *(the results of a feasibility study)*

Roadmap

Figure 9.2: More specific list of deliverables of a System Architect

9.3 System Architect Responsibilities

The System Architect has a limited set of primary responsibilities, as visualized in figure 9.3. The primary responsibilities are:

Balance of system properties as well as internal design properties. The system should be balanced: for example, the cost of subsystems should correspond with its added value in terms of functionality and performance. Architecting is a continuous balancing act in many incomparable dimensions and quantities.

Consistency across many organizational and design boundaries; From needs to implementation details, from system level to detailed implementation.

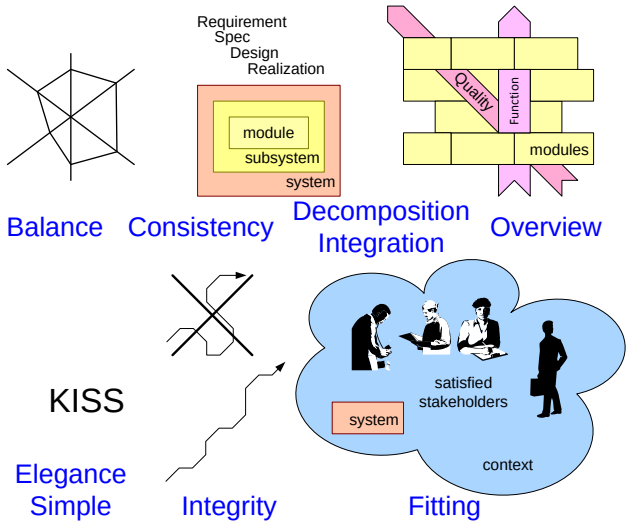


Figure 9.3: The primary responsibilities of the system architect are not tangible and easily measurable

Decomposition, Integration Decomposition is the standard answer in dealing with complex and big problems. Decomposing Systems in subsystems, subsystems in modules et cetera is a major responsibility of the architect. In most systems many decomposition dimensions are required: physical, logical, functional, and many more, see [18]. The complementary action of decomposition, however, is integration. The integral functioning and performance of the system is the ultimate goal of product creation, which emphasizes the importance of integration. In practice integration is much more difficult than decomposition, in fact the architect must decompose in such a way that integration is feasible.

Overview of the entire system and its context helps to make sensible specification and design decisions. The architect should provide overview to all members of the product creation team. Most of these members have a very limited horizon. The architect should help them by providing proper context information to make local design decisions.

Elegance, Simplicity are properties of a "good" architecture. The dangerous aspect of this responsibility is the highly subjective nature of elegance and simplicity. The appreciation of simplicity and elegance should be assessed or acknowledged by others than the architect.

Integrity of the system specification and design over time. The focus of a development team is often wandering over time, sometimes it depends on the

hype of the week. The architect is responsible for maintaining a balanced and focused development over time. For instance, when cost price reduction is required then the architect should keep performance and reliability on the agenda.

Fitting in stakeholder needs and system context, during the entire life cycle, is one of the core responsibilities of the architect. The architect must connect depth knowledge with breadth knowledge.

We can condense the primary responsibility of the System Architect as: to ensure the good functioning of the System Architecting Process. In practice, this responsibility is often shared by a team of System Architects, with one chief architect taking the overall responsibility.

responsibility	primary owner
business plan, profit	business manager
schedule, resources	project leader
market, saleability	marketing manager
technology	technology manager
process, people	line manager
detailed designs	engineers

Figure 9.4: (Incomplete) list of secondary responsibilities of the system architect and the related primary owner

The list of primary responsibilities as discussed above is suffering from a lack of measurability and is rather intangible. Systems Architects also have secondary responsibilities, where these are primarily owned by other persons. Most other roles in product creation are much sharper defined, as shown in Figure 9.4. For instance the business manager is responsible for the business plan and the financial results. The project leader is responsible for the schedule and hence for completing the project in time and within budget. The marketing manager is responsible for addressing the relevant markets and hence for market share and salability of the product. The technology manager is responsible for the timely availability of technologies and related tools. The line manager is responsible for the availability of the right people, with skills and processes to do their job. Final example are the engineers who are responsible for the design of their component or module.

9.4 What does the System Architect do?

Figure 9.5 shows the variety of activities of the day to day work of a system architect. A large amount of time is spent in gathering, filtering, processing and discussing detailed data in an informal setting. These activities are complemented by more formal activities like meetings, visits, reviews et cetera.

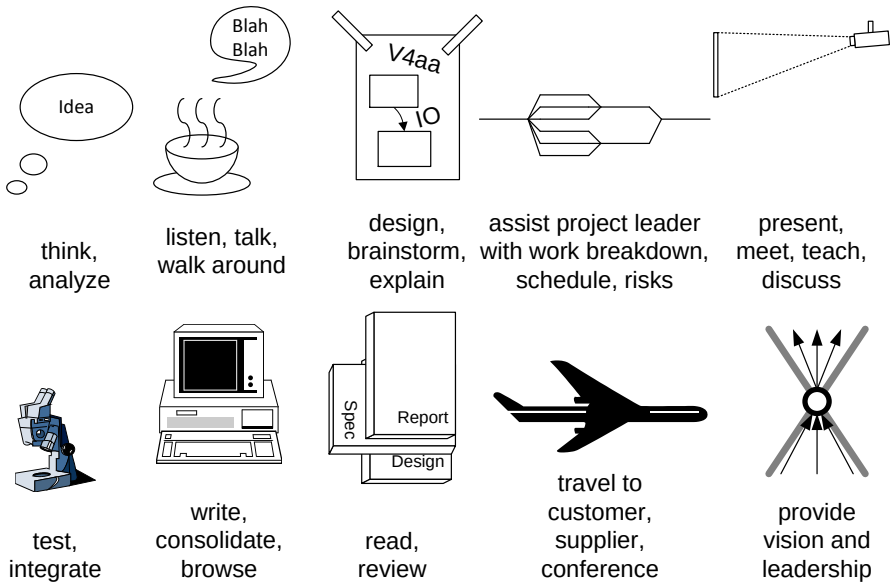


Figure 9.5: The System Architect performs a large amount of activities, where most of the activities are barely visible for the environment, while they are crucial for the functioning of architects

The system architect is rapidly switching between specific detailed views and abstract higher level views. The concurrent development of these views is a key characteristic of the way a system architect works.

Abstractions only exist for concrete facts

System Architects which stay too long at "high" abstraction levels drift away from reality, by creating their own virtual reality.

Figure 9.6 shows the bottom up elicitation of higher level views. A system architect sees a tremendous amount of details, most of these details are skipped, a smaller amount is analyzed or discussed. A small subset of these discussed details is shared as an issue with a broader team of designers and architects. Finally, the system architect consolidates the outcome in a limited set of views. The order of magnitude numbers cover the activities in one year.

The opposite flow in 9.6 is the implementation of many of the responsibilities

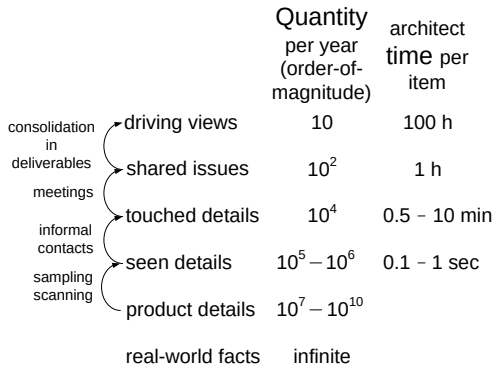


Figure 9.6: Bottom up elicitation of high level views

of the system architect. By providing overview, insight and fact-based direction a simple, elegant, balanced and consistent design will crystalize, where the integrity of designs goals and solutions are maintained during the project.

A lot of time spent by the architect serves the purpose of communication between many project members. The architect not only responsible for the system integration, but has also an integrating role in the project itself. The architect has to interact a lot with all the people mentioned in Figure 9.4, in order to fulfil the architect's responsibilities.

9.5 Task versus Role

The task of the system architect is to generate the agreed deliverables, see section 9.2 This measurable output is requested and tracked by the related managers: project leaders and the line managers. Many managers appreciate their architects only for this visible subset of their work.

The deliverables are only one of the means to fulfil the System Architect Responsibilities, as described in section 9.3. The system architect is doing a lot of nearly invisible work to achieve the system level goals, his primary responsibility. This work is described in section 9.4. Figure 9.7 shows this as a pyramid or iceberg: the top is clearly visible, the majority of the work is hidden in the bottom.

9.6 Acknowledgements

Nicolette Yovanof pointed out that the text belonging to Figure 2 and Table 2 was rather incomplete. She also mentioned that some more attention for the interaction with non-architects would be helpful. Chuck Kilmer provided feedback on "The Awakening of a System Architect", which resulted also in an update of

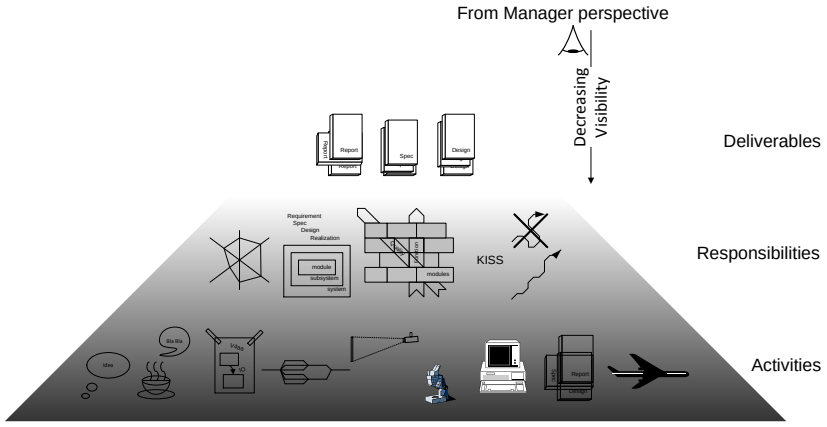
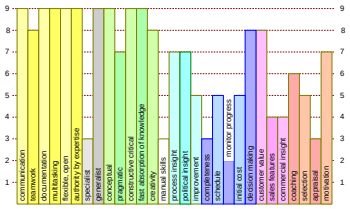


Figure 9.7: The visible outputs versus the (nearly) invisible work at the bottom

this paper. Byeong Ho Gong suggested a better coverage of the interfacing with customers/stakeholders. Pierre van de Laar provided textual improvements.

Chapter 10

Function Profiles; The Sheep with Seven Legs



10.1 Introduction

Many human resource and line managers struggle with the questions:

- What people have the potential to become good system architects?
- How to select (potential) system architects?

Employees thinking about their careers might similarly wonder if they have the capabilities to become a good systems architect.

We list a number of characteristics of individual humans. We map the these characteristics on different jobs, such as system architect, developer, and line manager, indicating the relative importance of this characteristic for that job. We first discuss the different jobs and their typical characteristics in 10.2 to 10.7. Then we elaborate the characteristics in 10.8.

The attention for this subject is increasing. Recent research is being carried out by Keith Frampton, see amongst others [5].

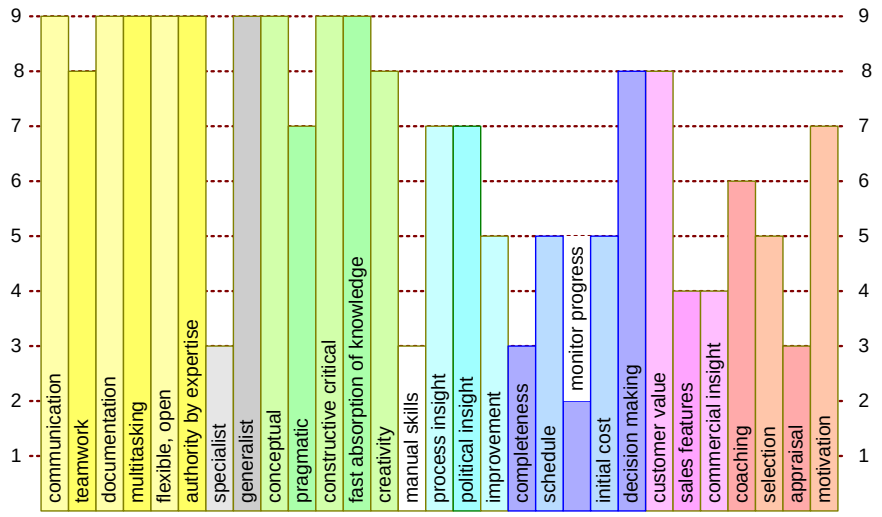


Figure 10.1: The function profile of the systems architect

10.2 Systems Architect Profile

The profile of the "ideal" system architect shows a broad spectrum of required skills. Quite some emphasis in the skill set is on *interpersonal skills*, *know-how*, and *reasoning power*.

This profile is strongly based upon an architecting style of technical leadership, where the architect provides direction *(know-how* and *reasoning power*) as well as moderates the integration (*interpersonal skills*).

The required profile is so requiring that not many people fit into it, it is a so-called **sheep with seven legs**. In real life we are quite happy if we have people available with a reasonable approximation of this profile. The combination of complementary approximations of such ideal architect allows for the formation of architecture teams. Such a team of architects can come close to this profile.

10.2.1 Most discriminating characteristics

In practice the following characteristics are quite discriminating when selecting (potential) systems architects:

- Generalist
- Multi-tasking
- Authority by expertise
- Balance between conceptual and pragmatic

Generalist The first reduction step is to select the *generalists only*, reducing the input stream with one order of magnitude. The majority of people feels more comfortable in the specialist role.

Multi-tasking The next step is to detect those people that need undisturbed time and concentration to make progress. These people become unnerved in the job of the systems architect, where frequent interrupts (meetings, telephone calls, people walking in) occur all the time. Ignoring these interrupts is not recommendable, this would block the progress of many other people. Whenever the people with poor multi-tasking capabilities become systems architect, then they are in severe danger of stress and burn out. Hence it is also the benefit to the person self to assess the multi-tasking characteristic fairly.

Authority by expertise The attitude of the (potential) architect is important for the long term effectiveness. Architects who work on the basis of delegated *power* instead of *authority by expertise* are often successful on the short term, creating a single focus in the beginning. However in the long run the inbreeding of ideas takes its toll. Architecting based on know-how and contribution (e.g. *authority by expertise*) costs a lot of energy, but it pays back in the long term.

Conceptual thinking and pragmatic The balance between conceptual thinking and being pragmatic is also rather discriminating. Conceptual thinking is a must for an architect. However the capability to translate these concepts in real world activities or implementations is crucial. This requires a pragmatic approach. Conceptual-only people dream up academic solutions.

10.3 Test Engineer Profile

The *test engineer* function at system level requires someone who *feels* and *understands* the system. Test engineers are capable of operating the system fluently and know its quirks inside out.

The main difference between an architect and a test engineer is the different balance between **conceptual thinking** and **practical doing**. Test engineers often have an excellent intuitive understanding of the system, however they lack the conceptual expression power and the communication skills to use this understanding pro-active, for instance to lead the design team.

10.4 Developer Profile

The core value of developers is their specific discipline know-how. Good developers excel in a limited set of specialties, knowing all tricks of the trade. On top

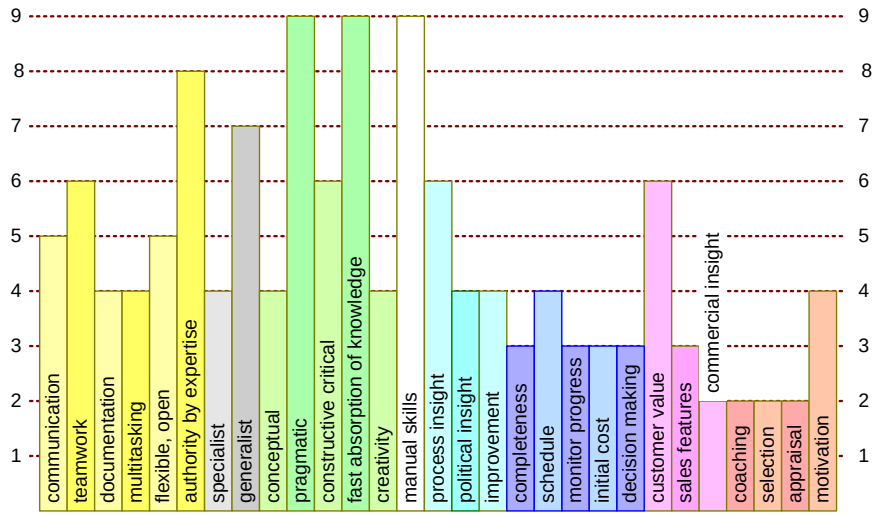


Figure 10.2: The function profile of the test engineer

of this they should be able to deploy this know-how in a creative way. In today's large development teams a reasonable amount of *interpersonal skills* are required as well as *reasoning power* and *project management* skills.

10.5 Operational Leader Profile

The *operational leader*, for instance a project leader, is totally focused on the result. This requires *project management* skills, the core discipline for operational leaders.

The *multi-tasking* capability is an important prerequisite for the operational leader too. If this capability is missing the person runs a severe risk of getting a burn out.

Note also that the operational leader functions as kind of gatekeeper, where the *completeness* is important.

10.6 Line Manager Profile

The *line manager* manages the intangible assets of an organization: the people, the technology and the processes. Technology and process know-how are tightly coupled with people, this know-how largely resides in people and is deployed by people. *Human resource management* skills and *process* skills are the core discipline for line managers, which need to be supported with sufficient *specialist* know-how.

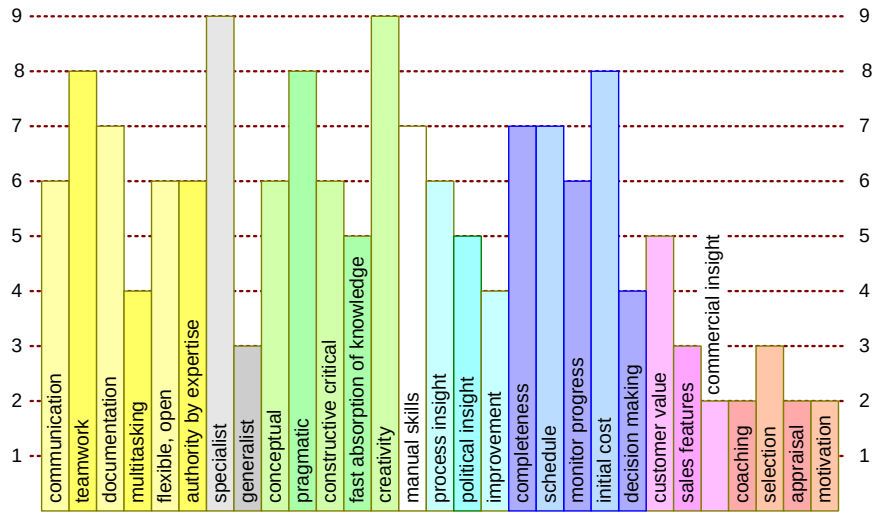


Figure 10.3: The function profile of the developer

10.7 Commercial Manager Profile

The *commercial manager* needs a commercial way of observing and thinking. This way of thinking appears to be fuzzy and not logical for technology oriented people. From technology oriented perspective a strange *mind warp* is required to perform a commercial manager function.

The commercial manager is a valuable complement to the other functions, responsible for aspects such as salability and value proposition.

10.8 Definition of Characteristics

10.8.1 Interpersonal skills

communication The ability to communicate effectively. Communication is a two-way activity, presenting information as well as receiving information is important.

teamwork The ability to work as member of a team, in such a way that the team is more than the collection of individuals.

documentation The ability to create clear, accessible and maintainable documentation in a reasonable amount of time.

multi-tasking The ability to work on many subjects concurrently, where (frequent) external events determine the task switching moments.

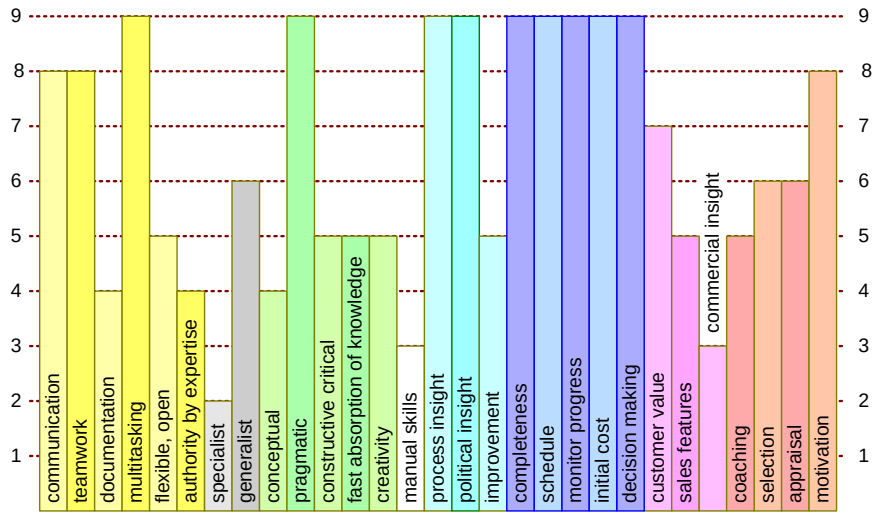


Figure 10.4: The function profile of the operational leader

flexible, open The attitude to respect contributions of others, the willingness to show all personal considerations, even if these are very uncertain, the willingness to adopt solutions of others, even in case of strong personal opinions.

Note that this overall attitude does not mean that a flexible and open person always adopts the ideas of others (chameleon behavior). The true strength of this characteristic is to apply it when necessary, so adopt an alternative solution if it is better.

authority by expertise The personality which convinces people by providing data, instead of citing formal responsibilities. Hard work is required before authority by expertise is obtained; a good track record and trust have to be build up. Authority is earned rather than being enforced.

10.8.2 Know-how

In terms of characteristics the know-how is qualified in 2 categories, generalist and specialist.

Generalist The persons which are always interested in the neighboring areas, how does it fit in the context? How does the “whole” work.

Specialist The persons which are always interested in knowing more detail.

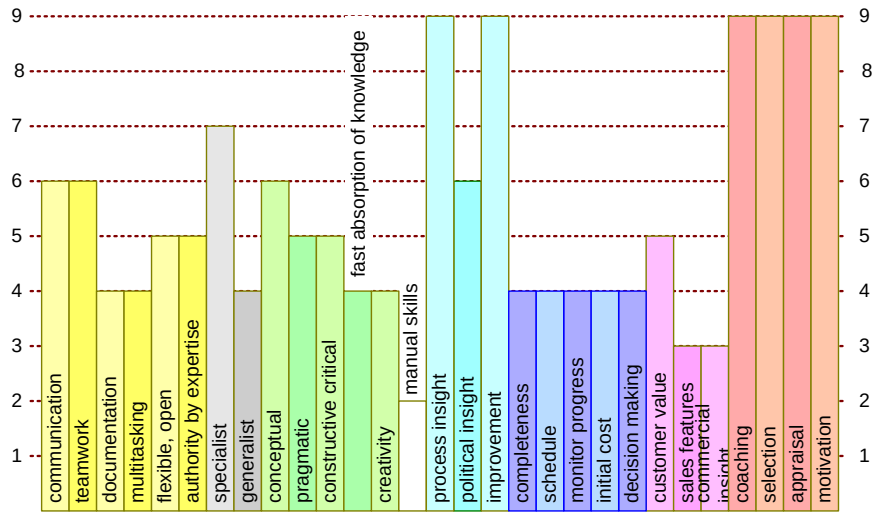


Figure 10.5: The function profile of the line manager

10.8.3 Reasoning Power

conceptual The ability to create the overview, to abstract the concepts from detailed data. The ability to reason in terms of concepts.

pragmatic The ability to accept non-ideal solutions, to go after the 80% solution. The ability to connect ”fuzzy” concepts to real world implementations.

constructive critical The ability to identify problems, formulate the problems and to trigger solutions. The term *critical thinking* is also used. Note that critics serves a constructive goal: to achieve better results.

fast absorption of know-how The ability to jump into a new discipline and to absorb the required know-how in a short time. Systems architect are never able to know all about the technologies used in the systems. This capability helps them to get the right knowledge when needed.

creativity The ability to come with new, original ideas. A specific subclass of this ability is lateral thinking: applying know-how from entirely different areas on the problem at hand.

10.8.4 Executing Skills

Manual Skills The ability to **do** things, for instance build or test something. This ability is complementary to the many ”mental” skills in this list of characteristics.

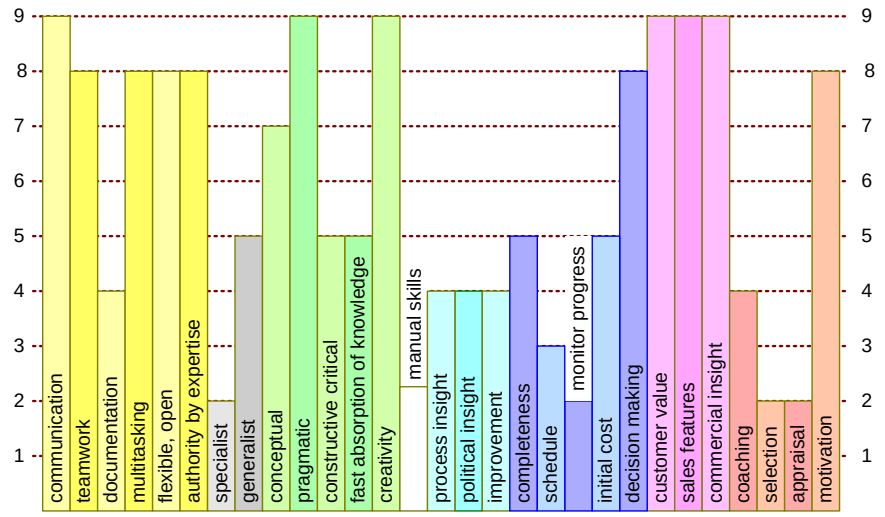


Figure 10.6: The function profile of the commercial manager

10.8.5 Process Skills

process insight The ability to understand specific processes, the ability to recognize the de facto processes, the ability to asses formal and de facto processes, both the strong points as well as the weak points.

politics insight The ability to recognize the political factors: persons, organizations, motivations, power. The ability to use this information as neutralizing force "depoliticizing": facts and objectives based decision making instead of power based decision making.

improvement drive The ever present drive to improve the current situation, never getting complacent.

10.8.6 Project Management Skills

Completeness The ability to pursue **all** information. This is often done by means of spreadsheets or databases. Large collections of issues are maintained and processed.

This ability is often complementary to, or even conflicting with, the ability to create understanding and overview: the parts view versus the holistic view.

schedule The ability to create schedules: activities and resources with their relationships, scheduled in time.

monitor progress The ability to monitor progress, the ability to chase people, and the ability to find and resolve the causes of delays.

initial cost The ability to create initial cost estimates and to refine these into budgets. The ability to understand and reason in terms of initial costs. Initial costs are the one time investments needed to develop new products and or businesses.

decision making The ability to make choices and to handle the consequences of these choices.

10.8.7 Commercial Skills

customer value The ability to see and understand the value of a product or service for a customer. The ability to asses the value for the customer.

sales feature The ability to recognize features needed to sell the product. The ability to characterize the relevant characteristics of these features ("tick-mark only", "competitive edge", "show-off", et cetera).

commercial insight The ability to think in commercial terms and concepts, ranging from "branding" to "business models".

10.8.8 Human Resource Management Skills

coaching The ability to coach other people; help other people by reflection, by stimulating independent thinking and acting.

selection The ability to select individuals for specific jobs. The ability to interview people and to asses them.

appraisal The ability to asses employees and to communicate this assessment in a fair and balanced way.

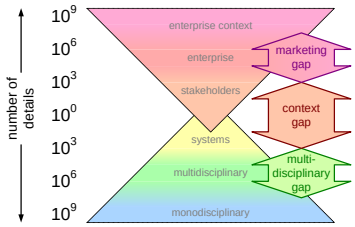
motivation The ability to make people enthusiastic, to motivate them beyond normal performance.

10.9 Acknowledgements

Pierre America applied fine tuning of translations, spelling and capitols. Lennart Hofland suggested an improvement for the description of the commercial manager. Sjir van Loo suggested an increase of coaching and selection skills of the architect. Keith Frampton pointed me to recent research about this subject.

Chapter 11

Dynamic Range of Abstraction Levels in Architecting



11.1 Introduction

System architects need the capability to "zoom in" and "zoom out". A tremendous dynamic range of abstraction has to be covered from high level business and customer objectives to detailed design decisions at engineering level. The system-of-interest itself spans many abstraction levels. However the architect has to look beyond the system-of-interest itself, towards the customer context, the life cycle, and to related products.

11.2 From System-of-Interest to Context

The translation of the product specification of the system-of-interest into detailed mono-disciplinary design decisions spans many orders of magnitude. The few statements of performance, cost and size in the system requirements specification ultimately result in millions of details in the technical product description: million(s) of lines of code, connections, and parts. The technical product description is the accumulation of *mono-disciplinary* formalizations. Figure 11.1 shows this

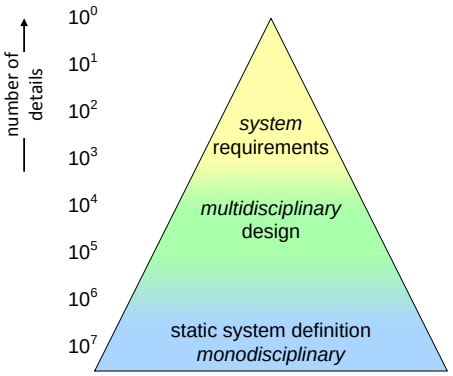


Figure 11.1: Connecting System Specification to Detailed Design

dynamic range as a pyramid with the system at the top and the millions of technical details at the bottom.

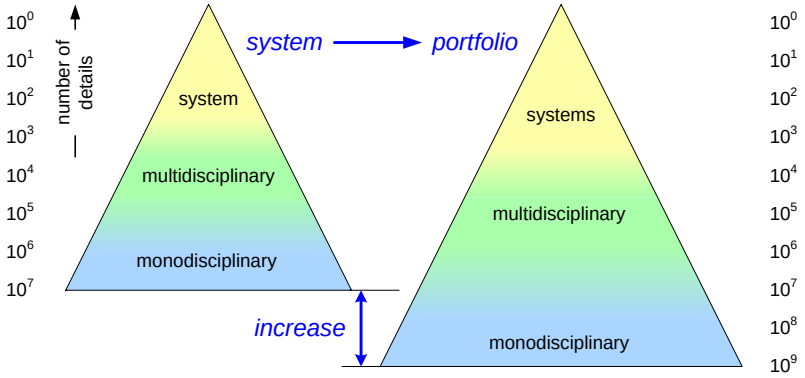


Figure 11.2: From system to Product Family or Portfolio

The current system-of-interest is most often part of a broader set of products that evolves over time: the product family or portfolio. The aggregate amount of details in the product family or portfolio can be several orders of magnitude larger than the amount of details for one system. Figure 11.2 shows the increase of the dynamic range from system to portfolio.

Architects also have to take the context of the system into account, from both customer as well as business perspective. We can transform the portfolio pyramid from Figure 11.2 into Figure 11.3 to show the number of details of a portfolio in its context. The context is also shown as a pyramid, representing the fact that in the outside world, where systems are actually used, can be viewed at many levels of abstractions.

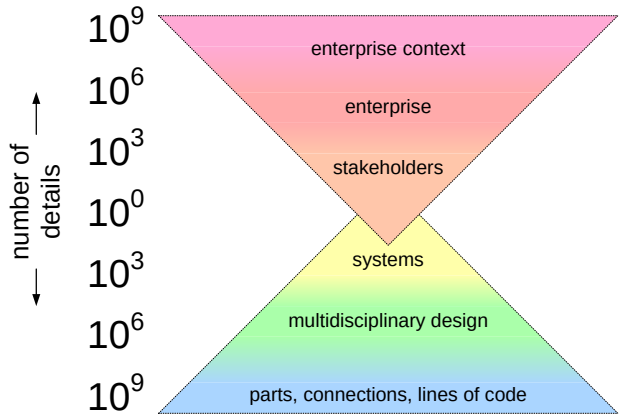


Figure 11.3: Product Family in Context

11.3 Architecture and Architecting

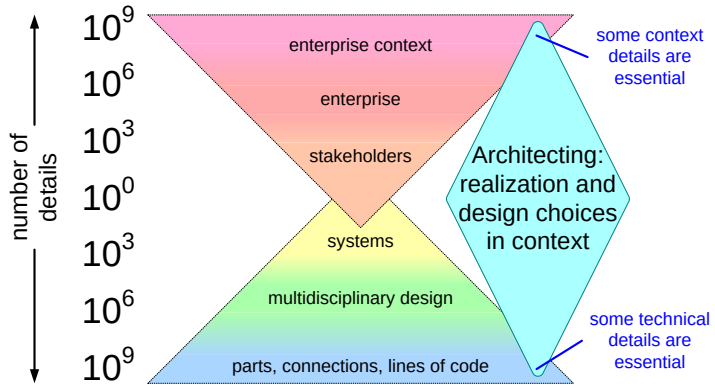


Figure 11.4: Architecture: the Essence of System and Context

The challenge of developing an architecture is to capture the essence of both the systems to be build as well as the contexts where systems are being created and used. Figure 11.4 shows that most of an architecture covers the higher abstraction levels. An architecture needs to abstract from most details to facilitate the capture of the essence. Only a simplified description or model can be used at system level to reason and facilitate communication.

However, some crucial details either from mono-disciplinary area or from the customer or business contexts might have to be included. Quite often the devil is in the detail. Hence known crucial details are part of an architecture description or

model.

Note that architectures do have a scope:

System architecture captures the essence of a system in its context. Note that the *system context* includes the product family or portfolio. However, the focus of the *system architecture* is on the system itself, and as such will position this system in the broader portfolio.

Family architecture captures the essence of the family of systems and its context. The focus is now on the family, explaining how different products can support specific market needs, and providing guidance to harvest synergy between products.

Portfolio architecture is similar to family, but at an higher aggregation level.

Architecting involves all activities to create an architecture: exploring details in system(s) and context, communication, design, specification, making decisions et cetera. In other words architecting combines *external* zoom-in and zoom-out (fact gathering and communication) with *internal* zoom-in and zoom-out (specification, design, integration).

11.4 Revisiting Design and Engineering

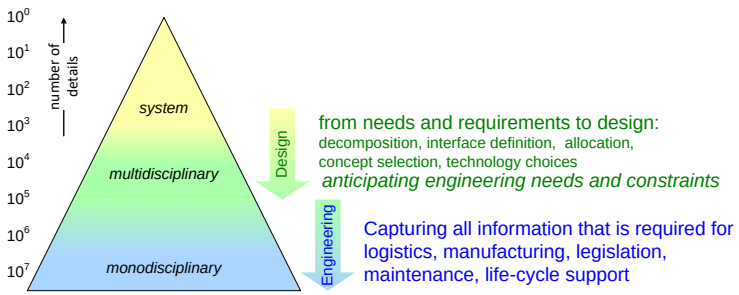


Figure 11.5: Positioning *design* and *engineering* in the dynamic range of abstraction levels

We can revisit the terms *design* and *engineering* based on the dynamic range of abstraction levels, as shown in Figure 11.5.

Designing is the activity to get from needs and requirements to a design: decomposition, interface definition, allocation, concept selection, technology choices, etc. The design has to anticipate the engineering needs and constraints.

Engineering is capturing all information that is required for the Customer Oriented Process, such as logistics, manufacturing, legislation, maintenance, life cycle support.

Engineering and design mostly takes place internally in the organization, with the exception of the communication with external suppliers.

11.5 Architecting and Design in Practice

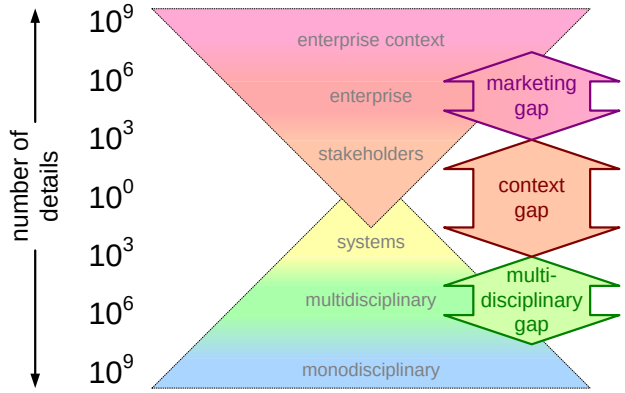


Figure 11.6: Frequently observed gaps in practice

In practice, several problems can be observed in most organizations that can be explained by "gaps". Figure 11.6 shows some gaps that can be observed in many organizations:

Multi-disciplinary gap is the gap between product specification and detailed design decisions.

Context gap is the gap between stakeholders and product specification.

Marketing gap is the gap between the detailed outside world with billions of individuals and our abstracted understanding in terms of stakeholders, concerns, and needs.

Architects have a core role in closing and preventing the multi-disciplinary and the context gaps. In practice, the marketing managers do have the irresponsibility for the marketing gap with their knowledge of stakeholders, enterprises, and enterprise contexts.

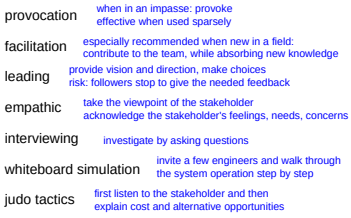
The multi-disciplinary gap, from specification to detailed design, is often bridged by experience: older engineers make decisions based on their past experiences.

Note that these decisions are often right. The problem is that the implicit nature of these decisions does not facilitate communication, review, or discussion. Worse is that this knowledge gradually disappears from the organization, making further evolution even less transparent.

The context gap, how marketing research information relates to choices in the product specification, requires an extrovert focus of architects. Early in their careers many architects look inward (to design and engineering) and too little outward (to customers and other stakeholders in the Customer Oriented Process). Architects make major development steps when they start to address both gaps in a balanced way.

Chapter 12

Architecting Interaction Styles



12.1 Introduction

A system architect has to use different interaction styles in different circumstances. In some circumstances a *leading* style is appropriate, while in other circumstances a *facilitating* style is more effective. Figure 12.1 shows the styles that are discussed in this chapter.

12.2 Provocation

A provocative style can be used by the architect when the discussion is in an impasse. The provocation can be based on taking an extreme viewpoint of one of the stakeholders and confronting the other stakeholders with the consequences. Such a provocation forces the involved stakeholders to formulate their needs more sharp, including the consequences of following the recommendation.

A provocative style should be applied scarcely. Once team members get used to this style then the style becomes ineffective. Most people do not like to be provoked continuously, so they stop to respond after a few provocations.

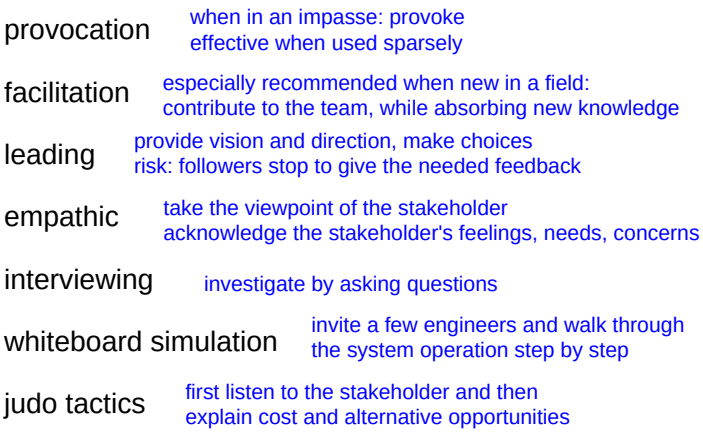


Figure 12.1: Interaction styles for architects

12.3 Facilitation

The facilitation style is a style where the architect serves the team by facilitating meetings and workshops. Facilitating a meeting means:

- preparing the meeting or workshop together with the owner of the meeting: determining the goal, participants, place, agenda, means.
- facilitating the meeting itself: timekeeping, managing the flips, writing action point and conclusions.
- finalizing the meeting: writing a report and presentation of the results, chasing follow-up actions.

The facilitation style is especially useful for architects entering a new domain. The architect provides visible value for the team, while as a spin off the architect learns a lot about the new domain.

12.4 Leading

A leading style is a style where the architect is highly visible. The architect provides vision and direction to the team. The leading architect can be recognized by looking at the followers: if they really follow the architect then the architect is effective as leader.

The risk of this style is that the team starts to trust the architect decisions too much. Most of the team members have much more know how about the design issues than the architect. The architect will often make decisions based on limited know how that should be corrected by the specialists with more know how. The

leading style sometimes inhibits the specialists to oppose the architect. The leading architect must be aware of this effect. Sometimes even invitations to oppose and provocations do not help to loosen up the followers.

12.5 Empathic

The empathic style is based on taking the viewpoint of the stakeholder under discussion. This goes much further than the objective rational view. The feelings and emotions of this stakeholder must be taken into account as well. The understanding of the state of mind is communicated back to the stakeholder. The result of this way of interacting is that the architect gets a much better insight in the stakeholder, while at the same time the stakeholder has the feeling to be taken seriously.

12.6 Interviewing

Architects pose lots of questions, questions are one of the most important instruments of the architect. The interviewing style makes excessive use of questions. The architect uses a priori knowledge to formulate open questions. These open questions must lead to an understanding of the stakeholder concerns.

The difficult part of this style is to use a priori know how in a limited and constructive way. The danger of a priori know how is that it limits observation and that suggestive questions are formulated instead of open questions.

12.7 White-board simulation

The white-board simulation style is used in meetings where a few specialists are present. The architect guides the specialists through a use case, where every specialist explains the system behavior from the specialist viewpoint. For example, the use case can be to push a *next channel* button on the user interface. In this example the user interface signal will trigger an avalanche of events in the system, going through many layers and propagating to many subsystems.

This guided simulation often reveals a lot of unknown system behavior, strange dependencies, inefficient sequences and many more engineering surprises. The normal reactions of the participants is that after a few steps they want to redesign the system. The architect should suppress this urge, by parking improvements at the side. The main purpose of this style is to build a shared understanding of the current design.

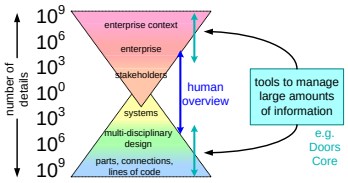
12.8 Judo tactics

The basis of judo tactics is that the architect starts to listen to the stakeholder, especially when the architect feels an urge to contradict the stakeholder. After listening to the stakeholder, and acknowledging the validity of the needs, the architect explains the costs and trade-offs. In many cases the stakeholders have a healthy feeling for value and cost and look for a reasonable balance. Quite often the result is a decision that the architect wanted to make right at the beginning. However, this style works only if the architect really listens, and is willing to take a different direction if needed. It might be that the architect discovers that the value for the stakeholder is much larger than originally assumed!

In many cases ill communication and bad listening skills block reasonable decisions. The judo style, where the architect starts to listen, avoids this trap.

Chapter 13

The Tool Box of the System Architect



13.1 Introduction

The subject of tools for systems architecting creates numerous debates. We will use a broad interpretation of the word tool, including intellectual tools, low-tech tools (such as pen and paper), but we will also discuss computer assisted tools. One of the key questions is when to apply what tool. An essential capability for systems architects is to pick an appropriate tool, and, if needed, to adapt it to the situation at hand.

We discussed in Sections 4 and 8 that the role of the systems architect depends on the organizational context. Similarly, there are organizations that force a set of tools on systems architects based on the perceived role and way of working of system architects.

We base our discussion of tools on the deliverables, responsibilities and activities as described in Sections 9 and 11 (Figures 11.4 and 11.6). Key contribution of systems architects in these sections is the simplification of complicated systems into understandable essentials. Main challenges in achieving this contribution are the heterogeneity of the system and its context, and the uncertainties and unknowns in the system and its context. The goal is to make systems specification and design decisions communicable, and to facilitate debate and reasoning about decisions.

Many organizations move in practice too fast to extensive use of computer assisted tools. As consequence the architects and stakeholders move away from overview and understanding essentials to more detailed concerns (that also have be to be addressed!). The purpose of this section is to help understand the impact of tool selection, and especially to bring balance in the application of *intellectual* tools versus *computer assisted* tools.

13.2 Overview of Systems Architecting Tools

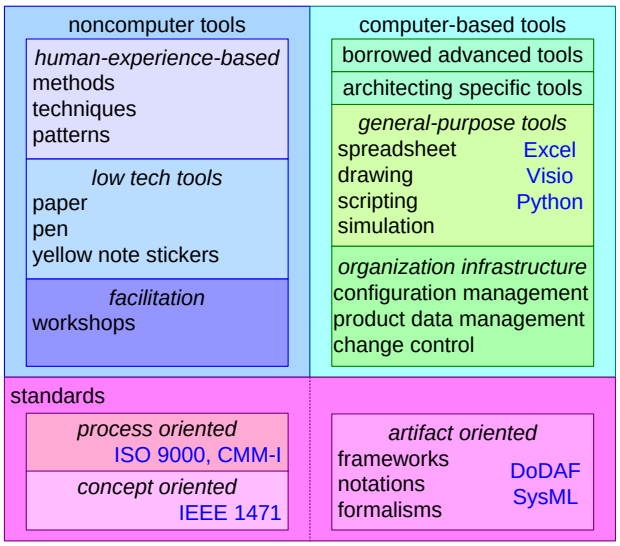


Figure 13.1: Classification of Architecting Tools

Figure 13.1 shows an overview and a classification of systems architecting tools. The left side shows the tools that are independent of computers and related software programs. The right hand side shows tools that depend on computers and specific software. The bottom part of the figure shows some of the standards that impact the selection and application of tools.

13.2.1 Human Experience Based tools

Experience is crucial for systems architects. Systems architects meet new approaches during their entire career, and they build a rich frame of reference by seeing many systems in many circumstances. Reflection on the way of working transforms events into valuable experience: approaches are transformed into *methods* and *techniques*, and problems and solutions are transformed into *patterns*.

Methods describe an approach in terms of objective, the order or logic of steps to follow, the techniques that can be applied, and the models, tools, notations, and formalisms that can be supportive.

Techniques are ways to address a specific aspect of a problem. For example, how to analyze timing requirements and problems. Techniques can be supported by specific tools and formalisms. A technique may require specific models. For instance, the analysis of response time may require functional flow models.

Patterns are recognized problem-solution combinations, including the considerations in what context and circumstances a solution is appropriate for the problem. Patterns can be highly technical, e.g. the publish-subscribe pattern in software to solve flexibility and extendibility needs. However, patterns can also be high level organizational or business, such as considerations about products versus services in Figure 6.5.

13.2.2 Low-tech tools

Systems architects, like building architects, often make sketches. The sketches are on napkins, paper, flip-charts, white-boards, using pens, pencils et cetera. Sketching is a fast way to express and exchange ideas, and as such has to be values as essential part of the systems architect toolbox. Note that similarly other low tech means, such as folded paper, wire frames, and yellow note stickers provide fast and intuitive ways to express and exchange ideas.

It might be a challenge to capture these sketches for further communication, later re-use, and archiving. However, with today's ubiquitous digital cameras this is easily captured. Later in the process these sketches get captured electronically in more structured form, e.g. in Visio.

13.2.3 Facilitation tools

System architects can contribute to teams by applying facilitation techniques, as described in Section 12. An example is the organization of work shops, where teams can explore and share ideas effectively. There are many more facilitation tools and techniques, such as:

- the use of flip charts to create a common memory on the wall
- the use of balanced feedback, e.g. soliciting benefits and concerns
- working in teams and plenary groups
- preparing meetings together with the leader
- round robin or random order contributions to get input from less dominant team members

13.2.4 Borrowed Advanced Tools

Systems architects cooperate with a large amount of experts. Every expert has its own set of tools. Sometimes the architect borrows such tool and adapt it to be used as system level. For example, mechanical engineers are used to tolerance budgets. Systems architects use budgets for many different system qualities (e.g. response time), where granularity of the budget and the algorithms behind the budget have to be adapted to the quality at hand. Most tools in systems architecting find their origin somewhere in another discipline.

13.2.5 Architecting Specific Tools

The problems to be addressed by a tool and the solutions for these problems need to be well-defined and repeatable Before a computer assisted tool can be made. The nature of many systems architecting problem is often quite opposite, with characteristics such as heterogeneous, uncertainties and unknowns. The systems architecting effort is mostly spend in understanding the problem. Solving well understood problems in a repeatable and predictable way is the domain of engineering.

Most *systems* specific tools are more engineering related (nailing down all detailed information to facilitate the ordering, production, sales, and support of the system) than architecting related. Examples are tools to capture requirements (e.g. Doors), functional and physical architectures such as IDEF0 (e.g. Core), or object oriented architectures in for instance SysML.

13.2.6 General Purpose computer based tools

Architects and engineers use computers all the time for many different purposes. Architects will use a lot of general purpose tools, such as spreadsheets (e.g. Excel), drawing programs (e.g. Visio), scripting (e.g. Python), or simulation (e.g. Python, MATLAB or many others).

The general purpose nature of these tools makes them attractive for architects, since that helps them to cope with heterogeneity, unknowns and uncertainties. The class of more advanced tools can be too restrictive to allow adaptation to the problem at hand and its circumstances.

13.2.7 Tools prescribed by the organization infrastructure

Organizations do have an engineering tool infrastructure that systems architects can not ignore. However, systems architects have to decide when and how to interface to the organizational infrastructure. Examples of typical organizational infrastructures are many data bases and repositories for engineering related information:

Configuration management describing the parts and the rules how the parts can be configured. This repository can be part of a larger system such as an Enterprise Resource Planning (ERP) system (typically SAP).

Product Data Management (PDM) storing all product and part related information required for the Customer Oriented Process.

Change Control and Problem Report data bases, where all Change Requests, Internal Problem Reports, and Field Problem reports are stored.

System architects sometimes have to work for some time outside these systems, because these systems tend to slow down more creative work full of unknowns and uncertainties. The challenge for project leaders and systems architects is to migrate to these systems at the right moment: using these systems too early slows dona too much, starting to use them too late might cause loss of information and quality problems.

13.2.8 Process Oriented Standards

There are many process oriented standards that influence the way of working of systems architects. For example the maturity models in CMM-I more or less prescribe most of the tools (Configuration management, change control) discussed in the previous paragraphs.

Process oriented standards tend to be agnostic for specific tools. In general these standards try to capture best practices from the past in an attempt to preventing past mistakes. Systems architects in practice suffer when these processes are implemented to the letter rather than the intent. An unintended side effect can be that systems architects are transformed into administrators, while their main contribution is in content rather than administration.

13.2.9 Concept oriented Standards

Some standards try to capture the shared understanding of the architecting discipline. A good example is the IEEE 1471 standard, where the concepts *stakeholders*, *concerns*, *architecture description*, and *viewpoints* are captured. These standards do no prescribe a way of working but provide a set of concepts and their relations to ease communication.

13.2.10 Artifact Oriented Standards

In the defense world several frameworks have been created defining the artifacts that can describe an architecture. Typical examples are DoDAF and MoDAF of respectively the USA Department of Defense and the UK Ministry of Defense. These frameworks do not define the process, but rather limit themselves to defining

the artifacts that may describe the architecture. These standards tend to see the artifacts as electronics artifacts with a significant degree of formalization to facilitate computer assistance.

Part of the Systems Engineering community has transformed UML from the software engineering world into a more systems oriented modeling language SysML. SysML is a set of formalisms to create artifacts that can be used for computer assisted tools.

13.3 Human versus Computer Assisted Tools

One of the main challenges is to decide when and for what to use computer assisted tools, as stated in the introduction. Figure 13.2 shows a so-called four quadrant analysis of intellectual (human) tools and computer assisted tools. The four quadrants are obtained by adding a second dimension: strength and weakness.

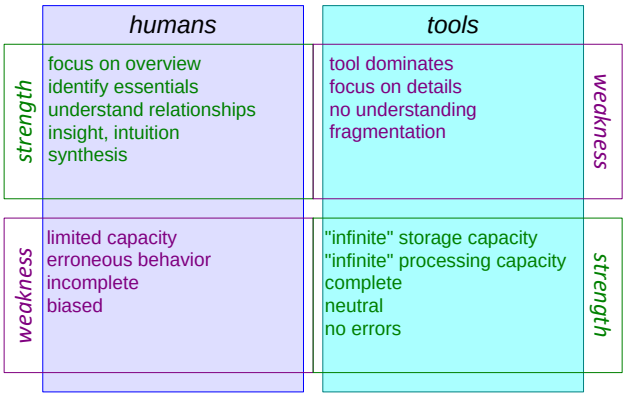


Figure 13.2: 4 Quadrant analysis of computerized and human tools

Strengths of humans , based on their intellect, are:

- to be able to focus on overview
- to be able to identify the essentials
- to understand relationships
- to have insight and intuition
- to be able to synthesize (to combine heterogeneous information into a meaningful picture)

Strengths of computers , based on current technological level, are:

- near-infinite storage capacity

- near-infinite processing capacity
- the ability to be *complete* by storing *all* information
- to be neutral, without emotions, opinions, or (political) interests
- to be perfect in execution, making *no errors*

Weaknesses of humans , inherent to their social and psychological background, are

- storage and processing capacity is limited.
- showing behavior that is erroneous
- memory is imperfect, information is often *incomplete*
- biased, for emotional, social or political reasons

Weaknesses of computers , inherent to their mechanistic technical nature, are:

- the *tool dominates*, because there is no "reasonable" flexibility
- the information is in full detail, moving the *focus on details*
- computers do not have any *understanding* (garbage in, garbage out)
- the data tends to be *fragmented*, only stored relations are present.

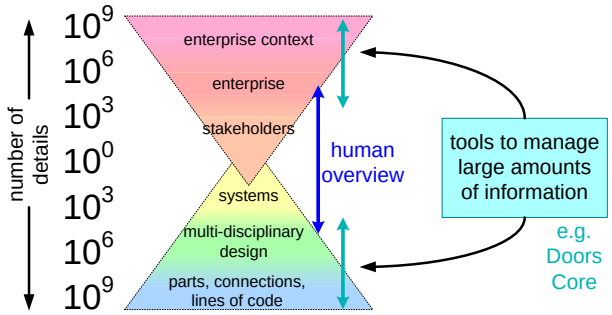


Figure 13.3: Tools Support Processing of Large Amounts of Details

The idea behind the four quadrants is that the weaknesses of humans can be compensated by the strengths of computers and vice versa. If we map these characteristics on the pyramids of Figure 11.4 then we see that human intellect is required at the higher abstraction levels where we strive for understanding between heterogeneous stakeholders. Computer assisted tools bring most of their value where large amounts of data have to be managed and processed. Most computer assisted tools address a limited set of concerns, such that the problem is well defined and the solutions can be applied repeatable and predictable. Many computer assisted tools are mono-discipline oriented, since disciplines capture repeatable knowledge.

13.4 Flow: from Data to Overview and Understanding

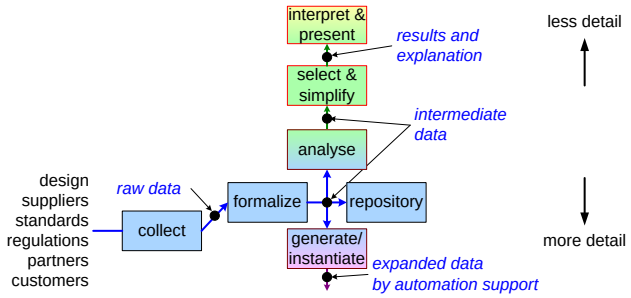


Figure 13.4: From Data to Understandable Information

We have seen in the previous Subsection that computer based tools create most of their value when large amounts of data have to be managed and processed. Other discussions that pop up when computer assistance is used is the degree of formalization and the use of automated outputs. Figure 13.4 shows the flow from input data up to the moment that the results are being used by a heterogeneous group of stakeholders. The figure shows the following functions:

Collect data from many inputs, e.g. the design, suppliers, standards, regulations, partners, and customers. The output of this function is a collection of *raw* data: data that still has to be processed to make it useful.

Formalize to be able to enter the data into computer based tools. The nature of the formalization is to look for appropriate abstractions to capture this data. The consequence of the abstraction is that the amount of detail can decrease slightly, for instance because repeated data is captured more structurally.

Repositories are used to store the formalized data so that this data can be used for many different purposes. For example an information model can be stored as entity relation ship model plus a data dictionary to capture all formatting details. This information model data can be used to generate data structures and code, it can be used to generate test cases for compliance testing, and the data can be used for analysis.

Generation and Instantiation can be applied on prescriptive data in the repository to generate or instantiate components, stubs or test harnesses.

Analysis techniques are applied on the data to determine characteristics of the design. For example, the form, shape and material characteristics of components can be used to calculate the center of gravity of components and of the

aggregate of multiple components. Another example is that configurations can be analyzed for feasibility and performance.

Selection and Simplification is a function that is applied by humans (architects or designers) to make the results ripe for communication and discussion. The output of automated analysis techniques is often rather detailed and highly formal, while the essential aspects are hidden in a huge amount of other details.

Interpretation and Presentation are the last steps in making the information accessible and understandable for the broader group of stakeholders. In interpretation the meaning of the the outputs is added: is a center of gravity deviation of 10mm a problem or is it quite good? The presentation is the format of the output, what visualization will engage the stakeholders, how to ensure that the information relates to the mental model of the diverse stakeholders?

A common mistake made by engineers is that they show their own intermediate data to stakeholders that use a different mental framework themselves. The consequence is that the communication is quite incomplete and the risk is significant that stakeholders will disconnect or will not give any reaction even when necessary.

Systems architects have to make the last steps of selection, simplification, interpretation and presentation. Note also that these steps bring their own risks: every simplification is only valid within its limits, so architects are also responsible to monitor the validity of discussions and decisions in light of the used simplifications.

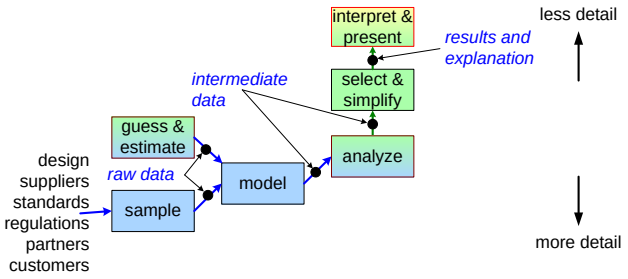


Figure 13.5: Data Flow Early in Creation Process

Early in the development projects architects are using a slightly simplified flow to facilitate system specification and design, as shown in Figure 13.5. This figure shows that early in the process many estimates and guesses are used, and that less formalization is used. Remember that formalization and computer based tools are especially relevant when large amounts of data have to be processed and managed. More simple models can be used by architects as long as the amount of information is small.

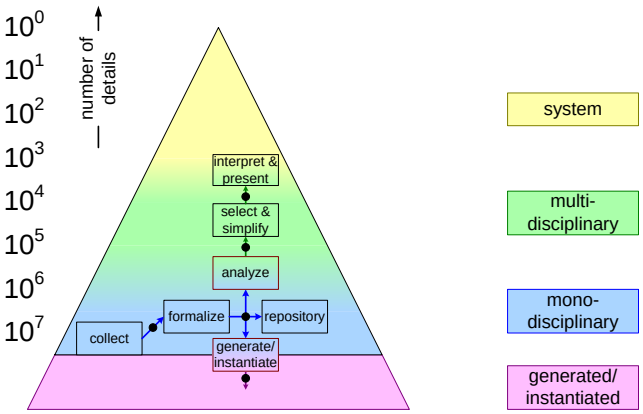


Figure 13.6: Data Flow Mapped on Pyramid

Figure 13.6 maps the data flow on the pyramid with the abstraction levels. This mapping shows again the relation between the amount of information and the kind of tools to be used: repositories, generator tools and analysis tools are typically computer assisted, while the intellectual challenges of selection, simplification, interpretation, and presentation are human activities.

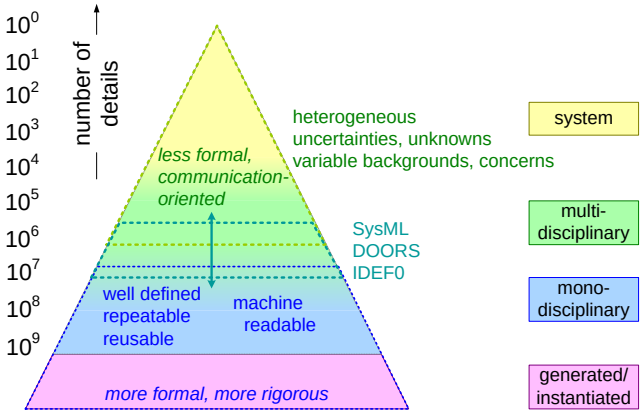


Figure 13.7: Formality Levels in Pyramid

Figure 13.7 summarizes these areas of application in the pyramid. The bottom parts of the pyramid with large amount of details can be characterized as more formal and requiring more rigor. Formalization requires well defined problems, data, and operations that are repeatable. The data is machine readable to allow automated tools. The use of repositories facilitates re-use over systems and components.

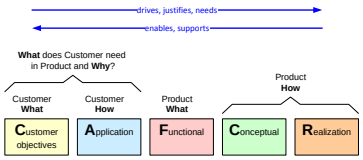
The upper part of the pyramid is characterized by the combination of quite heterogeneous data with uncertainties and unknowns used ba heterogeneous group of stakeholders with variable backgrounds and concerns. This upper part is less formal and oriented towards communication, discussion and decision making.

Part III

Market, Requirements, Roadmapping

Chapter 14

Short introduction to basic "CAFCR" model



14.1 Introduction

A simple reference model is used to enable the understanding of the inside of a system in relation to its context.

An early tutorial[20] of this model used the concatenation of the first letters of the views in this model to form the acronym "CAFCR" (Customer Objectives, Application, Functional, Conceptual, Realization). This acronym is used so often within the company, that changing the acronym has become impossible. We keep the name constant, despite the fact that better names for some of the views have been proposed. The weakest name of the views is *Functional*, because this view also contains the so-called *non functional* requirements. A better name for this view is the Black-Box view, expressing the fact that the system is described from outside, without assumptions about the internals.

The model has been used effectively in a wide variety of applications, ranging from motor way management systems to component models for audio/video streaming. The model is not a silver bullet and should be applied only if it helps the design team.

14.2 The CAFCR model

A useful top level decomposition of an architecture is provided by the so-called "CAFCR" model, as shown in figure 19.4. The *Customer Objectives* view and the *Application* view provide the **why** from the customer. The *Functional* view describes the **what** of the product, which includes (despite the name) also the *non-functional* requirements. The **how** of the product is described in the *Conceptual* and *Realization* view, where the conceptual view is changing less in time than the fast changing realization (Moore's law!).

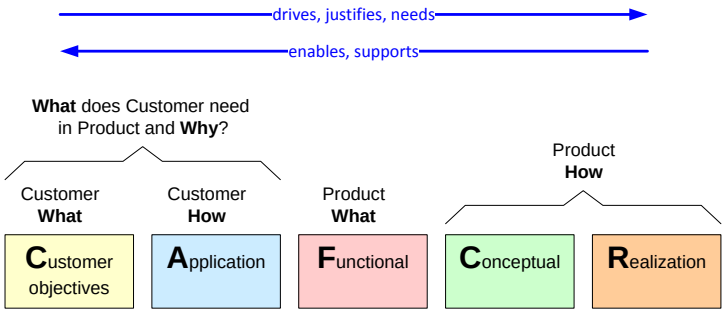


Figure 14.1: The "CAFCR" model

The job of the architect is to integrate these views in a consistent and balanced way. Architects do this job by *frequent viewpoint hopping*: looking at the problem from many different viewpoints, sampling the problem and solution space in order to build up an understanding of the business. Top-down (objective driven, based on intention and context understanding) in combination with bottom-up (constraint aware, identifying opportunities, know how based), see figure 19.5.

In other words the views must be used concurrently, not top down like the waterfall model. However at the end a consistent story-line must be available, where the justification and the needs are expressed at the customer side, while the technical solution side enables and support the customer side.

The model will be used to provide a next level of reference models and submethods. Although the 5 views are presented here as sharp disjunct views, many subsequent models and methods don't fit entirely in one single view. This in itself not a problem, the model is a means to build up understanding, it is not a goal in itself.

14.3 Who is the customer?

The term *customer* is easily used, but it is far from trivial to determine the customer. The position in the value chain shows that multiple customers are involved. In

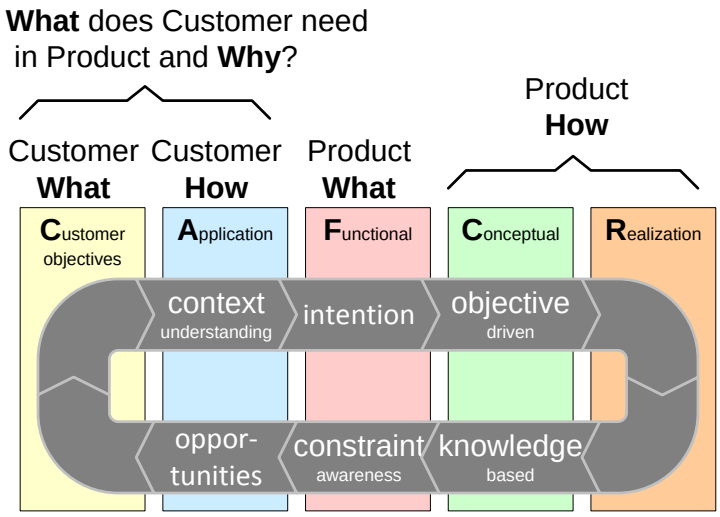


Figure 14.2: Five viewpoints for an architecture. The task of the architect is to integrate all these viewpoints, in order to get a *valuable*, *usable* and *feasible* product.

figure 19.6 the multiple customers are addressed by applying the CAFCR model recursively.

The customer is a gross generalization. Marketing managers make a classification of customers by means of a market segmentation. It is recommended to start building up insight by making specific choices for the customer, for example by selecting specific market segments. Making early assumptions about synergy between market segments can handicap the build-up of customer understanding These kind of assumptions tend to pollute the model and inhibits clear and sharp reasoning.

many stakeholders are involved for any given customer. Multiple stakeholders are involved even when the customer is a consumer, such as parents, other family, and friends. Figure 14.4 shows an example of the people involved in a small company. Note that most of these people have different interests with respect to the system.

Market segments are also still tremendous abstractions. Architect have to stay aware all the time of the distance between the abstract models they are using and the reality, with all unique infinitely complex individuals.

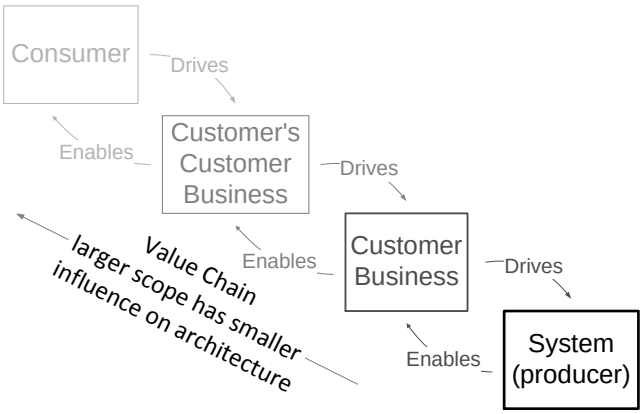


Figure 14.3: CAFCR can be applied recursively

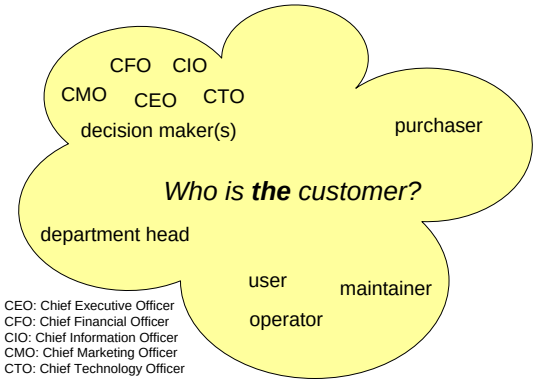


Figure 14.4: Which person is **the** customer?

14.4 Life Cycle view

The basic CAFCR model relates the customer needs to design decisions. However, in practice we have one more major input for the system requirements: the life cycle needs. Figure 14.5 shows the CAFCR+ model that extends the basic CAFCR model with a *Life Cycle view*.

The system life cycle starts with the conception of the system that trigger the development. When the system has been developed then it can be reproduced by manufacturing, ordered by logistics, installed by service engineers, sold by sales representatives, and supported throughout its life time. Once delivered every produced system goes through a life cycle of its own with scheduled maintenance, unscheduled repairs, upgrades, extensions, and operational support. Many stakeholders are involved in the entire life cycle: sales, service, logistics, production,

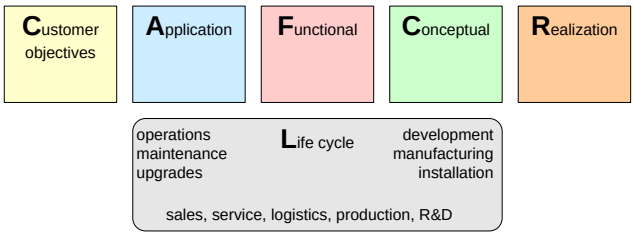
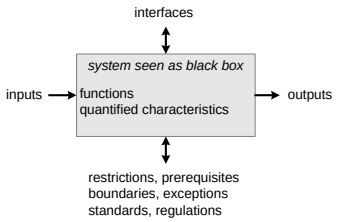


Figure 14.5: CAFCR+ model; Life Cycle View

R&D. Note that all these stakeholders can be part of the same company or that these functions can be distributed over several companies.

Chapter 15

Fundamentals of Requirements Engineering



15.1 Introduction

The basis of a good system architecture is the availability and understanding of the needs of all stakeholders. Stakeholder needs are primary inputs for the system specification. The terms *requirements elicitation*, *requirements analysis*, and *requirements management* are frequently used as parts of the Product Creation Process that cope with the trandormation of needs into specification and design.

15.2 Definition of Requirements

The term requirement is quite heavily overloaded in Product Creation context. *Requirement* is sometimes used non-obligatory, e.g. to express wants or needs. In other cases it used as mandatory prescription, e.g. a must that will be verified. Obviously, dangerous misunderstandings can grow if some stakeholders interpret a requirement as want, while other stakeholders see it as must.

We will adopt the following terms to avoid this misunderstanding:

Customer Needs The term *Customer Needs* is used for the non-mandatory wishes, wants, and needs.

Product Specification The term *Product Specification* is used for the mandatory characteristics the system must fulfill.

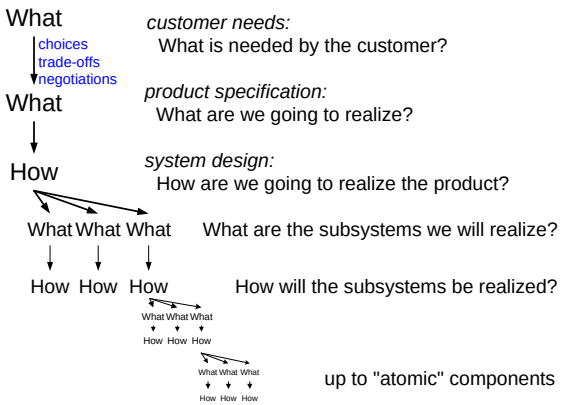


Figure 15.1: The flow of requirements

In the system engineering world the term *Requirements Management* or *Requirements Engineering* is also being used. This term goes beyond the two previous interpretations. The requirements management or engineering process deals with the propagation of the requirements in the product specification towards the requirements of the atomic components. Several propagation steps take place between the product specification and atomic components, such as requirements of the subsystems defined by the first design decomposition. In fact the requirement definition is recursively applied for every decomposition level similar to the product specification and subsystem decomposition.

Figure 15.1 shows the requirements engineering flow. The customer needs are used to determine the product specification. Many choices are made going from needs to specification, sometimes by negotiation, sometimes as trade-off. Often the needs are not fully satisfied for mundane reasons such as cost or other constraints. In some cases the product specification exceeds the formulated needs, for instance anticipating future changes.

Figure 15.1 also show the separation of specification, *what*, and design, *how*. This separation facilitates clear and sharp decision making, where goals *what* and means *how* are separated. In practice decision are often polluted by confusing goals and means.

An other source of requirements is the organization that creates and supplies the product. The needs of the organization itself a nd of the supply and support chain during the life cycle are described in this chapter as *Life Cycle Needs*.

15.3 System as a black box

One of the main characteristics of requirements in the product specification is that they describe *what* has to be achieved and not it how this has to be achieved. In other words, the product specification describes the system as *black box*. Figure 15.2 provides a starting point to write a product specification.

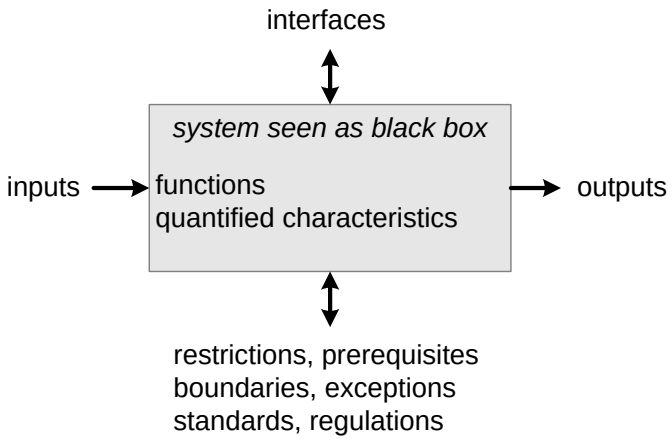


Figure 15.2: System as a Black Box

The system is seen as black box. What goes into the box, what comes out and what functions have to be performed on the inputs to get the outputs. Note that the functions tell something about the black box, but without prescribing how to realize them. All interfaces need to be described, interfaces between the system and humans as well as interfaces to other systems. The specification must also quantify desired characteristics, such as how fast, how much, how large, how costly, et cetera.

Prerequisites and constraints enforced on the system form another class of information in the product specification. Further scoping can be done by stating boundaries and desired behavior in case of exceptions. Regulations and standards can be mandatory for a system, in which case compliance with these regulations and standards is part of the product specification.

15.4 Stakeholders

A simplified process model is shown in figure 15.3. The stakeholders of the product specification are of course the customers, but also people in the Customer Oriented Process, the Product Creation Process, People, Process, and Technology

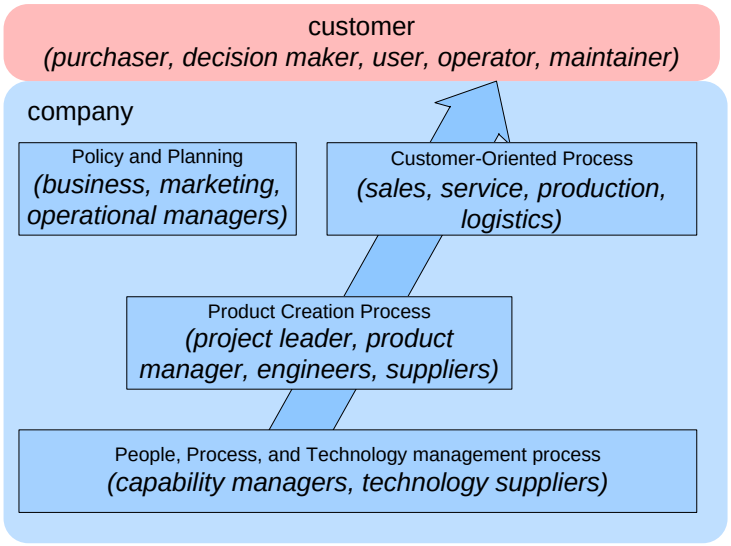


Figure 15.3: A simplified process decomposition of the business. The stakeholders of the requirements are beside the customer self, mainly active in the customer oriented process and the product creation process.

Management Process, and the Policy and Planning Process. The figure gives a number of examples of stekholders per process.

The customer can be a consumer, but it can also be a business or even a group of businesses. Businesses are complex entities with lots of stakeholders. A good understanding of the customer business is required to identify the customer-stakeholders.

15.5 Requirements for Requirements

Standards like ISO 9000 or methods like CMM prescribe the requirements for the requirements management process. The left side of Figure 15.4 shows typical requirements for the requirements itself.

Specific , what is exactly needed? For example, The system shall be *user friendly* is way too generic. Instead a set of specific requirements is needed that together will contribute to user friendliness.

Unambiguous so that stakeholders don't have different expectations on the outcome. In natural language statements are quite often context sensitive, making the statement ambiguous.

Verifiable so that the specification can be verified when realized.

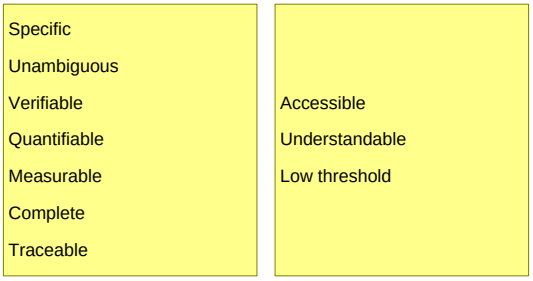


Figure 15.4: Requirements for Requirements

Quantifiable is often the way to make requirements verifiable. Quantified requirements also help to make requirements specific

Measurable to support the verification. Note that not all quantified characteristics can also be measured. For example in wafer steppers and electron microscopes many key performance parameters are defined in nanometers or smaller. There are many physical uncertainties to measure such small quantities.

Complete for all main requirements. *Completeness* is a dangerous criterion. In practice a specification is never complete, it would take infinite time to approach completeness. The real need is that all crucial requirements are specified.

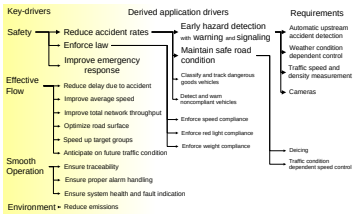
Traceable for all main relations and dependencies. *Traceability* is also a dangerous criterion. Full traceability requires more than infinite time and effort. Understanding how system characteristics contribute to an aggregate performance supports reasoning about changes and decision making.

Unfortunately, these requirements are always biased towards the formal side. A process that fulfills these requirements is from theoretical point of view sound and robust. However, an aspect that is forgotten quite often, is that product creation is a human activity, with human capabilities and constraints. The human point of view adds a number of requirements, shown at the right hand side of Figure 15.4: accessibility, understandability, and a low threshold. These requirements are required for **every** (human) stakeholder.

These requirements, imposed because of the human element, can be conflicting with the requirements prescribed by the management process. Many problems in practice can be traced back to violation of the human imposed requirements. For instance, an abstract description of a customer requirement such that no real customer can understand the requiremnts anymore. Lack of understanding is a severe risk, because early validation does not take place.

Chapter 16

Key Drivers How To



Note: the graph is only partially elaborated for application drivers and requirements

16.1 Introduction

A key driver graph is a graph that relates the key drivers (the essential needs) of the customer with the requirements in the product specification. This graph helps to understand the customer better, and the graph helps to assess the importance of requirements. The combination of customer understanding and value assessment makes the graph into an instrument to lead the project.

We will discuss one example, a Motor way management system, and we will discuss a method to create a customer key driver graph.

16.2 Example Motor Way Management

In this section we discuss an example from practice. The graph discussed here was created in 2000 by a group of marketing managers and systems architects. Creating this version took a few days. Note that we only show and discuss a small part of the entire graph to prevent overload.

Figure 16.1 shows an example of a key driver graph of a motor way management system. A motor way management system is a system that provides information to traffic controllers, and it allows traffic controllers to take measures on the road

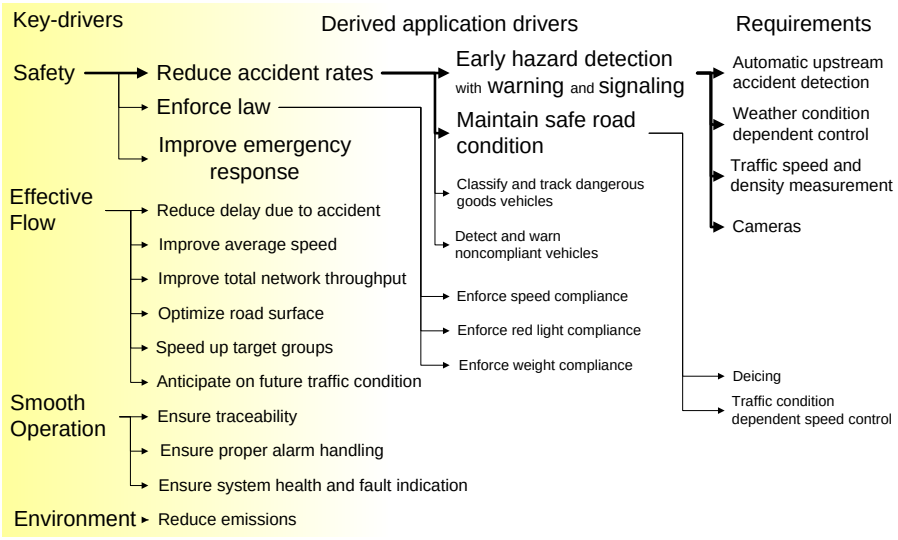


Figure 16.1: The key driver graph of a Motor way Management System

or to inform drivers on the road. As driver we typically see electronic information and traffic signs that are part of these systems. Also the cameras along the road are part of such system.

The key drivers of a motor way management owner are:

Safety for all people on the road: drivers and road maintainers.

Effective Flow of the traffic.

Smooth Operation of the motor way management.

Environment such as low emissions.

To realize these key drivers the owner applies a number of application processes. For example the traffic controllers can improve safety by reducing the accident rate. The accident rate can be reduced by detecting hazards and warning drivers about the hazards. Examples of hazards are accidents that already have happened and in turn may trigger new accidents. Another example of a hazard are bad weather conditions. Hence the automatic detection of accidents and controls that are weather dependent will help to cope with hazards, and hence will reduce accident rates and improve the safety.

Note that the 4 key drivers shown here are the key drivers of the motor way management system. Other systems will also share these concerns, but might not have these as key drivers. For example, smart phones will have a completely different set of key drivers. Do not use this example as template for your own key driver graph, because it biases the effort.

16.3 CAF-views and Key Drivers

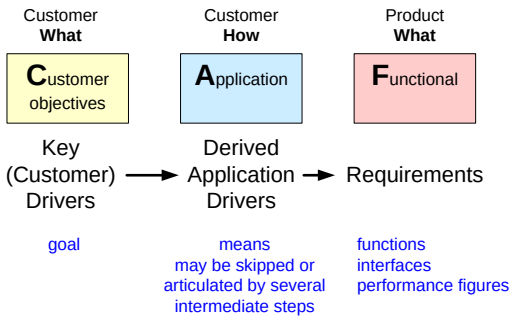


Figure 16.2: The flow from Key Drivers via derived application drivers to requirements

We can capture the essence of the customer world in the *Customer Objectives* view of the CAFCR model by means of customer key drivers. The customer will organize the way of working such that these key drivers are achieved. Figure 16.2 shows how the key drivers as part of the *Customer Objectives* view are supported by application drivers. The application drivers are means to satisfy the customer key drivers. These application drivers in turn will partially be fulfilled by the system-of-interest. Appropriate requirements, e.g. specific functions, interfaces or performance figures, of the system-of-interest will help the customer to use the system to satisfy their customer key drivers. The key drivers are one of the submethods in the Customer Objectives view.

Figure 16.3 shows a method to define key drivers.

Define the scope specific . Identify a specific customer and within the customer a specific stakeholder to make the graph. Choosing a customer implies choosing a market segment. A narrow well defined scope results in a more clear understanding of the customer. The method can be repeated a few times to understand other customers/stakeholders. Products normally have to serve a class of customers. A common pitfall is that the project team too early "averages" the needs and by averaging compromises the value for

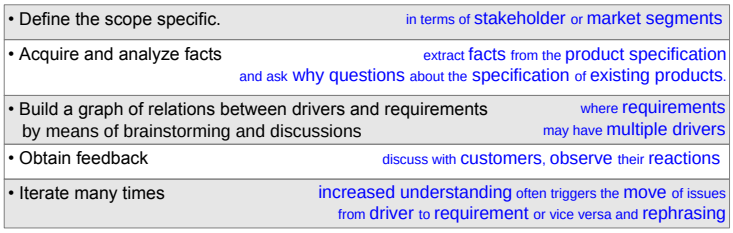


Figure 16.3: Method to define key drivers

specific customers. We recommend to first create some understanding of the target customers before any compromising takes place.

Acquire and analyze facts We recommend to start building the graph by looking for known facts. For example, in most organizations there is already an extensive draft product specification, with many proposed requirements. For every requirement in the draft specification the *why* question can be asked: "Why does the customer need this feature, what will the customer do with this feature?". Repeating the *why* question relates the requirement in a few steps to a (potential) key driver.

Note that starting with facts often means working bottom-up[1]. When marketing and application managers have a good understanding of the customer, then the facts can also be found in the CA-views, allowing a more top-down approach. Iteration, repeated top-down and bottom-up discussions, is necessary in either case.

Build a graph of relations between drivers and requirements by means of brainstorms and discussions. A great deal of the value of this method is in this discussion, where team members create a shared understanding of the customer and the product specification. Note that the graph is often many-to-many: one requirement can serve multiple key drivers, and one key driver results in many different requirements.

Obtain feedback from customers by showing them the graph and by discussing the graph. Note that it is a good sign when customers dispute the graph, since the graph in that case apparently is understandable. When customers say that the graph is OK, then that is often a bad sign, mostly showing that the customer is polite.

Iterate many times top-down and bottom-up. During these iteration it is quite normal that issues move left to right or opposite due to increased under-

[1]Every time that course participants ignore this recommendation, and start top-down while lacking customer insight, they come up with a set of too abstract not usable key drivers.

standing. It is also quite normal that issues are rephrased to sharpen and clarify.

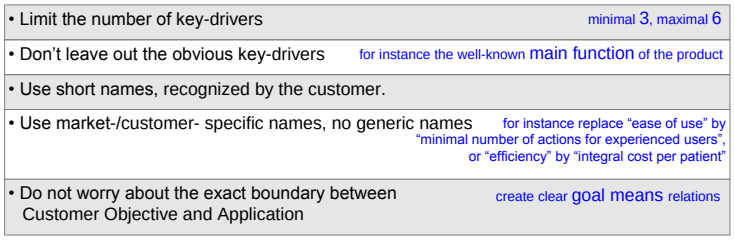


Figure 16.4: Recommendations when defining key drivers

Figure 16.4 shows some recommendations with respect to the definition of key drivers.

Limit the number of key drivers to maximal 6 and minimal 3. A maximum of 6 Key Drivers is recommended to maintain focus on the essence, the name is on purpose **Key** driver. The minimum (three) avoids oversimplification, and it helps to identify the inherent tensions in the customer world. In real life we always have to balance objectives. For example, we have a strong need to maximize safety and performance, while at the same time we will have cost pressure. A good set of key drivers captures also the main tensions from customer perspective.

Do not leave out the obvious key drivers such as the main function of the product. For example, the communication must be recognizable when discussing smart phones; the focus might be on all kinds of innovative features and services, while the main function is forgotten.

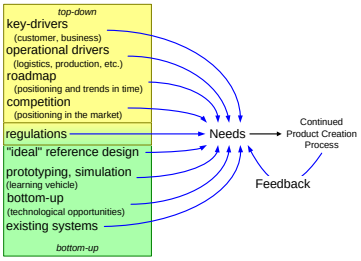
Use short names, recognized by the customer. Key drivers must be expressed in the language of the customer so that customers recognize and understand them. The risk in teams of engineers is that the terminology drifts away and becomes too abstract or too analytical. Another risk is that descriptions or sentences are used in the graph to explain what is meant. These clarifying texts should not be in the graph itself, because the overview function of the graph gets lost. The challenge is to find short labels that resonate with customers.

Use market/customer specific names, no generic names . The more specific a name or label is, the more it helps in understanding. Generic names facilitate the "escape" of diving into the customer world. For example, the term *ease of use* is way too much of a motherhood statement. Instead *minimal number of actions (for experienced users)* might be the real issue.

Allocation to Customer Objectives or Application View Do not worry about the exact boundary between Customer Objective and Application The purpose of the graph is to get a clear separation of goals and means, where goals and means are recursive: an application driver is a means to achieve the customer key driver, and at the same time it is a goal for the functions of the system of interest. Sometimes we need five steps to relate customer key drivers to requirements, sometimes the relation is obvious and is directly linked. The CAFCR model is a means to think about the architecture, it is not a goal to fit everything right in the different views!

Chapter 17

Requirements Elicitation and Selection



17.1 Introduction

The quality of the system under development depends strongly on the quality of the elicitation process. We can only make a fitting system when we understand the needs of our customer. The outcome of an elicitation process is often an overload of needs. We need a selection process to balance what is needed with all kinds of constraints, such as cost, effort, and time.

17.2 Viewpoints on Needs

Needs for a new product can be found in a wide variety of sources. The challenge in identifying needs is, in general, to distinguish a solution for a need from the need itself. Stakeholders, when asked for needs, nearly always answer in terms of a solution. For example, consumers might ask for a *flash based video recorder*, where the underlying need might be a light-weight, small, portable video recorder. It is the architect's job, together with marketing and product managers, to reconstruct the actual needs from the answers that stakeholders give.

Many complementary viewpoints provide a good collection of needs. Figure 17.1 shows a useful number of viewpoints when collecting needs.

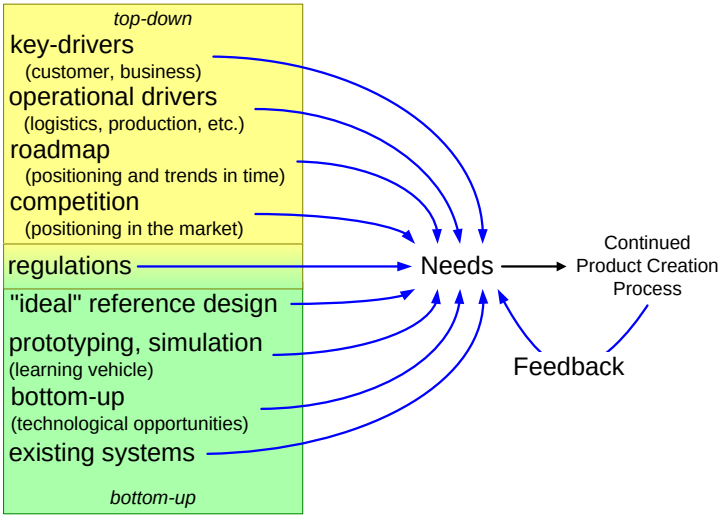


Figure 17.1: Complementary viewpoints to collect needs

The **key-driver** viewpoint and the **operational** viewpoint are the viewpoints of the stakeholders who are “consuming” or “using” the output of the Product Creation Process. These viewpoints represent the "demand side".

The **roadmap** and the **competition** viewpoints are viewpoints to position the products in time and in the market. These viewpoints are important because they emphasize the fact that a product is being created in a dynamic and evolving world. The product context is not static and isolated.

Regulations result in requirements both top-down, as well as bottom-up. A top down example are labor regulations that can have impact on product functionality and performance. A bottom up example are materials regulations, for instance do not use lead, that may strongly influence design options.

The **“ideal” reference design** is the challenge for the architect. What is in the architect’s vision the perfect solution? From this perfect solution the implicit needs can be reconstructed and added to the collection of needs.

Prototyping or simulations are an important means in communication with customers. This “pro-active feedback” is a very effective filter for nice but impractical features at the one hand and it often uncovers many new requirements. An approach using only concepts easily misses practical constraints and opportunities.

The **bottom up** viewpoint is the viewpoint where the technology is taken as the starting point. This viewpoint sometimes triggers new opportunities that have been overlooked by the other viewpoints due to an implicit bias towards today’s technology.

The **existing system** is one of the most important sources of needs. In fact it contains the accumulated wisdom of years of practical application. Especially a large amount of small, but practical, needs can be extracted from existing systems.

The product specification is a dynamic entity, because the world is dynamic: the users change, the competition changes, the technology changes, the company itself changes. For that reason the **Continuation of the Product Creation Process** will generate input for the specification as well. In fact nearly all viewpoints are present and relevant during the entire Product Creation Process.

17.3 Requirements Value and Selection

The collection of customer and operational needs is often larger than can be realized in the first release of a product. A selection step is required to generate a product specification with the customer and operational needs as input. Figure 17.2 shows the selection process as black box with its inputs and outputs.

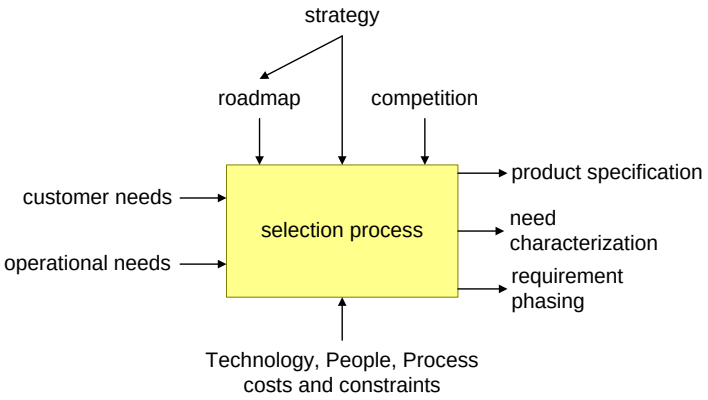


Figure 17.2: The selection process produces a product specification and a phasing and characterization of requirements to prevent repetition of discussion

The selection process is primarily controlled by the strategy of the company. The strategy determines market, geography, timing and investments. The roadmap, based on the strategy, is giving context to the selection process for a individual products. The reality of the competitive market is the last influencing factor on the selection.

The selection will often be constrained by technology, people, and process. The decisions in the selection require facts or estimates of these constraints.

During the selection a lot of insight is obtained in needs, the value of needs, and the urgency. We recommend to consolidate these insights, for example by

documenting the characterization of needs. The timing insights can be documented in a phased plan for requirements.

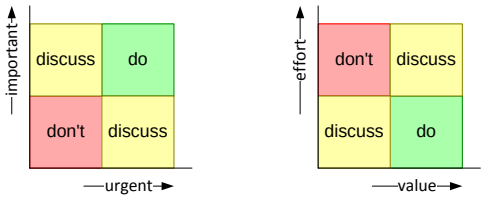


Figure 17.3: Simple methods for a first selection

The amount of needs can be so large that it is beneficial to quickly filter out the“obvious” requirements. For some needs it is immediately obvious that they have to be fulfilled anyway, while other needs can be delayed without any problem. Figure 17.3 shows a number of qualitative characterizations of needs, visualized in a two-dimensional matrix. For every quadrant in the matrix a conclusion is given, a need must be included in the specification, a need has to be discarded or the need must be discussed further.

This simple qualitative approach can, for instance, be done with the following criteria:

- importance versus urgency
- customer value versus effort

In the final selection step a more detailed analysis step is preferable, because this improves the understanding of the needs and results in a less changes during the development.

A possible way to do this more detailed analysis is to “quantify” the characteristics for every requirement for the most business relevant aspects, see for examples Figure 24.10.

These quantifications can be given for the immediate future, but also for the somewhat remote future. In that way insight is obtained in the trend, while this information is also very useful for a discussion on the timing of the different requirements. In [4] a much more elaborated method for requirement evaluation and selection is described.

The output of the requirement characterization and the proposed phasing can be used as input for the next update cycle of the roadmap.

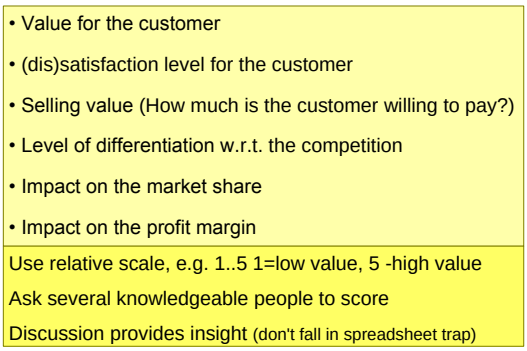
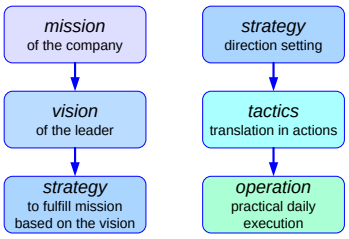


Figure 17.4: Quantifiable Aspects for Requirements Selection

Chapter 18

Business Strategy; Methods and Models



18.1 Introduction

The business strategy is input to many activities of architects. Lack of clear strategy complicates the work of architects. At the other hand architects need to contribute to the creation and evolution of the business strategy.

The "strategy world" is full of concepts. We will provide a few simple models to position and explain these concepts. There is also an extensive amount of methods and techniques to create and evolve a strategy. We discuss a few methods and techniques that fit in the architecting contribution.

18.2 Basic Concepts

Nowadays companies foster an identity of the company by formulating a *mission*. The mission can be supported by the articulation of typical four company values. The company identity is used for branding: what is the image of the company, how is the company perceived by the market, its customers, and its shareholders. The mission and company values tend to be very generic, providing a direction to managers and employees. The *mission* is shown at the top in Figure 18.1.

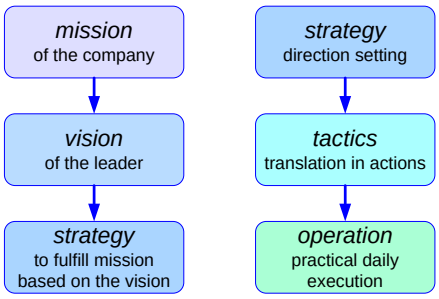


Figure 18.1: Some Basic Concepts

The leaders in the company formulate a *vision*: what value can the company bring to the world, what role can the company play. The vision tends to be more market domain specific and will evolve over time.

A true vision is a powerful instrument, uniting the company employees by a shared vision. Unfortunately, too many visions are the result of a mechanistic process. The creation of a vision depends on leaders with the ability to combine a huge amount of context data in a sensible picture. A poor vision might result in ghost hunting or lack of cohesion in the organization.

The mission and vision set the scope for the strategy: where does the company want to go and why. At the right side of Figure 18.1 a often used layering is shown of *strategy*, *tactics*, and *operation*.

The *tactics* is an elaboration of the strategy, how can the strategy best be achieved. For example, do we start with top-of-the-line systems, followed by cost reduced systems, or vice versa.

The *operations* focus on the execution: get things done. Typically the operations has a fast hear beat, where resources and activities are managed continuously and deviations or problems are resolved as soon as possible.

Systems architects will often get the mission and company values as given. They will work using mission and values as guiding principles. Architects might be involved in the creation and evolution of the vision. System Architects should be involved in the strategy creation and evolution. They are typically involved in the tactics. A significant amount of the architect's time is spend in the operational aspects of product creation.

Figure 18.2 shows the "BAPO" framework developed at Philips Research by Henk Obbink. The *Business* needs drive the *Architecture*. Ideally, the *Process* and *Organization* should be designed to facilitate the creation of the *Architecture*. In practice often the opposite is happening: the *Organization* structure is superimposed on the *Architecture*. In other words, we compromise the *Architecture* to fit in the existing *Organization*. The room for *Organization* changes triggered by the *Architecture* is limited since *Organization* changes tend to be slow. The conse-

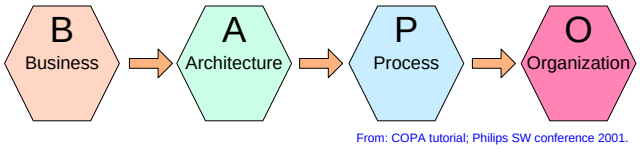


Figure 18.2: BAPO framework

quence for architecting is that *Process* and *Organization* are part of the playing field. *Process* and *Organization* should not be seen as fixed entities.

18.3 Methods for Strategy Support

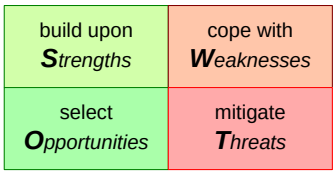


Figure 18.3: SWOT analysis

One of the methods that is frequently used when creating or evolving a strategy is a SWOT-analysis, see Figure 18.3, where the letters stand for:

Strengths of the own organization, including technology and market position, where the organization can build on.

Weaknesses of the own organization, where the organization has to cope with these weaknesses. Note that acknowledgment of a weakness and relying on outside support is a legitimate way to cope with weaknesses.

Opportunities in the world where the organization can benefit of their current strengths. Opportunities have to be identified, assessed, and finally a subset has to be selected to pursue.

Threats in the world, e.g. from changing markets or regulations, or from upcoming competition. Threats have to be identified and assessed, and, when serious, counter measures need to be formulated.

The SWOT analysis results in a “big picture” of the current situation that can be used as starting point for the formulation for a strategy.

One of the strategic choices is what a company will do itself and when it will rely on suppliers. There is spectrum of possibilities, from create and make it self,

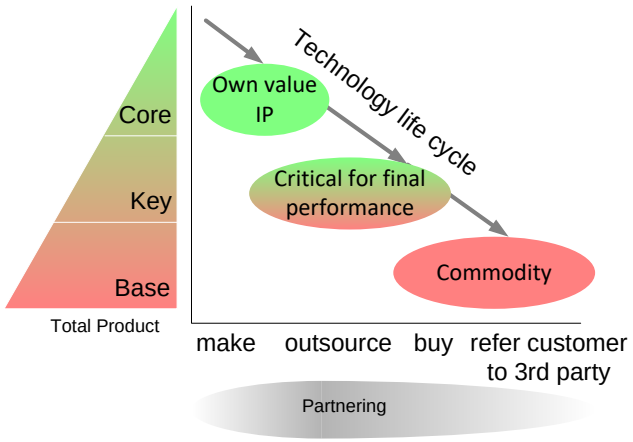


Figure 18.4: Core, Key or Base technology

via outsourcing, to buy. Figure 18.4 shows a technology classification model to reason about these choices. The decision how to obtain the needed technology should be based on where the company intents to add value. The technology classification model uses *core*, *key*, and *base* technology:

Core technology is technology where the company is adding value. In order to be able to add value, this technology should be developed by the company itself.

Key technology is technology which is critical for the final system performance. If the system performance can not be reached by means of third party technology than the company must develop it themselves. Otherwise outsourcing or buying is attractive, in order to focus as much as possible on *core* technology added value. However when outsourcing or buying an intimate partnership is recommended to ensure the proper performance level.

Base technology is technology which is available on the market and where the development is driven by other systems or applications. Care should be taken that these external developments can be followed. Own developments here are de-focusing the attention from the company's core technology.

18.4 Examples of strategic choices

Figure 18.5 shows a list of *business models*. Every business model has specific characteristics in terms of capital use, return on investment, recurring revenues, variability over time, and margin. At the other hand will the business model have

Pay for product
Pay for accessories (cell phone, MP3 cases, skins, etc.)
Pay per use (per printed page, per accessed image)
Pay for service (imaging, printing)
Pay for capability (diagnosis, booklet)
Pay as part of subscription (telecom)
Pay for content (music, movies, eBooks)
Pay for consumables (ink, toner)
Advertizing company pays (Google)
Insurance pays (health care)

Figure 18.5: Examples of Business Models

significant impact on the product specification, design choices, organization, staff, and processes.

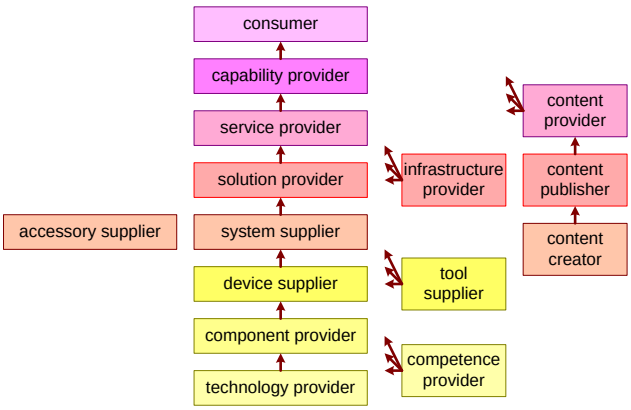


Figure 18.6: Where in the Value Chain?

The position in the value chain is also a strategic choice. Figure 18.6 shows an example of value chain. Companies that stay at the same position in the value chain must protect their margin by excellence in that position. The risk is that "lower" positions in the value chain get commoditized, meaning that the margin gets small or negative. Many organizations address this margin problem by trying to rise in the value chain or by expansion in the value chain.

The choice of the business model and the position in the value chain are primarily business decisions. However, these decisions do have such large impact on the architecting that architects should be involved in the decision making. The consequence for the architects is that they have to participate in a largely financial and economical discussion about the business.

18.5 Innovation

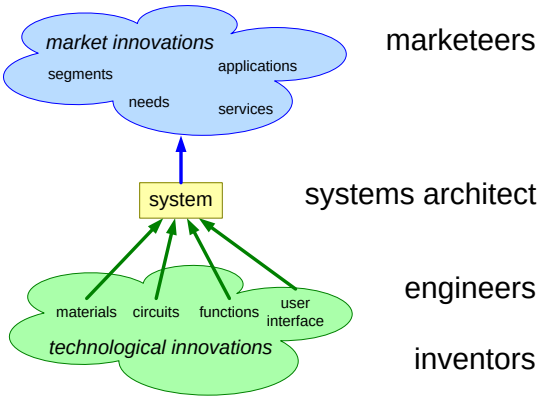


Figure 18.7: Innovation requires all major contributors

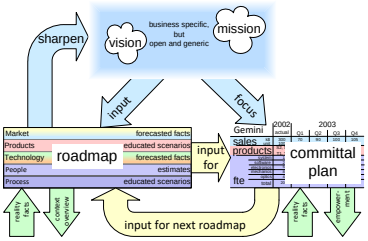
In many organizations the holy grail of strategy is *innovation*. *Innovation* is is a fundamental way to increase the value proposition to the market. Companies have a continuous need for a better value proposition in a world with constant pressure on the margin. The alternative to maintain the margin at an healthy level is to reduce cost levels.

Most (mature) organizations achieve the desired improvement of the value proposition by repetitive small improvement steps. However, many small steps often do not open new markets, or create new applications. *Innovation* is the result of a creative effort both in the technology side, as well as the application and marketing side. Figure 18.7 shows that a concerted effort is needed of truly innovative technology people ("inventors"), engineers, architects, and marketeers.

There is a tension between processes and management and innovation. The inherent nature of innovation is to go beyond today's limitations, while processes and management also tend to enforce limitations. Innovation requires inspiration rather than control. This same tension can also be observed in the architecting role. Many architects are used to identify and mitigate risks, a valuable contribution to product creation. However, the risk based focus can be a severe limitation when searching for innovative solutions.

Chapter 19

The role of roadmapping in the strategy process



19.1 Process decomposition of a business

The business process for an organization which creates and builds systems consisting of hardware and software is decomposed in 4 main processes as shown in figure 19.1.

The decomposition in 4 main processes leaves out all connecting supporting and other processes. The function of the 4 main processes is:

Customer Oriented Process This process performs in repetitive mode all direct interaction with the customer. This primary process is the cashflow generating part of the enterprise. All other processes only spend money.

Product Creation Process This Process feeds the Customer Oriented Process with new products. This process ensures the continuity of the enterprise by creating products which enables the primary process to generate cashflow tomorrow as well.

People and Technology Management Process Here the main assets of the company are managed: the know how and skills residing in people.

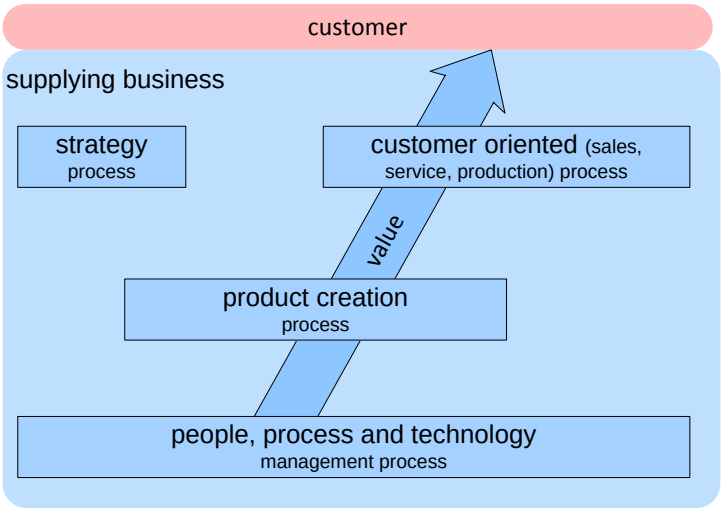


Figure 19.1: Simplified decomposition of the business in 4 main processes

Policy and Planning Process This process is future oriented, not constrained by short term goals, it is defining the future direction of the company by means of roadmaps. These roadmaps give direction to the Product Creation Process and the People and Technology Management Process. For the medium term these roadmaps are transformed in budgets and plans, which are committal for all stakeholders.

Figure 19.2 characterizes the processes from the financial point of view. From bottom to top soft or latent value (the assets) are transformed in harder value, to become true money when the customers are paying for the products and services (the cashflow).

At the same time figure 19.2 shows that the feedback flow from the customer into the organization moves in the opposite direction. A nasty phenomenon is the deformation and loss of feedback information while it flows through these processes. The further away from the customer, the less sense of urgency and the less know how of the customer needs. In many organizations this is a significant problem: competence organizations which have lost the sight of the customer and become introvert.

In many companies the value chain is optimized further, by using the synergy between products and product families. Figure 19.3 shows that the simplified process decomposition model can be extended by one process *component or platform creation* to visualize this strategy. This optimization is far from trivial. At the one hand synergy must be used, most companies cannot afford to create everything from scratch all the time. At the other hand is the consequence of the set up shown here that the value chain becomes longer (and takes somewhat longer),

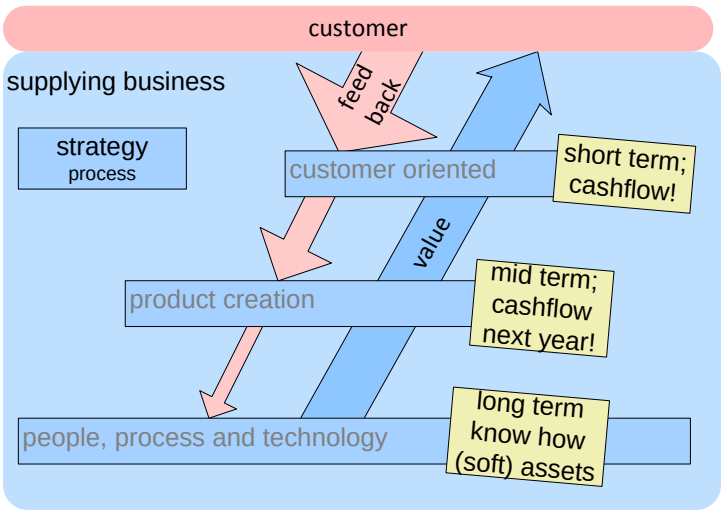


Figure 19.2: Tension between processes

while the feedback deformation and loss increases even further! A more elaborated discussion on these aspects can be found in [13].

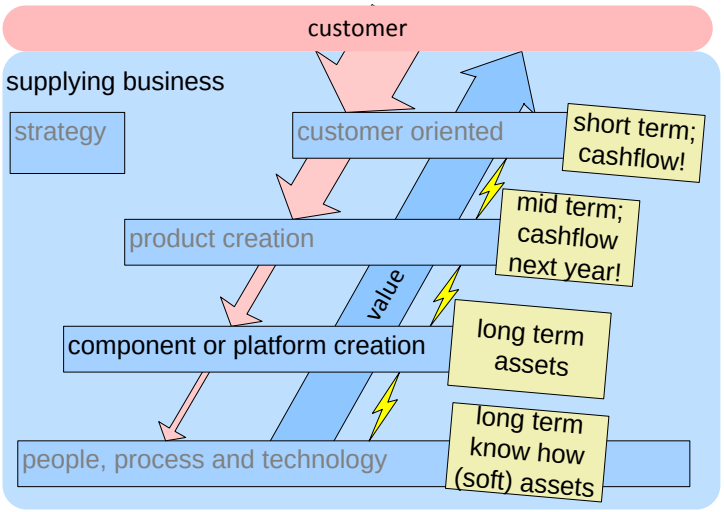


Figure 19.3: Platform strategy adds one layer

19.2 Framework for architecting and roadmapping

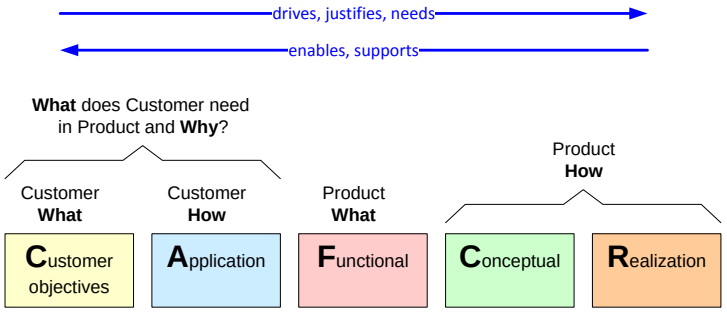


Figure 19.4: CAFCR framework for architecting

Figure 19.4 shows the ”CAFCR” framework for system architecting, see [18]. The *customer objectives* view and the *application* view provide the **why** from the customer. The *functional* view describes the **what** of the product, which includes (despite the name) also the *non functional* requirements. The **how** of the product is described in the *conceptual* and *realization* view, where the conceptual view is changing less in time than the fast changing realization (Moore's law!).

The job of the architect is to integrate these views in a consistent and balanced way. Architects do this job by *frequent viewpoint hopping*, looking at the problem from many different viewpoints, sampling the problem and solution space in order to build up an understanding of the business. Top down (objective driven, based on intention and context understanding) in combination with bottom up (constraint aware, identifying opportunities, know how based), see figure 19.5.

In other words the views must be used concurrently, not top down like the waterfall model. However at the end a consistent story must be available, where the justification and the needs are expressed in the customer side, while the technical solution side enables and support the customer side.

The term *customer* is easily used, but it is far from trivial to determine the customer. The position in the value chain shows that multiple customers are involved. In figure 19.6 the multiple customers are addressed by applying the CAFCR model recursively.

The customer is a gross generalization. Marketing managers make a classification of customers by means of a market segmentation. Nevertheless stay aware of the level of abstraction used when discussing **the** customer/market/market segment.

The viewpoints of the ”CAFCR” framework are useful for setting up a roadmap as well. However on top of these views also *business*, *people* and *process* views are needed in a roadmap, see figure 20.1 and [15].

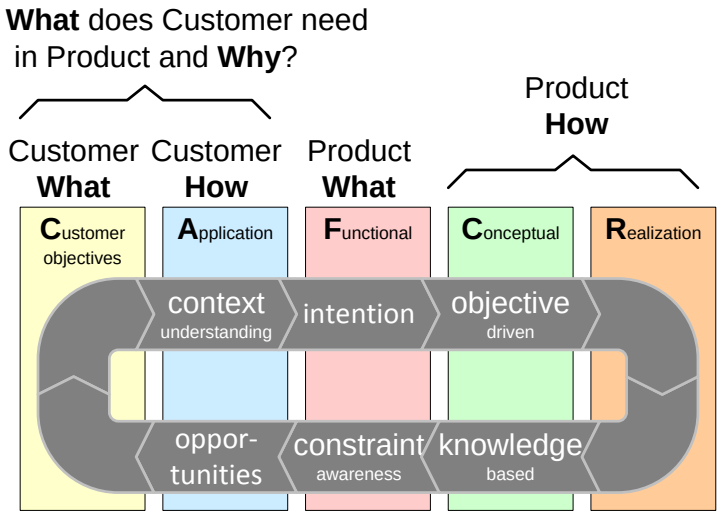


Figure 19.5: Five viewpoints for an architecture. The task of the architect is to integrate all these viewpoints, in order to get a *valuable*, *usable* and *feasible* product.

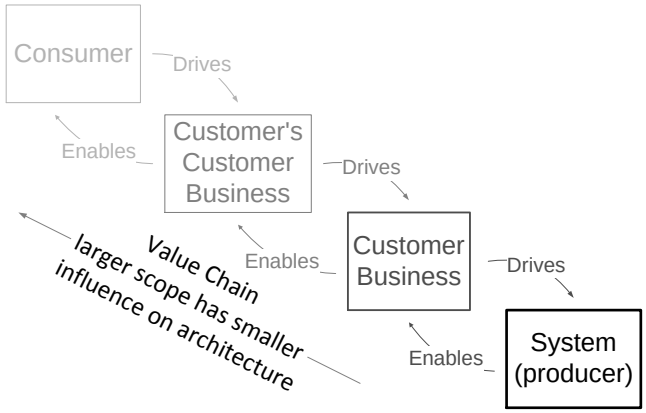


Figure 19.6: CAFCR can be applied recursively

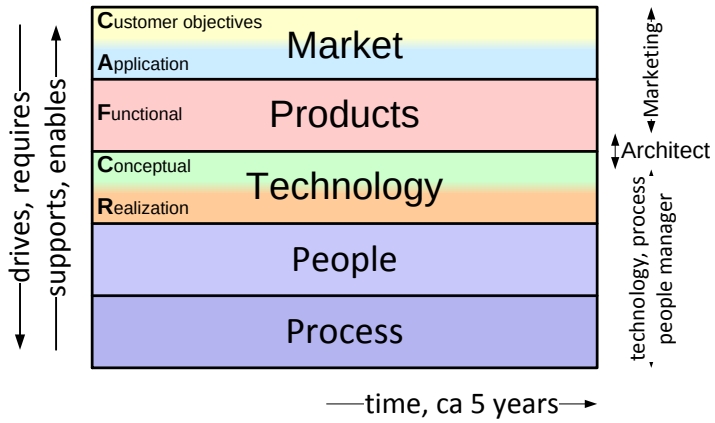


Figure 19.7: Structure of a roadmap

19.3 From vision to roadmap to plan and further

The identity or the main focus of a company is often expressed in a mission statement, supported by a vision on the market, the domain and its own position in market and domain. The nature of both mission and vision is highly generic, although business specific. Mission and vision is a compact articulation of the company and its strategy.

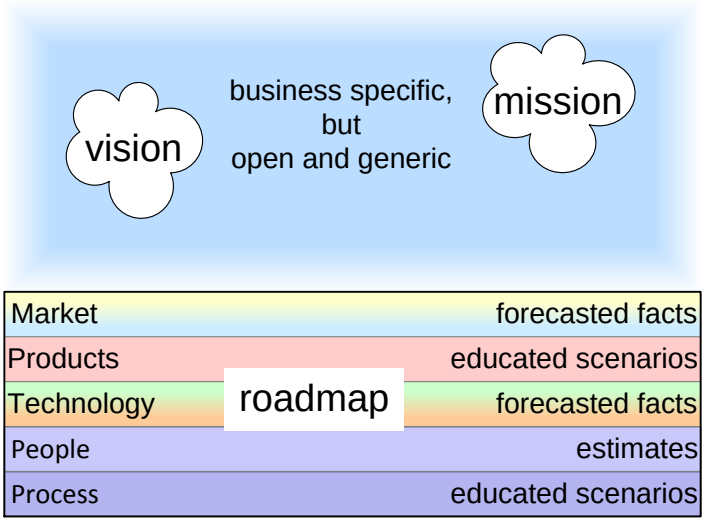


Figure 19.8: From generic mission to factual roadmap

The roadmap builds on vision and mission and makes the strategy much more specific in time as well as in contents. Figure 19.8 shows the generic mission and vision statement as overarching entities for the roadmap. As indicated within the roadmap segments its content is much more specific, containing (forecasted) facts, (educated) scenarios and estimates.

An integrated roadmap is made in steps:

1. Explore *market*, *product* and *technology* segments; what is happening in the outside world, what is needed, where are opportunities in market and/or technology.

2. Estimate *people* and *process* needs for the identified *product* and *technology* needs. These estimates should be made without constraints. The question is what is **needed**, rather than what is **possible**.

3. Determine a balanced, economic attractive and skills wise feasible content for *product*, *technology*, *people* and *process*. Here trade-offs have to be made

and creative marketing as well as technological skills are required to define an effective product roadmap, which is at the same time realistic with respect to the people and processes.

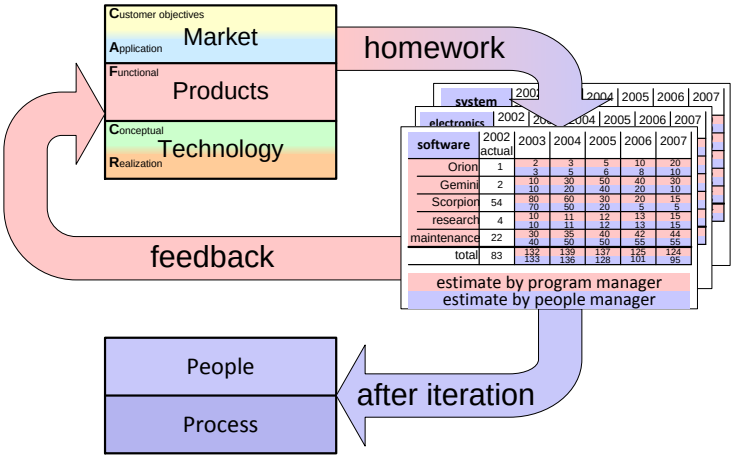


Figure 19.9: From Market, Product, Technology to People, Process

Figure 19.9 shows how to make the last few steps. The estimations for the amount of people are made from 2 viewpoints: the people and technology manager (the supplier of resources) and the operational manager (responsible for the timely and reliable result of the product creation process and hence the ”consumer” of these resources).

The people and technology manager will make estimates which are discipline specific, decomposed towards the programs, see figure 19.10

The operational manager (or program manager) will make an estimate which is program specific. A program is a cohesive set of products, where the program manager is responsible for the timely development and quality of all products within the program. This estimate will be decomposed into disciplines, see figure 19.11.

Every activity is estimated twice via this approach. In both figure 19.10 and figure 19.11 the corresponding second estimate is shown as well, in other words the results are merged. This merge immediately shows differences in interpretation of the input or differences in opinion. These differences should be discussed, so either the inputs are reiterated, resulting in a shared estimate, or the difference in opinion is analyzed and a shared estimate must be the result (although the compromise may be marked as highly uncertain)

After this ”harmonization” of the estimates the real difficult work starts, of tweaking the product program, the required features and being more creative in the solutions in order to come to a feasible roadmap. This step will change the *product* and *technology* segments, with corresponding changes in *people* and *process*.

Orion	1	2 3	3 5		10 8	20 10
Gemini	2	10 10	30 20		40 20	30 10
Scorpion	54	80 70	60 50		20 5	15 5
research	4	10 10	11 11		13 13	15 15
maintenance	22	30 40	35 50		42 55	44 55
total	83	132 133	139 136		125 101	124 95

software	2002 actual	2003	2004	2005	2006	2007
				5		
				6		
				50		
				40		
				30		
				20		
				12		
				12		
				40		
				50		
				137		
				128		

estimate by program manager

estimate by people manager

Figure 19.10: People estimate, discipline view

Figure 19.12 shows the people roadmap from another domain in a more visual format. In this example a clear growth of the staffing is visible, where for instance system and software are growing much faster than electronics. Besides these typical product creation disciplines also the *customer oriented* people and skills are shown. The decomposition choosen here is to the needed or expected education level (high, medium and low). The clear trend here is a significant growth of customer support people, while at the same time it is expected that the education level will decrease significantly[1].

If we decompose the people estimates from figure 19.12 in the operational direction then a much more dynamic picture emerges. Operational activities have a faster rhythm than disciplines. Understanding of this dynamics helps in the total balancing act required from the strategy process. Special attention should be given to the often implicit programs, such as:

- installed base management

[1]This is a quite normal trend. Young products are supported by highly skilled people, which is possible because the installed base is still small. When the installed base is growing it is difficult to find sufficient well trained people, who are motivated to work as support personnel. At the same time the cost pressure increases, which makes it economically unattractive to hire expensive support people. All together the consequence is that investments in the product and the processes are required to operate in the more mature phase with less educated customer support people.

Gemini	2002 actual	2003	2004	2005	2006	2007
system	1	2 3	4 5	5 6	4 5	3 4
software	2	10 10	30 20	50 40	40 20	30 10
electronics	5	16 12	20 18	12 16	4 12	2 6
mechanics	8	8 12	5 14	2 8	1 6	1 3
optics	4	6 6	6 6	5 5	4 4	3 3
total	20	42 43	64 63	74 75	52 47	39 26

estimates by program manager

estimates by discipline manager

Figure 19.11: People estimate, program view

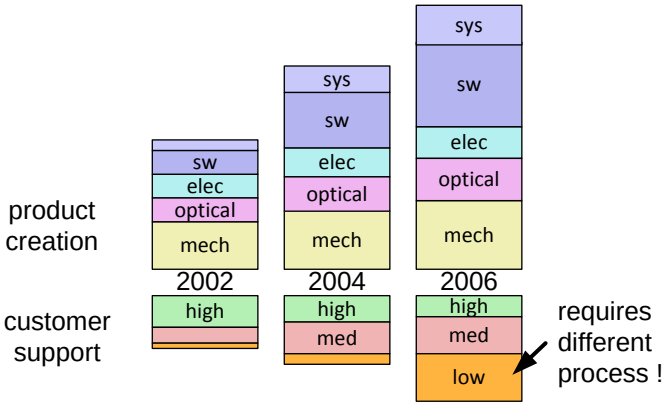


Figure 19.12: Roadmap of people skills

- component and platform creation
- research
- development infrastructure

At the end a sanity check should be made of the balance between the explicit programs and the less explicit programs mentioned here. The explicit, product oriented programs in general should use a significant amount of the total man count, otherwise it is a symptom of an introvert organization (focus on **how** do we do it, instead of **what** is needed).

The roadmap created as described above is a means to share insight in the market and the future and to provide overview and focus to the entire organization, in a broad time perspective. This process should take place in an open, explorative

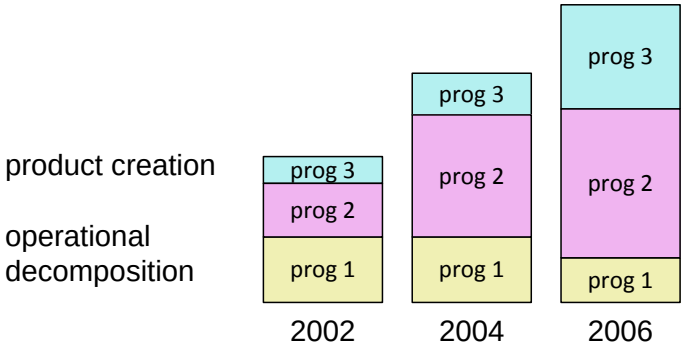


Figure 19.13: Operational axis is more dynamic

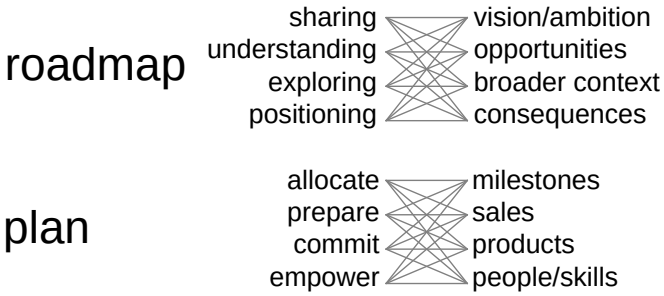


Figure 19.14: From roadmap to planning

atmosphere. This can be achieved by keeping the roadmap as a shared snapshot of the future and not make it a committal plan. In other words nobody gains any right because of the roadmap. The roadmap does not contain hard decisions, it contains shared understanding and expectations.

The roadmap is used as input to create a committal plan, with a shorter time horizon. It does not make any sense to make long term commitments, the future is way too uncertain for hard decisions. The committal plan will typically have a scope of 1 year. Within this year a consistent set of decisions are needed, ranging from sales and turnover commitments to product creation commitments (main product characteristics and timing) to technology, people and process commitments. This commitment serves also as a means to delegate and empower, which also requires allocation of resources. Figure 19.14 shows the essentials of the roadmap and the committal plan.

Figure 19.15 shows an example of a committal plan, containing the business commitments (sales), the PCP commitments (products to be created) and the people

Gemini		2002	2003			
		actual	Q1	Q2	Q3	Q4
sales	k$	300	70	90	100	105
	unit	100	20	25	25+3	22+7
products		S2, S3 T1, T4	S4		V6	S6
fte	system	1	2	3	3	4
	software	2	10	18	24	28
	electronics	5	16	17	19	20
	mechanics	8	8	8	6	6
	optics	4	6	6	6	6
	total	20	42	50	58	64

Figure 19.15: Example of committal plan

and technology commitments (allocated fte's[2]). Such a plan must be available per program, in this example it is the *Gemini* program.

19.4 Summary

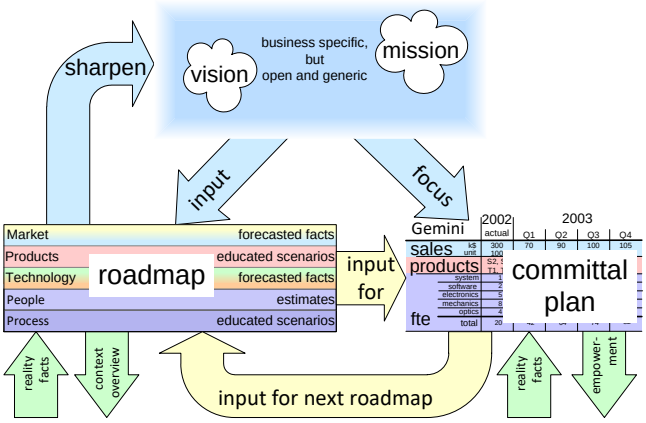


Figure 19.16: Overview of strategic entities

The mission, vision, roadmap and plan will normally be used as part of the business plan, which is used towards the financial stakeholders of the company. These entities together define the strategy and the deployment of the strategy. Figure 19.16 shows an overview of the entities which play a role in the startegy process.

The value of roadmap for the other processes is to provide context and overview for the specific goal of that process. Especially for the product creation process it

[2] fte = full time equivalents

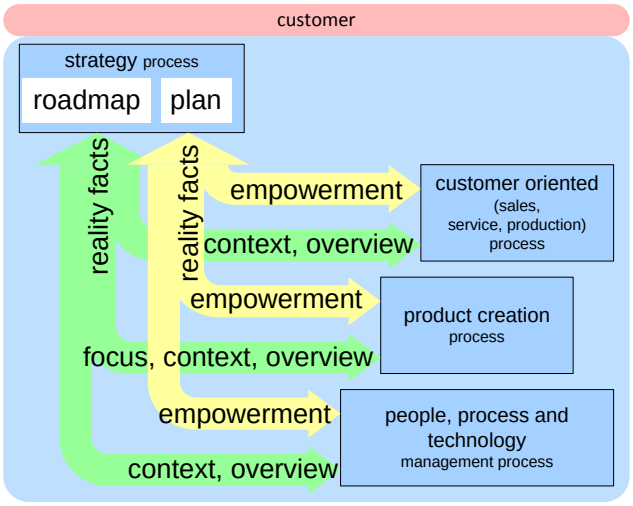


Figure 19.17: Summary of role in business

also provides focus, the development team can concentrate on the product, which is currently being developed, without discussions of all other alternatives.

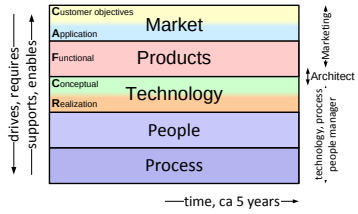
The value of the plan for the other processes is that it provides the delegation boundaries, which allows for empowerment. Figure 19.17 shows the value of roadmap and plan for the other processes. In the opposite direction the other processes should provide the reality facts to be used in next roadmap and plan.

19.5 Acknowledgements

Philip Bucher asked me for this presentation for the GATIC workshop, which provided me with the right trigger to write this already long ago planned article. Philip also helped by discussing the purpose and content.

Chapter 20

Roadmapping



20.1 Introduction

The definition of new products is a difficult activity, which frequently ends in a stalemate: “It must be don” versus “It is impossible to realize in such a short time frame”. The root cause of this frustrating stalemate is most often the fact that we try to solve a problem in a much too limited scope. Roadmapping is a method to prevent these discussions by lifting the discussion to a wider scope: from single product to product portfolio and from a single generation of products to several generations in many years.

The roadmap is the integrating vision shared by the main stakeholders. A shared vision generates focus for the entire organization and enables a higher degree of cooperating concurrent activities.

We discuss what a roadmap is, how to create and maintain a roadmap, the involvement of the stakeholders and gives criteria for the structure of a roadmap.

20.2 What is in a roadmap?

A roadmap is a visualization of the future (for example 5 years) integrating all relevant business aspects. Figure 20.1 shows the typical contents of a roadmap. At the right hand side the owner of the view is shown, while the left hand side shows

the asymmetry of the views: the market is driving, while technology people and process are enabling.

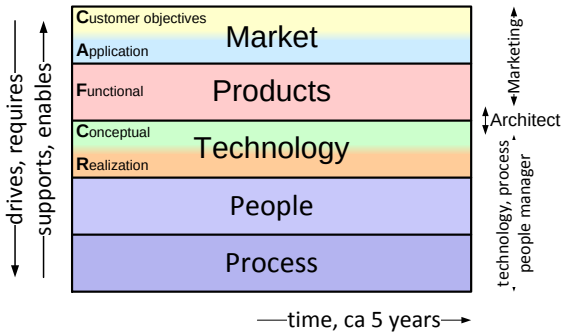


Figure 20.1: The contents of a typical roadmaps

Key to a good roadmap is the skill of showing the important, relevant issues. The roadmap should provide an immediate insight in the most relevant developments from the 5 mentioned points of view. These issues are primarily related by the time dimension.

The convention used in this article is to show products, technologies, people or process when they are or should be available. In other words the convention is to be extrovert, be oriented to the outside world. The introvert aspect, when and how to achieve these items, are not directly shown. This information is often implicitly present, since people and process often have to be available before the availability of the technology, and technology often precedes the product.

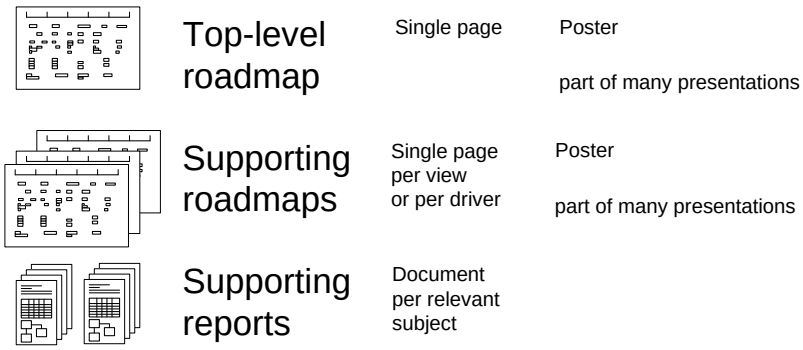


Figure 20.2: The roadmap is documented at several levels of detail

A good roadmap is documented and presented at top level and at a secondary level with more details. Figure 20.2 shows the desired granularity of the roadmap documentation, the secondary level is called supporting roadmaps. The top level

is important to create and maintain the overview, while the more detailed levels explain the supporting data. The choice of the decomposition into supporting roadmaps depends on the domain. Typically, the supporting roadmaps should maintain an integrated view. Examples of decomposition are:

- One supporting roadmap per key driver.
- One supporting roadmap per application area.

20.3 Why Roadmapping?

The Policy and Planning process as discussed in Chapter 1 relies heavily on roadmapping as tool. The main function of roadmapping is to provide a shared insight and overview of the business in time. This insight and overview enables the management of the 3 other processes:

- the Customer Oriented Process
- the Product Creation Process
- the People, Process, and Technology management Process

Where managing these processes means defining the charter and the constraints for these processes in terms of budgets and results: Where do we spend our money and what do we get back for it?

When no roadmapping is applied then the following problems can occur:

Frequent changes in product policy due to lack of anticipation.

Late start up of long lead activities , such as people recruitment and process change.

Diverging activities of teams due to a lack of shared vision.

Missed market opportunities , due to a too late start.

The frequent changes in the product policy are caused by the lack of time perspective. In extreme cases the planning is done with a limited time horizon of, for instance, 1 year. External events which are uncertain in time can shift into view within the limited horizon when popular and disappear again when some other hype is passing by. This effect is shown in Figure 20.3

The availability of a roadmap will help the operational management to apply a low pass filter on their decisions. The control becomes more analog rather than discrete, where the amount of people can be increased or decreased dependent on the expected delivery date, as shown in figure 20.4.

An inherent benefit of roadmapping is the anticipation, which is especially important for all long lead time aspects. Examples are technology, people and process. This is not limited to development activities only; market preparation, manufacturing and customer support also require anticipation. For example, reliable mass production has a significant lead time.

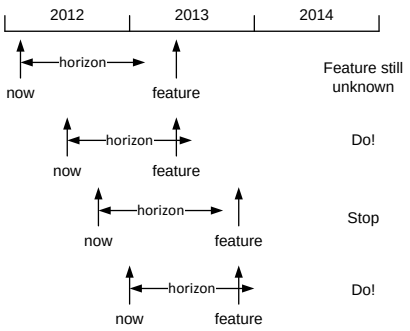


Figure 20.3: Management based on a limited horizon can result in a binary control of product policy decisions

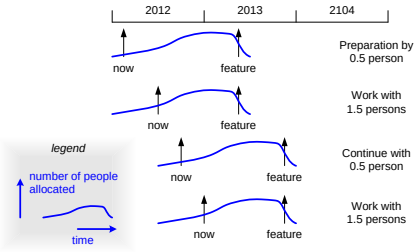


Figure 20.4: Management with a broader time and business perspective results in more moderate control: work with some more or some less people on the feature

20.4 How to create and update a roadmap

A roadmap is a joint effort of all relevant stakeholders. Typical stakeholders for roadmapping at a typical high-tech company are

business manager , overall responsible for the enterprise

marketing manager(s)

people, process, and technology manager(s) , often called line or discipline managers

operational manager(s) , e.g. program managers or project leaders

architect(s)

An efficient way to create or update a roadmap is to work in “burst-mode”: concentrate for a few days entirely on this subject. To make these days productive a good preparation is essential. Figure 20.5 shows the roadmap creation or update as three successive bursts of 2 days.

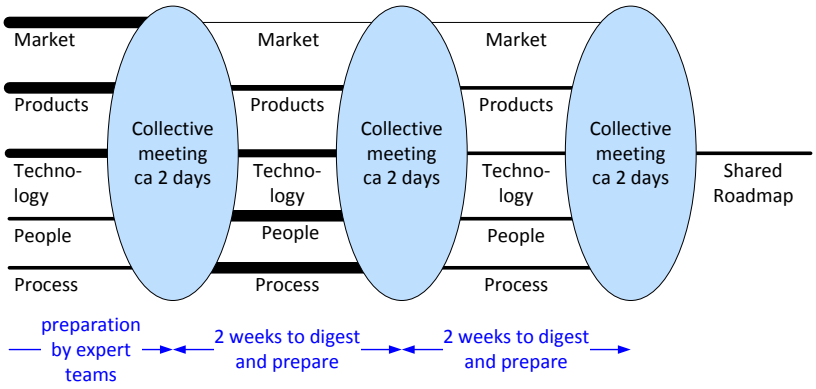


Figure 20.5: Creation or Update of a roadmap in "Burst-mode"

The input for the first days is prepared by expert teams. The expert teams focus on the *market*, the *products*, and the *technology* layers of the roadmap. The current status of *people* and *process* should be available in presentable format. The target of the first burst is:

- to get a shared vision on the market
- to make an inventory of possible products as an answer to the needs and developments in the market
- to share the technology status, trends and ongoing work, as starting point for technology roadmap
- to explore the current status of people and process and to identify main issues

Between the first and second burst and between the second and third burst some time should be available, at the one hand to digest the presented material and the discussions, at the other hand to prepare the next session. The target of the second burst is:

- to obtaining a shared vision on the desired technology roadmap
- to sharing the people and process needs for the products and technology defined in the first iteration
- to analyze a few scenarios for the layers *products*, *technologies*, *people*, and *process*

The thickness of the lines in figure 20.5 indicates the amount of preparation work for that specific part of the roadmap. It clearly shows the the shift in attention from the market side in the beginning to the people and process side later. This shift in attention corresponds with the asymmetry in figure 20.1: the market is driving the business, the people and processes are enabling the business.

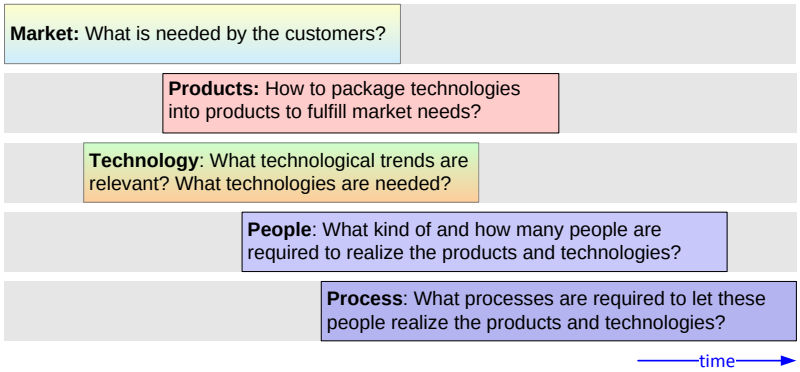


Figure 20.6: The roadmap activities visualized in time.

The function of the collective meetings is to iterate over all these aspects and to make explicit business decisions. The *products* layer of the roadmap should be consistent with the *technology*, *people* and *process* layers of the roadmap. Note that the marketing roadmap may not be fulfilled by the products roadmap, an explicit business decision can be made to leave market segments to the competition.

Figure 20.6 shows the roadmap activities in time. Vertical the same convention is used as in figure 20.1: the higher layers drive the lower layers in the roadmap. This figure immediately shows that although "products" are driving the technology, the sequence in making and updating the roadmap is different: the technological opportunities are discussed before detailing the *products* layer of the roadmap.

20.5 Roadmap deployment

The roadmap is a shared vision of the organization. This vision is implemented in smaller steps, for instance by defining outputs per program and the related resource allocations per program. In Figure 20.7 it is shown that roadmap updates are performed regularly, in this figure every year. After determining the vision a "budget" is derived that sets the charter for the programs. The budget is revised with an higher update frequency, typically every 3 months. The budget itself sets goals and constraints for the operation. The programs and projects in the operation have to realize the outputs defined in the budget. The operational activity itself uses detailed schedules as means for control. The schedules are updated more frequently than the budget update. Within the operational activity the updates are mostly event driven: changes in the market, technology or resources that render the existing plan obsolete.

From long term vision to short term realization is a 3-tier approach as shown in Figure 20.8. The roadmap provides the context for the budget, the budget defines

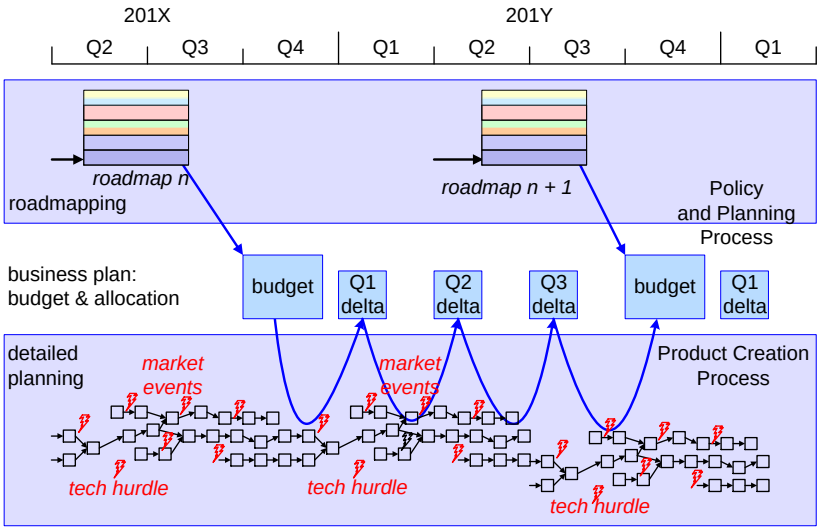


Figure 20.7: The roadmap is used to create a budget and resource allocation. The operational programs and projects use more detailed plans for control.

	horizon	*update*	*scope*	*type*
roadmap	5 years	1 year	portfolio	vision
budget	1 year	3 months	program	commitment
detailed plan	1 mnth-1yr	1 day-1 mnth	program or activity	control means

Figure 20.8: Three planning tiers and their characteristics

the context for the detailed plans. The highest tier, the roadmap, has the longest horizon, the slowest update rate, and the broadest scope. When going down in tiers, the horizon tends to decrease, the update rate increases, and the scope decreases. The roadmap provides a vision, and as such is not committal. A budget is a commitment to all involved parties. Plans are means to realize the programs and projects, and tend to adapt frequently to changed circumstances.

20.6 Roadmap Essentials

We recommend to create a roadmap that fulfills the following requirements:

- Issues are recognizable for all stakeholders.
- All items are clearly positioned in time; uncertainty can be visualized explicitly.
- The main events (enabling or constraining) must be present.

- The amount of information has to be limited to maintain the overview.

20.6.1 Selection of most important or relevant issues

The art of making a roadmap is the selection of the most relevant issues. It is quite easy to generate an extensive roadmap, visualizing all marketing and technological information. However, such superset roadmap is only the first step in making the roadmap. The superset of information will create an overload of information that inhibits the overview we strive for.

20.6.2 Key drivers as a means to structure the roadmap

In [14] key drivers are explained as an effective method to elicit and understand requirements. Key drivers can also be very helpful in the creation and update of the roadmap. At the marketing side the trend in these key drivers must be visible in the roadmap. Showing key driver trends also helps to structure the roadmap.

The supporting roadmaps can clarify how the key driver trends will be supported For instance, a technology roadmap per key driver is a very explicit way to visualize the relationship between the market in terms of key drivers, the products with the expected performance levels, and enabling technologies.

20.6.3 Nothing is certain, ambiguity is normal

A roadmap is a means to share insight and understanding in a broader time and business perspective. Both dimensions are full of uncertainties and mostly outside the control of the stakeholders. It can not be repeated often enough that a roadmap is **only** a vision (or dream?).

The only certainty about a roadmap is that reality will differ from the vision presented in the roadmap.

As a consequence the investment in making the roadmap more accurate and more complete should be limited. Nobody can predict the future, we will have to live with rather ambiguous visions and expectations of the future.

20.6.4 Use facts whenever possible

The disclaimer that *ambiguity is normal* can be used as an excuse to deliver sloppy work Unfortunately, a sloppy roadmap will backfire to the creators. It is recommended to base a roadmap on facts whenever possible Examples of sources of facts are:

- Market analysis reports (number of customers, market size, competition, trends)
- Installed base (change requests, problem reports, historical data)

- Manufacturing (statistical process control)
- Suppliers (roadmaps, historical data)
- Internal reports (technology studies, simulations)

Use of multiple data sources enable cross-verification of the sanity of assumptions For instance, predictions of the market size in units or in money should fit with the amount of potential customers and the amount of money these customers are capable (and willing) to spend.

20.6.5 Do not panic in case of impossibilities

It is quite normal that the roadmap layers appear to be totally inconsistent For instance, a frequent occurring effect is that the budget estimate in response to the market requirements is 3 times the available budget[1]. Retrospective analysis of past roadmaps shows that the realized amount of work for the given budget is often twice the estimate made for the roadmap. In other words, due to a number of effects the roadmap estimates tend to have a pessimistic bias. The overestimation can be caused by:

- Quantization effects of small activities (the amount of time is rounded to person weeks/months/years).
- Uncertainty is translated into margins at every level (module, subsystem, system).
- Counting activities twice (e.g., in technology development and in product development).
- Quantization effects of persons/roles (full time project leader, architect, product manager, et cetera per product).
- Lack of pragmatism, a more extensive technical realization than required for the market needs.
- Too many bells and whistles without business or customer value.

Initial technical proposals might be more extensive than required for market needs, as mentioned in the lack of pragmatism. Technical ambition is good during the roadmap process, as long as it does not pre-empt a healthy decisions. The roadmapping discussions should help to balance the amount of technology anticipation with needs and practical constraints.

20.7 Acknowledgements

The insight that a roadmap should cover all 5 views form market to process came to me via Hans Brouwhuis. Roadmapping as a business tool gained momentum

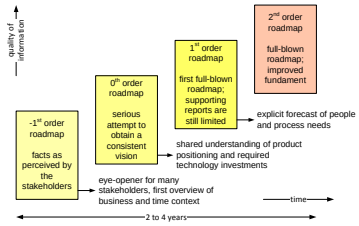
[1] This factor 3 is an empirical number which of course depends on the company and its culture

within Philips during the quality actions inspired by Jan Timmer.

The critical and constructive remarks by Jürgen Müller helped to shape this article.

Chapter 21

Change Management; Introducing Systems Architecting Aspects



21.1 Introduction

Many organizations do not have explicit roles for systems work or do not use explicit processes, methods or techniques for systems architecting. There are also many organizations that are unaware of any *systems* aspects. The introduction of any *systems* aspect in an organization is far from trivial. Introducing something new induces a negative reaction, not only for *systems* related aspects. The field of *Change Management* addresses the question how to introduce changes into an organization.

Some heuristics from *Change Management* are:

- People do not want to **be** changed. They are quite often willing to change.
- Changing the way of working or changing the culture costs many years.
- It is recommended to work at multiple tracks at the same time: amongst others managerial, operational, strategic, etc.

- Changes are better accepted when the initiators earn credit by showing usable results.

21.2 Earning Credit, Work on Urgent Issues

An effective way to introduce changes, such as new systems architecting methods, or introducing the role of a systems architect, is to earn credit by actively contributing to the organizational results. Earning credit works fastest when urgent problems are resolved. For example, systems architects typically can contribute in trouble shooting during systems integration. The systems integration phase is always hectic with lots of time and resource pressure, where mon-disciplinary engineers point to other disciplines as the source of problems. The integral overview and the systems thinking capabilities of system architects make them into ideal trouble shooters. Unfortunately, systems architects not always fancy this "foot in the mud" work.

An approach that nearly always fails is the "evangelism" approach, where systems architects try to convince the stakeholders of the value of new methods or roles by promoting the (theoretical) benefits. Most stakeholders are wary about unproven claims, especially if the messenger does not have shown any ability yet.

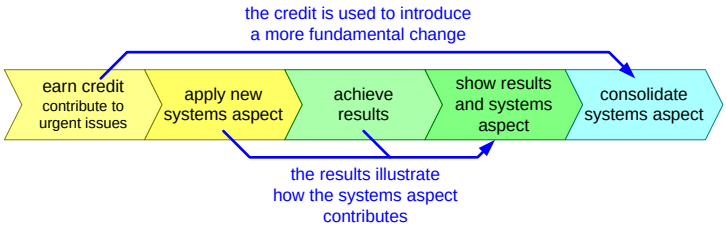


Figure 21.1: Earn Credit and Work by Example

Spending credit is going faster and easier then earning credit. We recommend to keep on earning credit, by working on actual (urgent) issues, when introducing systems aspects. Every time that some small change is introduced, architects have used some of their earned credit. Note that forcing changes costs a huge amount of credit, architects can rarely afford that.

Figure 21.1 shows how to introduce changes, earning credit, followed by creating an example, and finally by consolidating the change, using the credit earned initially. This flow shows that the introduction of the change is done by showing an example rather than preaching the change. An example is more easy understood than a theoretical explanation, while the success of the application helps to sell the idea.

21.3 Example: Bootstrapping the Roadmapping Process

Many companies and business units have no ongoing roadmapping activity or only a limited roadmapping activity, for instance a *products* roadmap only. The introduction of a roadmapping process, as described in Section 20, is a daunting task for a system architect. Roadmapping is an improvements at strategic level with mostly a long term impact. System architects need to be sufficiently known and respected in an organization to introduce roadmapping; it requires a significant amount of credit to introduce such long term improvement.

Introduction of a roadmapping process can be viewed as part of a change management process. Based on the Change Management heuristics we recommend to introduce roadmapping in a number of smaller steps. The motto here is: *Think big, act small*.

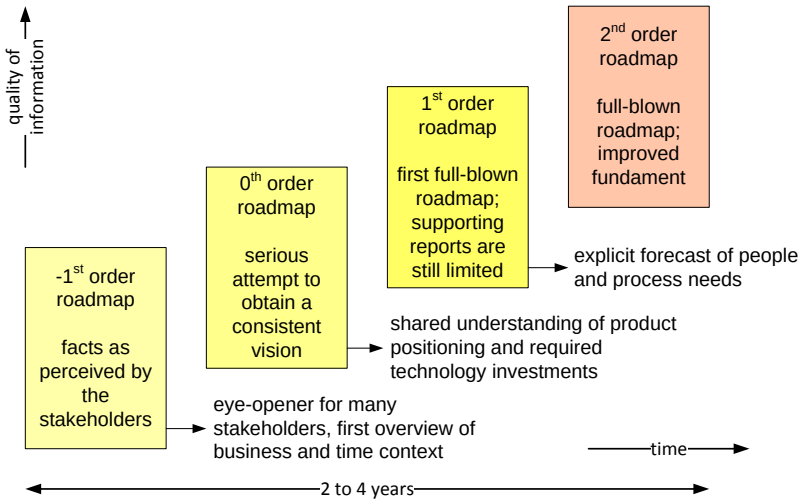


Figure 21.2: Bootstrapping the Roadmap Process

Figure 21.2 shows the bootstrapping of a roadmap process, typically taking 2 to 4 years. The benefits of starting with roadmapping become available during the first iteration. The mature roadmap, achieved in 2 to 4 years, will bring the full benefits of organizational efficacy and efficiency.

A good start is to capture the existing visions, plans, budgets, et cetera, and to integrate this information into a "minus one" order roadmap. In most cases posing such questions forces the stakeholders to reflect on the current status. In many cases the stakeholders discover that their outlook is rather unbalanced (for instance, the first half year is covered in minute detail, the latter period is fuzzy) or the outlook appears to be totally inconsistent (for instance, marketing has an entirely different expectation than development). Hopefully, the stakeholders get an overview and

gain insight in the broader context.

The result of the “minus one” order roadmap is that the architect gains credit and that the stakeholders are motivated to change somewhat. The stakeholders get ripe to make a next step, for instance to make a zeroth order roadmap.

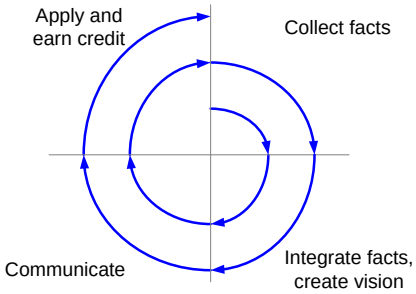


Figure 21.3: Bootstrapping the roadmap process requires a repetition of 4 steps, as visualized by this spiral

A zeroth order roadmap is the first attempt to get the *market*, the *product* and the *technology* roadmap in place. Such a partial roadmap again helps to earn credit, but it also helps to keep the stakeholders involved. Critical aspect here is the team building aspect. Roadmapping is a team activity, requiring mutual respect and trust, to enable the open and critical communication needed for the selection of the truly essential issues in the roadmap.

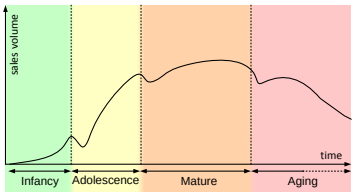
The entire roadmapping process is a repetition of the same activities, visualized in figure 21.3:

- Collect facts (e.g. market, product, technology)
- Integrate facts and create a vision, where the architect helps in the selection, the simplification, the interpretation, and the presentation.
- Communicate to a broad group of stakeholders in the organization.
- Apply the consequences for the short term and earn credit by showing a positive contribution.

Of course these four steps are not entirely sequential, they represent the main flow of the process.

Chapter 22

Market Product Life Cycle Consequences for Architecting



22.1 Introduction

A class of products serving a specific market evolves over time. This evolution is reflected in the sales volume of these products. The systems architecting approach depends where products are in this evolution.

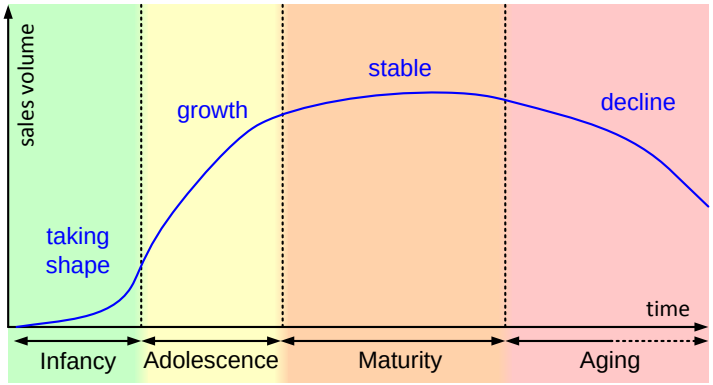


Figure 22.1: Compared with ideal bathtub curve

The life cycle of a product market combination can be visualized by showing the sales volume as a function of the time. In literature the form of the curve of the sales volume as function of the time is described as bathtub, see figure 22.1. It is customary to recognize four phases in this curve:

- The life cycle starts with very small sales in the **infancy** phase, where the product finds its shape.
- A fast increasing sales volume in the **adolescent phase**.
- A more or less stable sales volume in the **mature** phase.
- A decreasing sales volume in the **aging** phase.

The curve and its phases represent the theoretical evolution. In the next paragraphs we will discuss observations in practice and an explanation, and we will show that the class of products and the market themselves also evolve on a macro scale.

22.2 Observed Life Cycle Curve in Practice

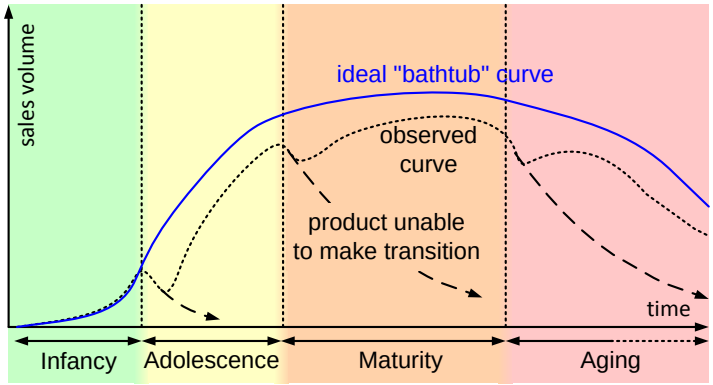


Figure 22.2: Market product life cycle phases

Henk Obbink (Philips Research) observed dips in the sales volume, as shown in figure 22.2. The transition from one phase to the next does not seem to happen smoothly. In some cases the sales drops further and the product does not make the transition at all.

The hypothesis for the dips in the curve is that characteristics of all stakeholders are different for the different life cycle phases. If the way of working of an organization is not adopted to these changes, then a mismatch with the changed circumstances results in decreasing sales. Figure 22.2 also indicates that, if no adaptation to the change takes place, that the sales might even drop to zero. Zero

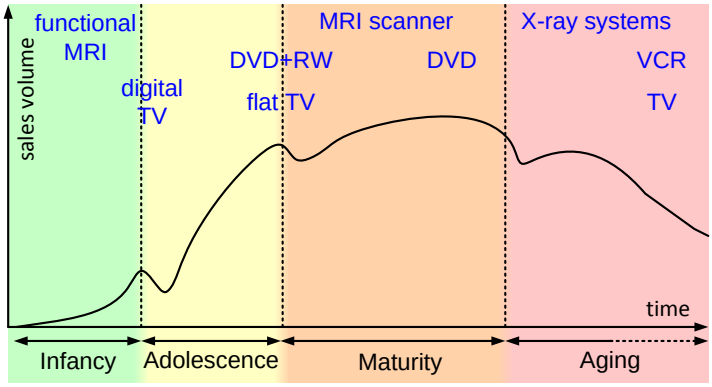


Figure 22.3: Examples of product classes on the curve

sales effectively is killing the business, while still plenty of market opportunity is present.

Figure 22.3 annotates the life cycle graph with a number of products and their positioning in the life cycle. As can be seen products can move backwards in the phases (i.e. become "younger") by the addition of innovative features. For instance MRS scanners moved backwards when *functional imaging* was added, an innovative way to visualize the activity of specific tissues. Similarly, conventional televisions rejuvenated multiple times by adding digital processing, flat screens, and digital interfaces.

22.3 Life Cycle Model

Figure 22.4 shows typical attributes of the life cycle phases.

The *infancy* phase is characterized by uncertainty about the customer needs, and therefore the product requirements. Essential is that the creator/producer is responsive to the customer needs, which will provide insight in needs and requirements. The way of working in this phase reflects the inherent uncertainty, the chaotic development, and the innovative and pioneering mind set. Product cost is still less of an issue, the risk related to the uncertainty is the dominant concern. The design copes with the uncertainty by over-dimensioning those aspects which are perceived to be the most uncertain.

The *adolescent* phase is characterized by strong (exponential) growth of the sales volume, concurrent with an increase in performance, features and product variants. The challenge is to cope with this strong growth in many dimensions. With respect to the requirements a strategic selection is needed, to serve the growing customer base, without drowning in an exploding complexity. The technical and process challenge is to scale up in all dimensions at the same time. Up-scaling

	Infancy	Adolescence	Mature	Ageing
Driving factor	Business vision		Stable business model	Harvesting of assets
Value from	Responsiveness	Features	Refinements / service	Refining existing assets
Requirements	Discovery	Select strategic	Prioritize	Low effort high value only
Dominant technical concerns	Feasibility	Scaling	Legacy Obsolescence	Lack of product knowledge Low effort for obsolete technologies
Type of people	Inventors & pioneers	Few inventors & pioneers "designers"	"Engineers"	"Maintainers"
Process	Chaotic		Bureaucratic	Budget driven
Dominant pattern	Overdimensioning	Conservative expansion	Midlife refactoring	UI gadgets

Figure 22.4: Attributes per phase

the Customer Oriented Processes and the Product Creation Process requires more shared structure between the participants. This involves a mind set change: less inventors, more designers. The design pattern used frequently in this phase is conservative extension of a base design.

The *mature* phase is characterized by more stability of the business model and the market, while the market has become much more cost sensitive. Instead of running along in the feature race more attention is required to optimize the specification and development choices. The value can be shifting from the core product itself to services and complements of the product, while the features of the product are mostly refined. The age of the product starts to interfere with the business, obsolescence problems occur, as well as legacy problems. Innovative contributions become counterproductive, more rigid engineers are preferred above creative designers. The cost optimization is obtained by process optimization, where the processes also become much more rigid, but also more predictable, controllable and executable by a large community of less educated engineers. The design copes with the aging technology by performing limited refactoring activities in areas where return on investment is still likely.

The *aging* phase is often the phase where the product is entirely seen as cash cow, maximize the return on (low) investments. This is done by searching all the low effort high value requirements, resulting mostly in small refinements to the existing product. Often the integral product know how and even specialist know how has been lost. Only very important obsolescence problems are tackled. Again the mind set of the people working on the product is changing to become more maintenance oriented. Cost is a very dominating concern, budgets are used to

control the cost. Many changes are cosmetic or superficial, taking place in the most visible parts of the product: the user interface and the outer packaging.

22.4 Acknowledgements

Henk Obbink observed the discontinuity of market success at the phase transitions. The analysis of this phenomenon was carried out by Jürgen Müller, Henk Obbink and Gerrit Muller.

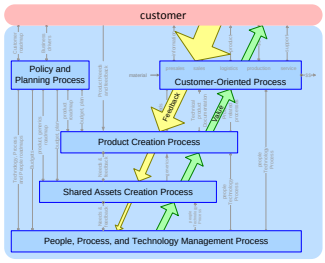
Pierre America improved the layout of the diagrams.

Part IV

Product Families, Generics and Software

Chapter 23

Product Families and Generic Aspects



23.1 Introduction

Platform
Common components
Standard design
Framework
Family architecture
Generic aspects, functions, or features
Reuse
Products (in project environment)

Figure 23.1: Different names for development strategies that strive to harvest synergy

Harvesting synergy between products or projects is being done under many different names, such as shown in Figure 23.1. We use *generic developments* or

harvesting synergy as label for this phenomena. The reader may substitute the name that is used in their organization.

Many trends (increased variability, increased number of features, increased interoperability and connectivity, decreased time to market, globalization of development, globalization of markets) in the world force organizations into these strategies where synergy is harvested. Harvesting synergy is, however, also a complicating factor both organizational and technical. We strive to give insight in both needs and complications of harvesting synergy, in the hope that awareness of the complications will help to establish an effective synergy harvesting strategy.

23.2 Why generic developments?

Many people advocate generic developments, claiming a wide range of advantages, such as listed in Figure 23.2.

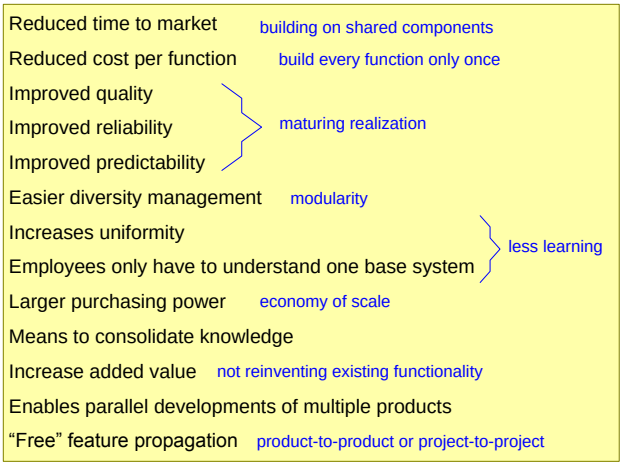


Figure 23.2: Advantages which are often claimed for generic developments

Effective implementation of generic development has proven to be quite difficult. Many attempts to achieve these claims by generic developments have resulted in the opposite of these claims and goals, such as increased time to market, quality and reliability problems et cetera. We need a better rationale to do generic developments, in order to design an effective Shared Assets Creation Process.

Figure 23.3 shows drivers for Generic Developments and the derived requirements for the Shared Assets Creation Process. The first driver (*Customer value* is extrovert: does the product have value for the customer and is the customer willing to buy the product? The second driver *Internal Benefits* is introvert, it is the normal economic constraint for a company.

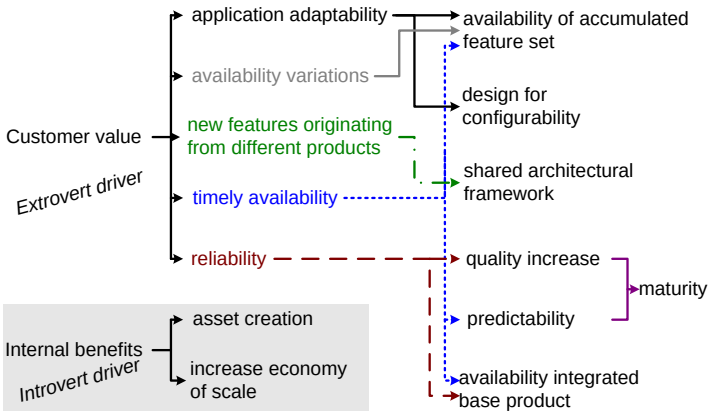


Figure 23.3: Drivers of Generic Developments

Today high tech companies are know how and skill constrained, in a market that is extremely fast changing and rather turbulent. Cost considerations are an economic constraint that has to be balanced with the capability to create valuable and sellable products.

The derivation of the requirements for the product development shows that these requirements are not a goal in itself, but are a means to facilitate an higher level goal. For instance, a shared architecture framework is required to enable features developed for one product to be used in other products too. This propagation of features makes sense if it creates value for a customer. So the verification of the shared architecture framework requirement has to involve the propagation a new feature from one product to the next, using limited effort and lead time.

We emphasize the derivation from drivers to requirements because many generic developments fulfil the requirements, such as *availability accumulated feature set*. *designed for configurability*, *shared architectural framework*, and *maturity or implementation*, without bringing the assumed customer or sales value. For example, many generic developments result in large monolithic solutions, without flexibility and long development times. Developers of such framework have been providing replies as: ”You can not have this easy shortcut, because our architectural framework does not support it, changing the framework will cost us 100 man-years in 3 years elapsed time”.

23.3 Granularity Of Generic Developments

Granularity is one of the key design choices for systems architects: what is an appropriate decomposition level for modularity? Granularity decisions have to be made at all levels for different purposes. For example, in the application granularity

of functions and roles, at specification level granularity of options and features, in conceptual design granularity of functions and concepts, and in implementation granularity of many operations.

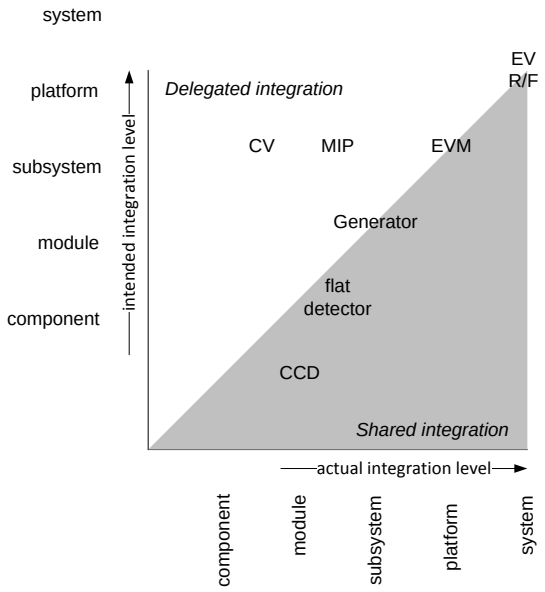


Figure 23.4: Granularity of generic developments shown in 2 dimensions.

Figure 23.4 shows the granularity of generic developments in 2 dimensions. The vertical dimension is the preparation level: What is the intended scope of the generic developments, how far is the deployment prepared? The horizontal dimension is the integration level: How far are the generic developments integrated when the *product developers* deploy the generic development?

Both axis range from (atomic) component until (configurable) system. Developments on the diagonal axis, which have a scope where the preparation level is equal to the integration level, are straightforward developments in which the integration takes place as far as autonomously possible. Some generic developments concentrate on the generation of building blocks, leaving ("delegating") the integration to the product developer. For rather critical generic developments the the integration of the shared asset goes beyond its own deliverable to ensure the correct performance of the asset in its future context(s).

In these figures a number of medical generic developments are shown, as an example for the categorization.

An extreme example of "delegated" integration is Common Viewing (CV). The organization made an attempt to harvest synergy at the end of the eighties. The vision was to create a large "toolbox" with building blocks that could be used in

a wide variety of medical products ranging from Magnetic Resonance Imaging (MRI) scanners to X-ray systems. A powerful set of (mostly SW) components was created, using Object Oriented technology and supporting a high degree of configurability

The CV toolbox proved difficult to sell to product developers, amongst others due to the low integration level. The perception of the product developers was that they still had to do the majority of difficult work: the integration. The vision of a marketing manager changed the direction of CV into creating a completely integrated product: EasyVision Radiography Fluoroscopy (EV RF). This medical workstation for the URF (Universal Radiography Fluoroscopy) market was highly successful, serving as an intelligent print server. The communication and print function were highly configurable to make the product adaptable to its environment.

The EasyVision RF was used as a basis for a whole series of medical workstations and servers. The shared functionality is developed as generic development at platform level. This platform is nowadays called EasyVision Modules (EVM). Despite its name it has still a significant integration level, with its upside (product developers are not bothered with the lower level integration) and its downside (predefined functionality and behavior).

The old CV vision is revived and a second generation of EVM is being created, covering the EVM platform functionality with finer granularity: a module level of integration. The whole evolution as described here from CV as toolbox to more fine grained EVM modules took about 15 years. During all these years the balance between genericity (degree of sharing) and customer value has been changing without ever achieving the combination of a high degree of sharing and a high customer value at the same time.

23.4 Modified Process Decomposition

In 3 we discussed a simplified process description of companies. This decomposition assumes that product creation processes for multiple products are more or less independent. When generic developments are factored out for strategic reasons then an additional process is added: the Shared Assets Creation Process. Figure 23.5 shows the (still simplified) modified process decomposition

Figure 23.6 shows these processes from the financial point of view. From financial point of view the purpose of this additional process is the generation of strategic assets. These assets are used by the Product Creation Process to ensure the cash flow for the near future by staying competitive.

The consequence of this additional process is an lengthening of the value chain and consequently a longer feedback chain as well. This is shown in figure 23.7. The increased length of the feedback chain is a significant threat for generic developments. The distance between designers and developers of shared assets and

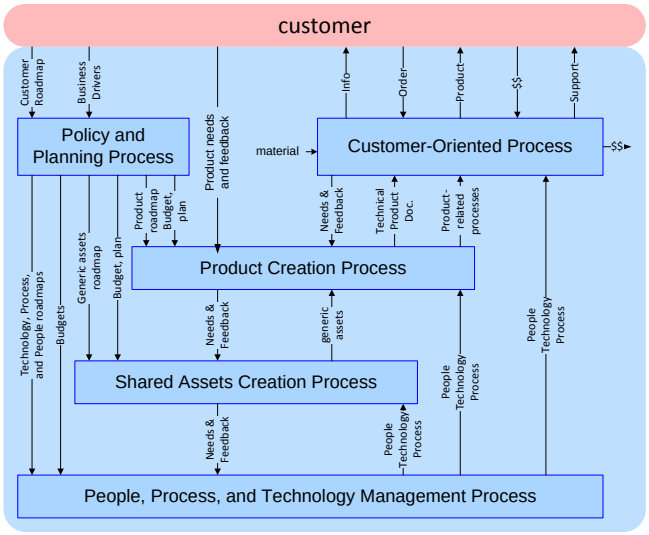


Figure 23.5: Modified process decomposition

the stakeholders in the outside world is large. These developers easily lose focus on customer value and may focus on the technology instead. Successful sharing requires a strong relation between customer value and technology.

23.5 Modified Operational Organization of Product Creation

The operational organization of the Product Creation Process is described in 3. This organization is a straightforward hierarchy, where the limited amount of relations (conflicts) between products or subsystems are managed at the closest hierarchical management level.

Introduction of generic developments complicates the operational structure significantly[1]. Figure 23.8 shows the operational organization of the Product Creation Process, with the necessary additions to support generic developments.

The conventional Product Creation Process is based on a relative straightforward hierarchy, where the control flow and delivery flow are opposite, where both flows follow the hierarchy. The introduction of generic developments breaks this simple structure: a generic development team delivers to multiple product developments, where the control is taking place from an encompassing operational level, to enable operational balancing of products and generic developments. In other words the principal of the project leader is not the customer anymore, but an

[1]The complication can be avoided by working sequentially. However in today's dynamic market sequential work results in unacceptable lead times. Concurrent engineering is a fact of life. Organizations are looking for opportunities to reduce the lead time more.

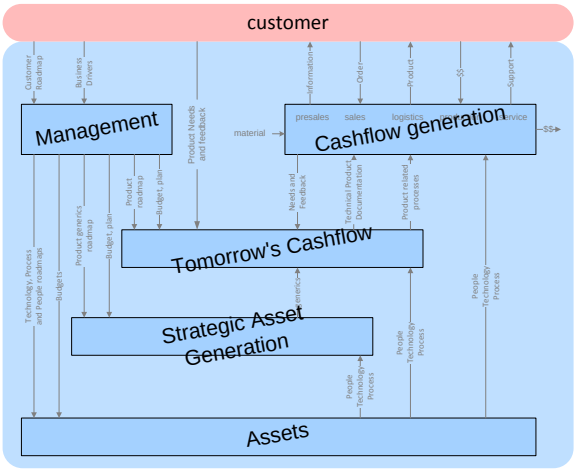


Figure 23.6: Financial viewpoint of processes

intermediate manager.

Every operational entity needs the 3 complementing processes in the product creation process: operational management, design control and commercial. For each of these processes a role is required of someone responsible for that process: the operational manager, the architect and the commercial manager. Together these 3 people form the core team of the operation. Introduction of generic developments also requires the introduction of these roles for the shared assets, such as platform or components.

For the architect role this means that a platform architect is needed, who is closely working together with the platform project leader and the platform manager. At the other hand the platform architect needs many architectural contacts with the product family architect, acting as the architectural principal, with the product architect, acting as customers, and with the component architects, acting as suppliers.

The separation of the roles of the platform architect and the product family architect is not obvious. For example in [9] 3 operational entities with related processes and roles are identified. Application Family Engineering (AFE), Component System Engineering (CSE), and Application System Engineering (ASE) map respectively on Product Family, Component, and Product as shown in Figure 23.8. We will either have a gap or a double role, when mapping 4 operational entities on 3 processes. In practice the result is that one of the roles is missing, or played implicit. For instance quite often the application family engineer starts to play platform architect, forgetting the original task *application family* engineering. We have observed that architects either tend to play the platform architect role or the product family role. Architects combining both roles naturally are scarce.

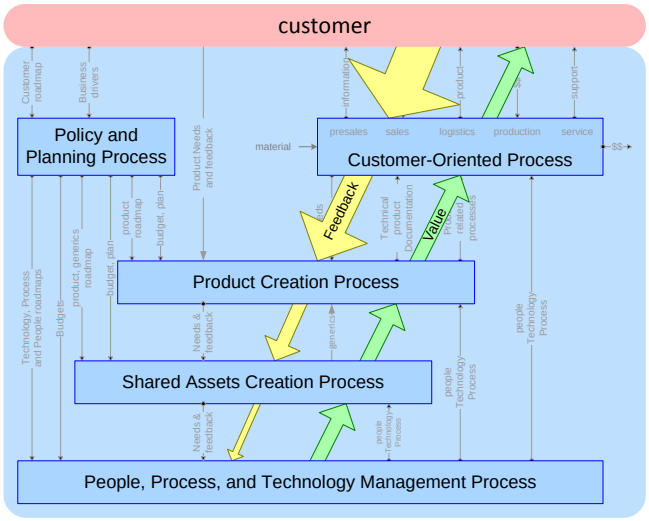


Figure 23.7: Feedback and Value flow

23.6 Models for Generic Developments

Many different models for the development of shared assets are in use. An important differentiating characteristic is the driving force, often directly related to the de facto organization structure. The main flavors of driving forces are shown in figure 23.9.

23.6.1 Lead Customer

The lead customer as driving force guarantees a direct feedback path from an actual customer. Due to the importance of feedback this is a very significant advantage. The main disadvantages of this approach are that the outcome of such a development often needs a lot of work to make it reusable as a generic product. The focus is on the key functions and performance parameters of the lead customer, while all other functions and performance parameters are secondary in the beginning. Also the requirements of this lead customer can be rather customer specific, with a low value for other customers.

23.6.2 Carrier Product

The combination of a generic development with one of the product developments also shortens the feedback cycle, although the feedback is not as direct as with the lead customer. Combination with a normal product development will result in a better coverage of performance parameters and functionality. Disadvantage can

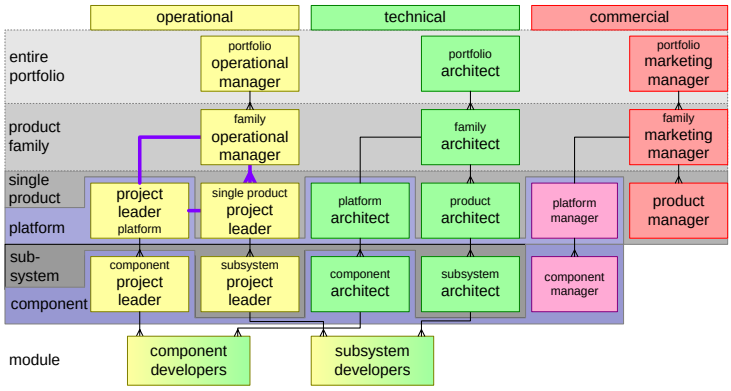


Figure 23.8: Operational Organization of the Product Creation Process, modified to enable generic developments

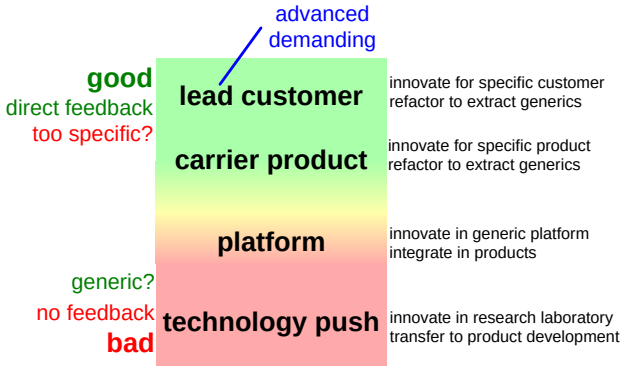


Figure 23.9: Models for SW reuse

be that the operational team takes full ownership for the product (which is good!), while giving the generic development second priority, which from family point of view is unwanted.

In larger product families the different charters of the product teams create a political tension. Especially in immature or power oriented cultures this can lead to horrible counterproductive political games.

Lead customer driven product development, where the product is at the same time the carrier for the platform combines the benefits of the lead customer and the carrier product approach. In our experience this is the most effective approach of generic developments. A prerequisite for success is an open and result driven culture to preempt any political games.

23.6.3 Platform

Generic developments are often decoupled from the product developments in maturing product families, by creating an autonomous Shared Asset Creation Process. In products where integration plays a major role (nearly all products) the shared assets are pre-integrated into a platform or base product. Such platform or base product follows its own release process before it can be used by product developments.

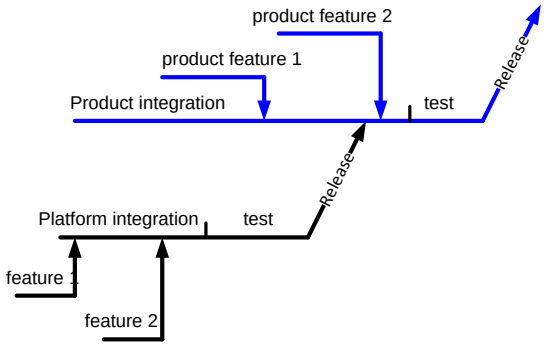


Figure 23.10: The introduction of a new feature as part of a platform causes an additional latency in the introduction to the market.

The benefit of this approach is separation of concerns and decoupling of products and platforms in smaller manageable units. These benefits are also the main weakness of such a model: as a consequence the feedback loop is stretched to a dangerous duration. At the same time the duration from feature/technology to market increases, see figure 23.10.

23.6.4 Alternative Generic Development Scenarios

A number alternative re-use strategies have been applied with more or less success:

Spin-out as an independent company is especially tried for key and base technologies. However, many spin-out companies have been re-absorbed by their parent companies. Examples are multimedia processors from TriMedia (parent Philips Semiconductors, later NXP) and cell phone operating system Symbian (parent Nokia)

Reuse after use works quite good in practice, especially for good clean designs.

Opportunistic copy where implementations are taken that are available. The results are quite mixed. Short term benefits are quick results and hence short feedback cycles. Longer term a problem can be that an architectural mess has been growing that turns into a legacy.

Open source where key and base technologies are shared and developed much more publicly.

Inner-source , where a company stimulates sharing takes place within a company modeled after an open source approach.

volutionary refactoring where the architecture and its components are actively re-factored to keep them fit for the future and for potentially increased scope of application

23.7 Common Pitfalls

We learn from out mistakes. Unfortunately, many mistakes have been made in the area of generic developments. We compiled the list of pitfalls shown in Figure 23.11 from mistakes in generic developments in the past. Some of the attempts to harvest synergy were partially successful, but issues from the list limited the degree of success.

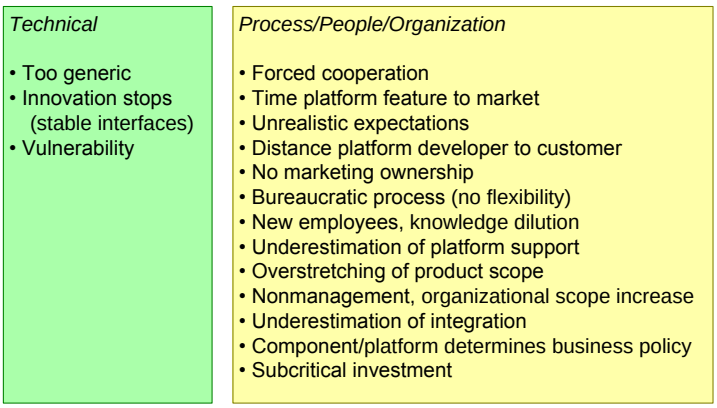


Figure 23.11: Sources of failure in generic developments

Most of the problems have a root cause in people, process, or organizational issues. The list with technical problems is relatively small:

Too generic platform or components that can do everything, but nothing really good: “the Swiss army knife”

Innovation stops , because existing interfaces are declared to be stable. Existing structure and interfaces can block innovation.

Vulnerability , because all products use one and the same core. If the shared core has a problem anywhere then all products are hit simultaneously. Diversity

is a characteristic that enhances resilience. In nature, species often survive disasters, such as diseases, due to the diversity in the population.

Forced cooperation by upper management, de-motivating employees, and creating social and political tensions in the organization.

Time platform feature to market because of stacked release procedures.

Unrealistic expectations by upper management, often as a consequence of the claims from architects and engineers o the benefits of harvesting synergy. When less is delivered than promised, then a negative spiral sets in of cost reduction and hence even more decreasing outcome.

Distance platform developer to customer , see Figure 23.7.

No marketing ownership , but engineering push only. Marketing support is crucial, since marketing is one of the key players when making decisions about investments. Lack of marketing ownership results in a continuous fight for funding, with starvation in the end.

Bureaucratic process , and loss of flexibility. The increased scope of the operation (common components or platform plus derived products) often requires a more formal organization than the individual products used to have. The formalization easily turns into bureaucratism, slowing down the entire organization.

Knowledge dilution caused by the hiring of new employees. Often an increase in resources is needed early during the creation of shared assets. If these new resources are inexperienced, then the knowledge is diluted, resulting in less quality of the created assets.

Underestimation of shared asset support required when the shared assets are used by products. Product designers need support when specifying and designing new products based on these assets, and they need support for trouble shooting during integration and introduction in the field. When components are used in new circumstances (e.g. new products), then always unexpected problems pop-up.

Overstretching of product scope beyond the natural level of synergy. Harvesting synergy is a balancing act, between maximum value creation for specific customers and minimizing diversity in the realization. When the minimization of diversity dominates over value creation, then customers are not served well, resulting in a loss of business. Organizations easily lose their customer focus, when creating a synergy drive.

Non-management of organizational scope increase that is inherent when multiple products share assets. The scope increase requires organization, process and staffing adaptations.

Underestimation of integration of shared assets in other products. Systems integration is often ill understood and hence underestimated, see **??**. When existing products have to migrate to the use of shared assets, then this requires that these products adapt their architecture too.

Component/platform determines business policy which is effectively an inversion of the need driven approach. This inversion relates to the distance between shared asset development and customers. What happens is that what *can* be done dominates over what *needs* to be done. The shared asset developers get de facto power, since all products depend on their delivery.

Subcritical investment , caused by a cost reduction focus. Shared asset development primarily should bring market and customer value, while keeping the cost limited by harvesting synergy. As soon as cost reduction dominates over value creation, then all products and shared assets can get too little investment, causing delays and quality problems.

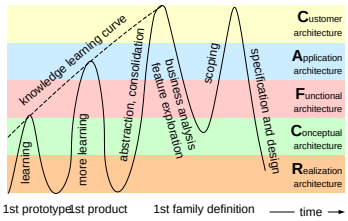
23.8 Acknowledgments

During the first CTT course system architecture, from november 22 until november 26 1999, a lively discussion about generic developments took place, which created a lot of input for this article. I am grateful to the following people, who attended this course: Dieter Hammer, Wil Hoogenstraaten, Juergen Mueller, Hans Gieles, Huib Eggenhuisen, Maurice Penners, Pierre America, Peter Jaspers, Joost Versteijlen, Peter Beelen, Jarl Blijd, Marcel Dijkema, Werner Roelandt, Paul Janson, Ashish Parasrampuria, Mahesh Bandakka, Jodie Ledeboer

I thank Pierre America for working on consistency in spelling and the use of capitols. Ad van den Langenberg pointed out a number of spelling errors.

Chapter 24

Product Familiy Business Analysis And Definition



24.1 Introduction

The creation and evolution of a product family is based on a business analysis. Such a business analysis is used for the definition of the family: which products are member of the family, what distribution of features, which performance range.

Several methods can be used to make the step from business analysis to product family definition, see for instance Figure 24.1.

- Roadmapping
- Reference Architecture
- Requirements Capturing
- Feature Space Exploration
- Value Engineering
- Scope Determination

Figure 24.1: Methods for Family Analysis

24.2 Roadmapping

About once per year it is recommended to work for a number of weeks on roadmaps. These roadmaps serve as a shared vision of the next 5 years, see [15]. Roadmapping is done at the level of a product portfolio or product family. The value of roadmapping is that it brings understanding over 5 views: market, product, technology, process and people. This understanding has the time perspective as the main dimension.

The roadmaps provide a time and product portfolio context for the definition of a product family. A number of the activities in roadmapping and product family definition are quite similar; both require an market analysis, a good understanding of commercial opportunities and insight in the technology.

Roadmapping is focused on insight, understanding and shared vision, without any commitment. The definition of a product family results in a more specific detailed output, which is at least partially committal. In other words. Roadmapping is transforming a strategy into tactics, while Product Family definition transforms the tactics into operational activities.

24.3 Reference Architecture

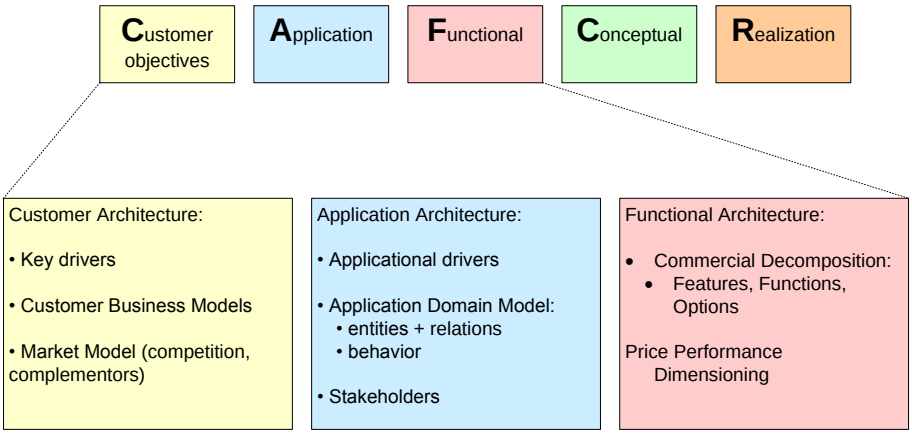


Figure 24.2: Product Family Reference Architecture, zooming in on the views determined by the business analysis and family definition process

A reference architecture covers 5 viewpoints on a product family, see figure 24.2. The business, application and functional architectures are the main subjects of interest during the business analysis and family definition process.

24.3.1 Business Architecture

The business architecture models the world of the customer. Again a number of complementary views are required.

The key drivers of the customer are identified, see [14]. A limited, but specific set of key drivers is a powerful guide in the entire creation process.

The business model of the customer is determined, see typical questions addressed by a business model in Figure 24.3

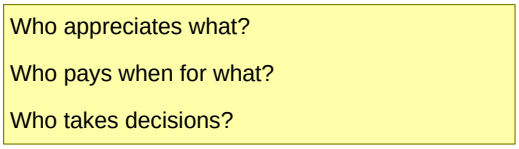


Figure 24.3: Questions addresses in the business model

The business of the customer is served by many different suppliers. Some of these suppliers are competing with your own business, while others are complementary. This information is compiled into a market model.

Example Set top boxes are supplied to different kinds of customers, varying from consumers to content providers. In case of the content providers different business models are practiced, ranging from pay-per-view to entirely paid by the advertisers.

The set top box is only a small part of the value chain. Many complementers are active in this entire chain, which starts at the content generation and ends at the television screen of the consumer. Philips is quite active in all complementing products at the consumer side, such as television and video storage, while it is active in parts of the value chain proceeding the set top box.

The competition exists from comparable set top box manufacturers, but also new devices such as game computers (Playstation 2) enter this market.

24.3.2 Application Architecture

The application architecture models the way the user works or enjoys your products in a broader context.

The key drivers of the business architecture are transformed into application drivers, which describe what the user needs to fulfill the key drivers of the business. These application drivers provide insight. The direct relation with the key drivers and the functional requirements provide traceability and a means to focus the requirement process.

Application domain models support the other processes by providing a shared reference. A model describing the entities and their relations ”sets the stage”; it

defines the relevant entities such as persons, tools, deliverables, consumables et cetera. A dynamic view on the application is given in the behavior model. Both models at this level should focus on the main issues, detailed definitions endanger the overview and understanding.

Figure 24.2 explicitly mentions stakeholders as part of the application architecture. Of course stakeholders will show up as an entity. The (human) stakeholders play such an dominant role in the application that it is useful to make a separate overview of the stakeholders and their roles.

Example: An X-ray diagnostic system requires predefined diagnostic procedures to be used easily. These procedures are based on rather specific domain knowledge, such as demographic data, pathology and anatomical data. The essentials of the way of working should be described in the application drivers.

The application model would describe all relevant entities, such as patient, patient table, monitors, UI devices, tube, detector, ECG monitor, film, examination room, technician, nurse, patient et cetera including their relationship.

Note that understanding is the aim of this exercise, not completeness. Those entities and relations should be shown which are relevant for the shared (by commercial and technical people) understanding of the application.

The behavior model would describe the dynamics of the application. It could for instance describe the patient flow, and the information flow.

The application stakeholder view focuses on the human players, which are in this case: referring physician, receptionist, radiologist, patient, technician, nurse, technical support staff of the hospital, et cetera.

24.3.3 Functional Architecture

The functional architecture is the commercial view on the system, describing the commercial flexibility of the products. The functional architecture is the basis of the sales catalog.

The commercial decomposition defines in terms of functions, features and options the capabilities of the products and their structure from commercial point of view. The product manager decides which items to package in sellable products.

The price performance ranges are also defined in the functional architecture.

24.4 ”YoYo-View” over time

To define a product family technical, business and application know how are a prerequisite. Figure 24.4 shows that this know how is often the result of previous experience with single products [1]. The curve indicates the architectural focus as

[1] recruitment of experienced people is also an effective way to obtain the know how. In fact the same learning curve is followed, but external.

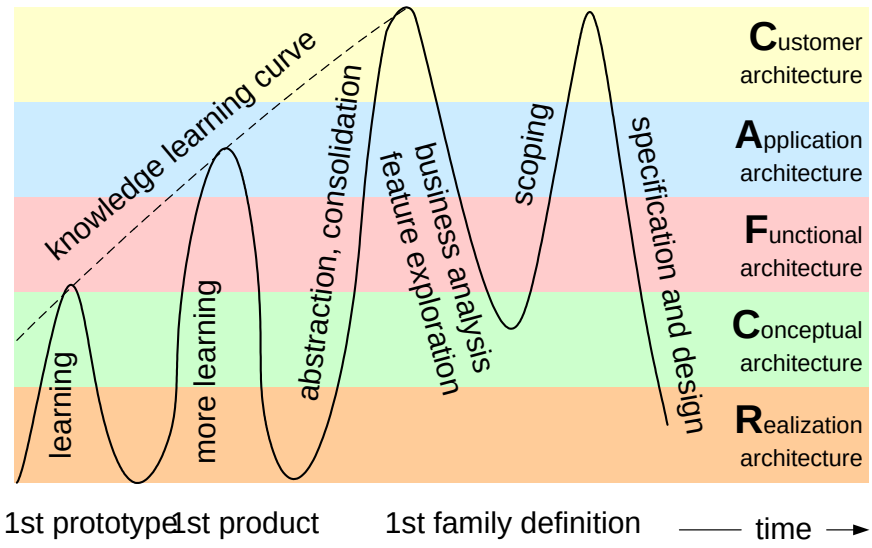


Figure 24.4: The analysis and definition of a family requires a number of iterations over the views in the reference architecture

a function of time. This focus is iterating over the CAFCR views. The diagram simplifies this learning curve to a single prototype and product, in reality more generations are required for the build up of the know how.

When enough knowhow is present in the group of people, this know how is made explicit in the form of a reference architecture. The ”problem” is now analyzed by making a business analysis, feature space exploration and valuation of the features. As shown in figure 24.4 this activity ranges over the business, application and functional architectures.

The next step is to go back to a more fundamental question: What is an effective *scope* for the product family? A broad scope is attractive for customers when they benefit from a rich offering of product members and options. From business perspective a broad scope is desirable to increase economy of scale effects and to harvest synergy by sharing development efforts. However, at the same time a broad scope increases dependencies between markets and products, increases organizational complexity, and increases internal communication and process needs. Overstretching the product scope has been a major cause of product family failures.

Once the organization has sufficient know and the scope is clear, then the actual product family specification and design can take place in a more or less top down fashion[2].

[2]Although iterative (evolutionary, agile) development approaches are highly recommended.

24.5 Relation with the Technical Architecture

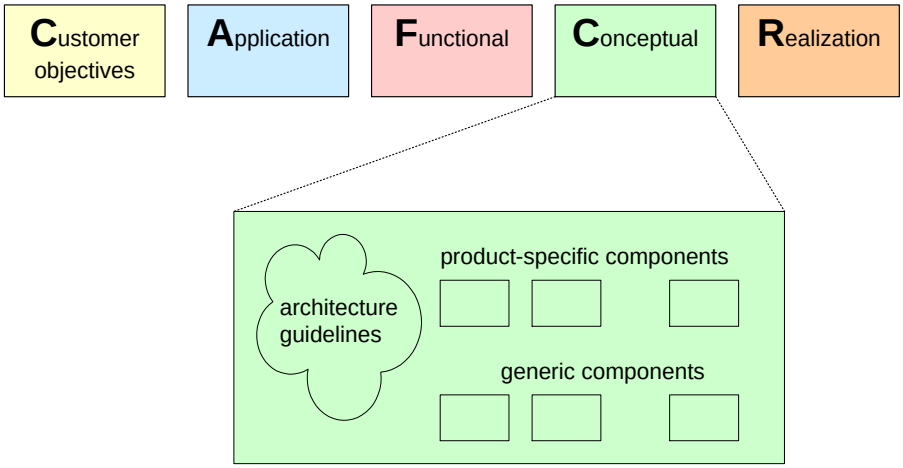


Figure 24.5: Technical Architecture for a Product Family

The family definition will have to iterate with the technical and implementation architecture. Figure 24.5 shows an example of the contents of a technical architecture in case of a Component Based Product Family.

Rather fundamental decisions which have to be taken for the technical architecture is where to address the requirements, in:

- Product Specific Components,
- Generic Components, or in
- Architecture Guidelines.

Ideally the technical structure closely resembles the functional structure, by a natural mapping of functions and features on components.

24.6 Requirements Capturing

Collection and analysis of requirements is indispensable. Many methods exists to do this. In [14] the requirements capturing is described for products. However the methods described in this article are also applicable for Product Families.

Product Family Definition requires special attention for commonality and variation analysis and for product positioning. In section 24.7 some more detailed method is described to address these issues.

Also special attention should be paid to the life cycle requirements, these requirements often originate at internal stakeholders, such as sales, manufacturing,

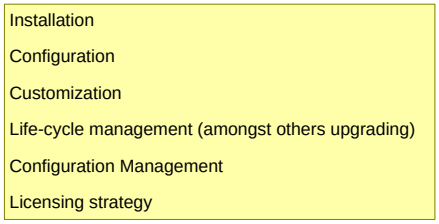


Figure 24.6: Subjects requiring special attention for Product Families

service et cetera. Figure 24.6 shows a list of subjects which require special attention in case of product families.

24.7 Feature Space Exploration and Value Engineering

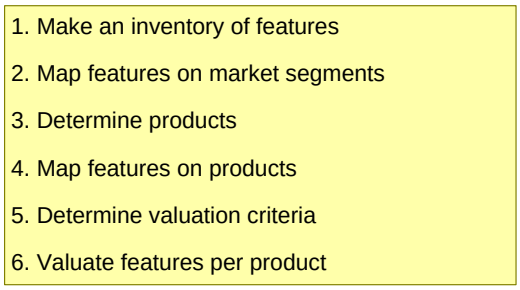


Figure 24.7: From Feature Exploration to Valuation per Product

Analysis of commonality and variation of features over products helps to define the product family in first instance and to make a family design in second instance. This analysis starts with an exploration of the feature space, and results in a valued set of features per product. Figure 24.7 shows which steps are taken in this process. See also [14] which describes how to obtain requirements.

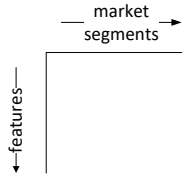


Figure 24.8: Market Feature Map

The features can be mapped on market segments, resulting in a matrix, see figure 24.8. The feature axis can be ordered, for instance by following the key driver, application driver derivation.

Again iteration is the magic word. Iterate a few times from Market segment to Features and vice versa. If key drivers are used as structure for the feature axis, then these key drivers should be included in the iteration. Market segments can have different key drivers!

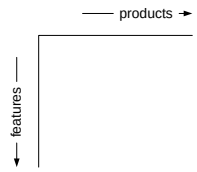


Figure 24.9: Product Feature Map

The Market segmentation can be transformed in products, once sufficient insight is obtained in the market segments and the features involved. This results in a Product Feature Map, see figure 24.9.

- Value for the customer
- (dis)satisfaction level for the customer
- Selling value (How much is the customer willing to pay?)
- Level of differentiation w.r.t. the competition
- Impact on the market share
- Impact on the profit margin

Use relative scale, e.g. 1..5 1=low value, 5 -high value
Ask several knowledgeable people to score
Discussion provides insight (don't fall in spreadsheet trap)

Figure 24.10: Example of Valuation Criteria

Valuation criteria are needed to determine the value of features. Figure 24.10 shows an example of Valuation Criteria.

Figure 24.11 shows the result of the entire process. Here all the features have been valuated, the corresponding values are substituted in the matrix.

This matrix is the starting point for the selection process, see section 24.4, which finally has to answer:
Which Feature will be realized When for Which product?

A much more elaborated method for feature space exploration, valuation and scoping can be found in [4].

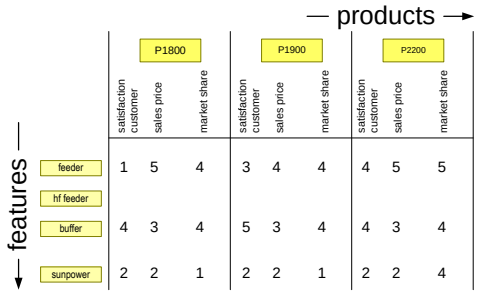


Figure 24.11: Product Feature Map with substituted Numbers

24.8 Scope Determination

A fundamental question in Product Family approach is the scope of the family

Which part of the Market do we want to serve?

A clear shared answer on this question is the key to an efficient continuation of the Family Creation Process. Some more nuance can be added to the question by including the time dimension (*When?*).

Note that figure 24.4 also simplifies the scoping to a single iteration. In reality some iteration with the technical and implementation architecture takes place.

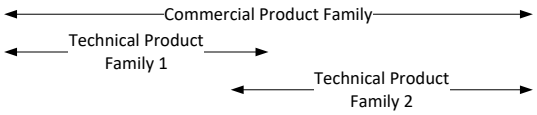


Figure 24.12: Commercial and Technical Viewpoint on Product Families

The scope determination is primarily a commercial scoping. Later in the process, as part of the Family Design, also technical scope determination is needed. Figure 24.12 shows that a commercial Product Family can be realized by two technical product Families.

Example High end products (”Up-market Televisions”) will emphasize a richness of features, irrespective of for instance memory and processor constraints, while the mid range products (”Mainstream Televisions”) have a severe cost constraint, which translates in memory and processor constraints. From commercial point of view it should appear as one continuous family. From technical point of view the requirements could be conflicting too much, while two technical families with a different optimization focus, match perfectly with the commercial requirements.

24.9 Acknowledgements

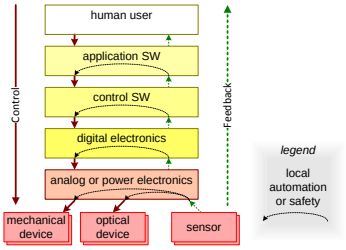
Frank van der Linden wrote a position paper on this subject to trigger the discussion for the ”Family Engineering Handbook”. After Frank left Philips Research to join Philips Medical Systems I inherited the job to write this section of the handbook. I thank Frank for writing the original position paper which served as a starting point of this article.

Discussions with Jürgen Müller helped to sharpen the contents of this article. Discussions in the composable architecture meeting, attended by Pierre America, William van der Sterren, Jan Gerben Wijnstra and Jürgen Müller helped to make the article more complete and consistent. Ad van den Langenberg pointed out a number of spelling errors.

James Sirota pointed out that the explanation of the Yoyo-figure was very limited. He also suggested to add a warning about opportunistic re-use.

Chapter 25

The Role of Software in Systems



25.1 Introduction

The relation between the software and system disciplines is difficult in many organizations. The poor relation between the disciplines results in gaps in the design and later in quality problems in the final systems. As a consequence software is in many organizations perceived as a problem and a bottleneck in product creation.

Part of the explanation is traditionally physical disciplines, e.g. mechanical, optical, or electrical engineering, dominated system design. Historically the engineers from these physical disciplines were confronted most with the application domain. These engineers have evolved into domain engineers.

In the modern world software has a significant impact on many system qualities, as we will show in this chapter. More and more customer value depends on software. Unfortunately, many software engineers have not yet build up sufficient knowledge of the physical aspects of their systems or of the application domain. At the same time the engineers from the physical disciplines, who dominate the system design, do not yet understand the jargon and the concepts from the "virtual" disciplines (software, digital electronics engineering).

25.2 Why is Software a Bottleneck in Product Development?

25.2.1 Growth of software effort

Software is a relative young discipline. The amount of software in systems is growing exponentially. The contribution of different disciplines to the system, measured in effort is shifting continuously. Figure 25.1 shows the growth of effort to make software and the related relative decrease of the other disciplines.

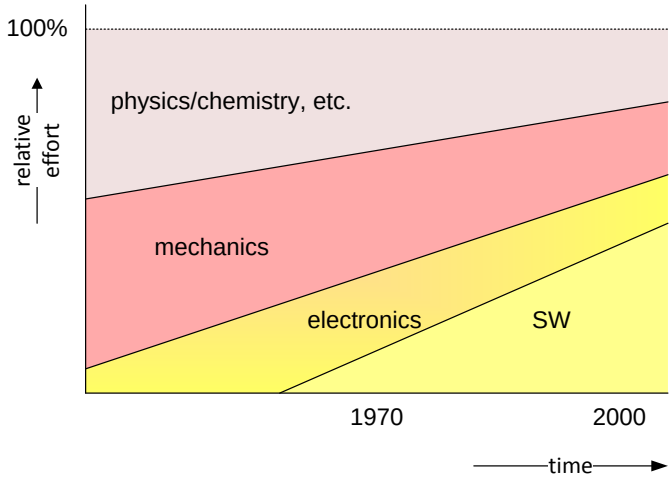


Figure 25.1: The relative contribution of software effort as function of time

25.2.2 Roles of the disciplines in a system

The different disciplines do have an asymmetric relation when we look at the control in systems. Figure 25.2 shows a typical control hierarchy in a system. At the bottom we see the physical disciplines who realize physical devices and sensors. We prefer to keep these physical components independent from each other seen from control perspective. Safety provisions are the major exception to this rule.

The physical devices need actuation that is delivered by some analog (power) electronics, e.g. amplifiers. Note that there might be all kinds of conversions in between in the more complex reality, e.g. pressure in a hydraulic system, light in an optics system. Again we prefer to keep the analog electronics mutually independent. The analog electronics is controlled by digital electronics. The control stack continues with control software that sits on top of the digital hardware. Finally, application software determines what the control software should do. Hopefully, the human user is the person who is really in control.

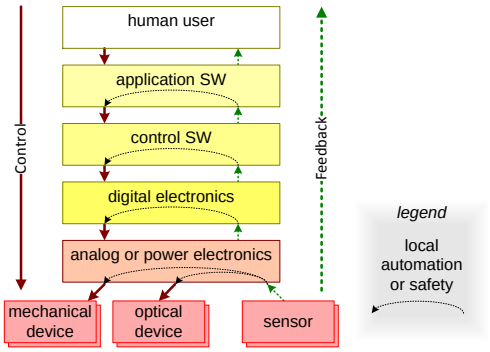


Figure 25.2: The Control Hierarchy of a system along the Technology dimension

Note that in all layers there are several reasons to have short cuts from sensors to control:

Safety is always kept as simple and direct as possible, since any complexity introduces new safety risks. A good safety design carefully allocates safety functions to the different layers to achieve the desired safety while achieving the desired control flexibility.

Automation can be done on lower layers if this simplifies the overall design. Automation provides value when the higher level work flows are well understood and well defined.

Performance is a special case of automation, where the short cut facilitates better performance, for example fast response times.

The software technology is in most modern systems the integrating technology, as shown by the control hierarchy. In the next section we will dive somewhat deeper in the relation between system qualities and software technology. In modern systems software technology determines to a high degree most system qualities. The inherent system qualities are often determined by the physical design, but the actually achieved quality is often determined by the way the software is constructed. For example, we can dimension a system with quite powerful motors to ensure high performance, but if the software does not fully utilize the motors, then the system performance is lower than can be expected from the physical design. Similarly for reliability that inherently is determined by the physical design. However, the software control may negatively impact reliability. For example, in a system with pumps, the software used a sequence where one of the pumps regularly ran dry. The consequence was that this pump failed often.

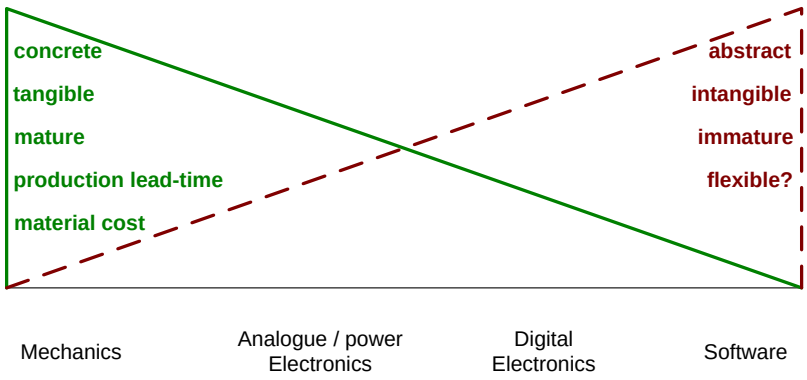


Figure 25.3: Characterization of disciplines, ordered along the level of abstraction

25.2.3 Characterization of disciplines

Physical disciplines work on aspects that can be touched, the subjects are tangible. Virtual disciplines work on abstract concepts, the subjects are intangible. Figure 25.3 shows the disciplines on an axes of decreasing tangibility and increasing abstractness. Mechanics is one of the older disciplines that is highly tangible. Analog (power) electronics is younger as discipline and less tangible. Digital electronics is again younger. Although the digital electronics itself can be touched, the circuitry itself is much more conceptual and abstract.

Figure 25.3 also provides a number of other characterizations that follow the same trend as tangibility and abstractness:

maturity The more tangible the more mature a discipline seems to be. Mature means here well known and founded; the discipline has an established and documented body of knowledge.

production lead time The physical world is constrained by nature. Processing and production of components have an inherent lead time. Software can be seen as infinitely fast. However, when testing, quality control and configuration management are included in the production lead time, then this lead time becomes strongly dependent on people, processes, and tools. Hence the question mark behind flexible at the right hand side of the figure.

material cost Physical systems do have inherent cost in the materials and its processing.

These differences in nature, especially *production lead time* and *material cost*, cause also differences in other business processes and the approach to life cycle aspects. For many physical components the logistics design is crucial for cycle time, stocks, and cost, where software does have zero reproduction cycle time, cost and infinite stocks.

25.3 System or Software Issues?

Systems can be specified in terms of their functionality and qualities. Most qualities of a system are strongly influenced or even determined by the software design. Figure 25.4 based on [19] shows a checklist for qualities. In this figure all qualities that have a strong or weak relation with the software design are highlighted.

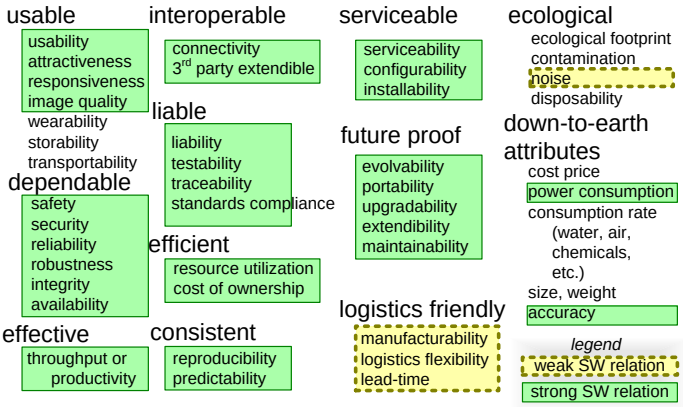


Figure 25.4: Quality Checklist annotated with the relation with software

During System Design the system is decomposed in subsystems and implementation technologies. The combination of subsystems and technologies together has to realize the qualities. During this step the contribution or the role of a subsystem and implementing technology is determined.

Figure 25.5 shows the System level design aspects that are strongly related to software. Figure 25.6 shows a list of mechanisms used by SW engineers. These mechanisms facilitate the system level design aspects mentioned in Figure 25.5.

Both *Quality Attributes* and *Design Aspects* are *System Level* issues, however most of these issues are predominantly influenced by the software. The System Architect should: define the system level **what**, co-design the system level **how** and be involved with the single technology or subsystem **how**.

Due to the strong Software impact the software architect should: understand/review the system level **what**, co-design the system level **how** and design the software **how**.

This requires significant domain know-how of the Software Architect, see [11].

Figures 25.5 and 25.6 contain too many design aspects and software mechanisms to discuss as part of this book. The main purpose of these lists is to show the variety of technology issues to be addressed by the software architect.

Many of the design aspects have a many to many relation to the software mechanisms. For example, the design strategies for *performance*, *safety*, and *security*

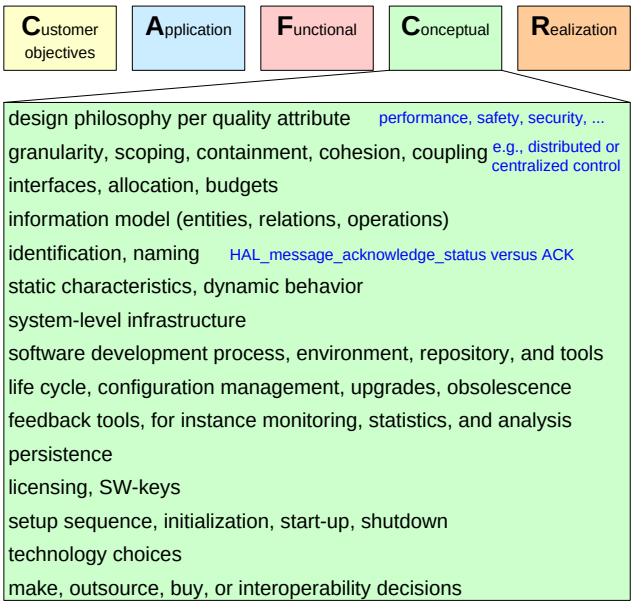


Figure 25.5: System design aspects that are strongly SW related

relate to nearly all software mechanisms. Vice versa most software mechanisms penetrate throughout most software and relate back to most of the design aspects.

The software part of systems is complex in itself. The software is a construct made by many people, stacking construct on construct. The risk is that software architects spend all their time internally in the software, while they also have to relate the software choices to the context, the system.

25.4 Acknowledgments

Jürgen Müller helped to sort out the attributes, aspects, mechanisms et cetera, which helps to position the Software Discipline in the System Development.

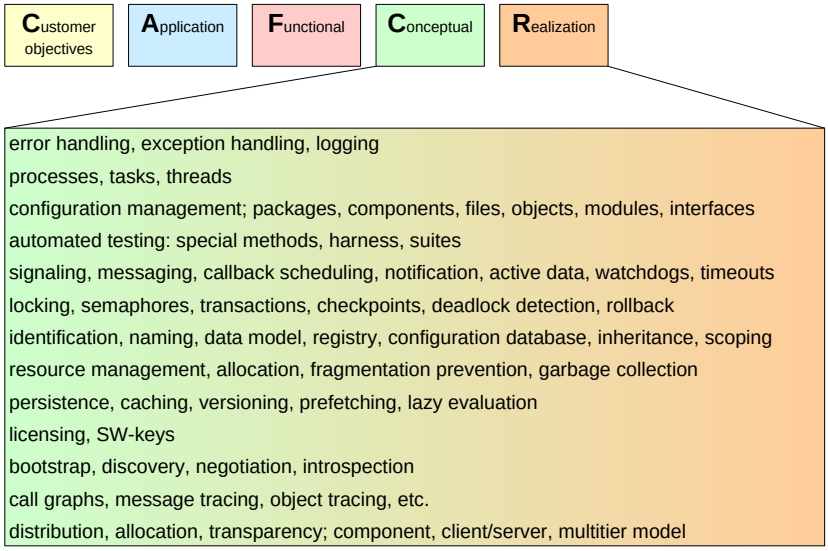


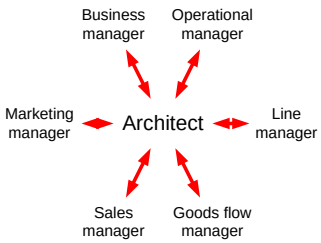
Figure 25.6: List of Software Mechanisms that are frequently applied to solve the system level design aspects

Part V

Management and Architects

Chapter 26

The Tense Relation between Architect and Manager



26.1 Introduction

The relation between managers and systems architects somehow tends to be somewhat difficult. This is not desired, since we position the systems architects as part of the leadership of an organization.

In this intermezzo we look at managers and architects in a generalized way. Generalizations are always risky; the purpose of this generalization is to better understand the inherent tensions between architects and managers. No “real” architect nor manager will exactly look like the generalization in this intermezzo.

26.2 What is a Manager?

A manager is someone who manages everything needed to get a task executed. The manager has the responsibility for the task. The responsibility comes with the required authority to do the task. Every Process in the simplified business decomposition in 1 and 3 generally has a manager associated with the process

who is responsible for the execution of that process. Often these tasks are further decomposed with managers associated with every subtask.

Systems architects frequently encounter the managers shown in Figure 26.1.

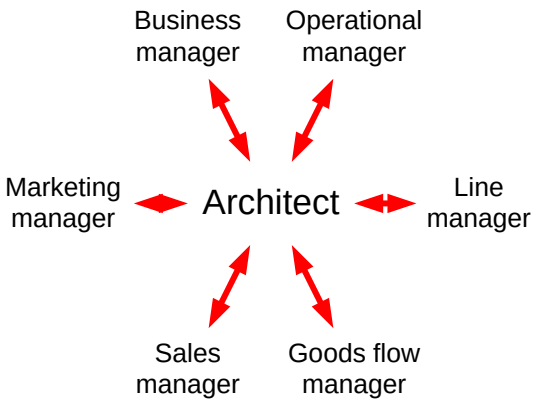


Figure 26.1: Managers frequently interacting with architects

26.3 Comparison of Architect and Manager

Figure 26.2 shows a comparison between architects and managers for 6 different aspects: *responsibility*, *view on solutions*, *view on changes*, *personal characteristics*, *leadership values*, and *personal ambition*.

26.3.1 Responsibility

Managers have a well defined responsibility, related to their function. In most organizations managers also are empowered accordingly. The scope of responsibility is limited, the total responsibility is divided over many managers.

The responsibility of the architect is much more fuzzy, see 9. For every aspect the architect is working on there is some manager who has the formal responsibility for that specific subject. The architect has limited formal power. At the other hand architects have a lot of informal influence.

26.3.2 View on Solutions

The view on solutions is quite different. The architect partially trusts his or her intuition, looking for the notion of an elegant solution. The word elegant can cover many aspects, such as: balanced, simple, beautiful. As representative of the stakeholders the architect will guard the fitness for use, is it the “right” solution? At the

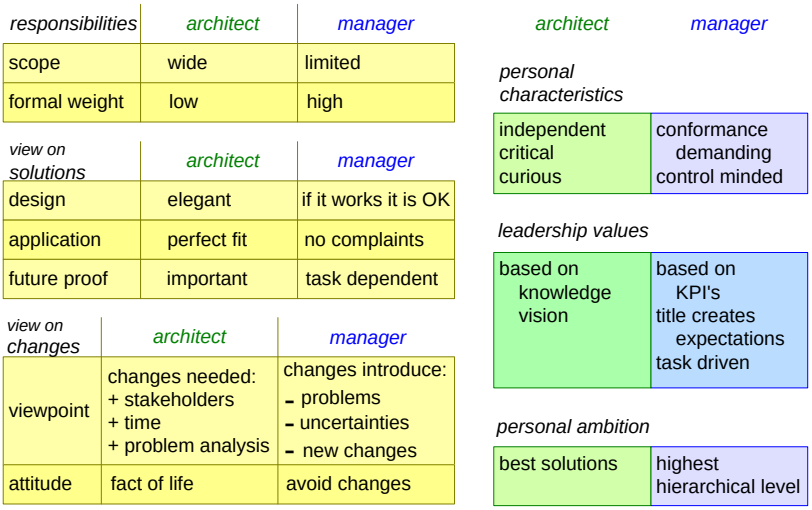


Figure 26.2: Comparison of caricature of architect and manager

same time the architect will place the solution in a time perspective, is the solution "future proof"?

Most managers stay close to their task and responsibility. A solution that matches the specification is by definition good. If there are no complaints, then there is no problem.

26.3.3 View on Changes

Architects (ought to) spend a significant part of their time in the turbulent outside world, inhabited with demanding customers in changing markets with aggressive inventive competitors, and innovative suppliers. At the same time architects are active in the company across many internal boundaries, enabling architects to detect, analyze and to help solving many internal problems. Architects are continuously confronted with situations where change is required. The internal and the external world are highly dynamic, causing need for change everywhere. Architects see changes as a fact of life.

Managers tend to take an opposite view on the need for change caused by the limited scope and the heavy weight of the responsibility of their task results. Managers have experienced that changes always introduce problems, involve uncertainties, and trigger more changes. The resulting behavior is to avoid changes/footnote Keep aware that we discuss caricatures of architects and managers. In practice there are many (bad) architects behaving very conservative..

26.3.4 Personal Characteristics

Managers are control minded, managers like to be in control of the task being performed; that is exactly their job. Managers demand conformance as a means to be in control. The people working at a task have to conform to the way the manager wants to perform the task.

Architects have an entirely different personality. Architects need independence and curiosity to be able to act as representative of the stakeholders. At the same time architects need to be critical, is this the best way to do address the task?

26.3.5 Leadership Values

Many organizations still think in hierarchical terms. Hence the manager is seen as the person who sets the direction. However, it is questionable if managers do have the appropriate knowledge and vision to determine the direction.

Architects have a broad perspective and know how, while (good) architects also have vision. This is a natural combination to provide true leadership.

Some architects are handicapped by an introvert personality making it difficult to "sell" the vision and to take the leaders position. It will be clear that team-work of manager and architect will work wonders in such a case.

26.3.6 Personal Ambition

The personal ambition of managers and architects are opposite as well. Many managers are driven by normal career incentives: higher position, power, status and more money. Architects seem to be driven by the case at hand, they want to achieve the "best" solution.

This difference in ambition makes the architect difficult to control, because architects are rather insensitive for the normal incentives, such as promotions and salary raises.

26.4 How to improve the relationship

The starting point for any solution is the recognition of the problem. This intermezzo is primarily provided to create awareness of the problem that there is tension between architects and managers. No silver bullets are given here as solution.

A quite promising direction to address this problem is modern management techniques, see Figure 26.3 for a list of suggestions.

The architect plays a vital role in bootstrapping these management techniques. In many techniques the architect plays the role of catalyst due to the combination of personal characteristics such as independence and know how. If the architect hides in technological solutions, then the architect does not trigger the required change.

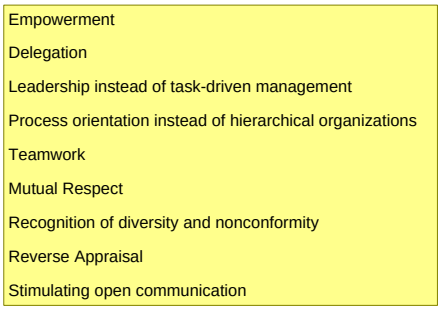


Figure 26.3: List of modern management techniques that can be used to improve the relation between managers and architects

We can also work at both sides to improve this relationship. Architects can be stuck in the solution world with little attention for all non technical aspects that determine the architecture. A vital step is that architects learn to communicate better what the impact of technical choices is on the less technical business aspects. Once architects are able to communicate more clearly with managers, then their recognition and influence will increase. See the next chapter for a further elaboration.

Many managers do not know what to expect from architects. It helps managers if they do understand the role of the architects better, so that they can ask the right questions and provide coaching. This book can be used in courses directed to management teams to help them to understand the architecting role.

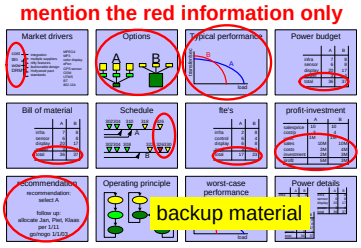
26.5 Acknowledgments

Jürgen Müller attended me on the fact that telling only the negative (The relationship is tense) is not good enough. An architect should always look for the constructive way out. I therefore added section **??**. Wil Hoogenstraaten also commented that the described relationship is well recognizable, but how to escape from this situation?

Louis Rubinfield pointed out the importance of communication, which resulted in adding "stimulating open communication" as improvement means.

Chapter 27

How to present architecture issues to higher management



27.1 Introduction

The architect bridges the technology world with the other business related worlds, by understanding these other worlds and by having ample know-how of technologies. Management teams are responsible for the overall business performance, which in the end is expressed in financial results.

Many architects and management teams are captured in a vicious circle:

- architects complain about management decisions and lack of know-how of managers
- managers complain about lack of input data and invisible architects

One way to break this vicious circle is to improve the managerial communication skills of architects. We address a frequently needed skill: presenting an architecture issue to a management team.

The architect should contribute to the managerial decision process by communicating technology options and consequences of technological decisions. Figure 27.1 shows a number of the relevant, somewhat overlapping, viewpoints. The figure indicates what links architects should communicate to management teams.

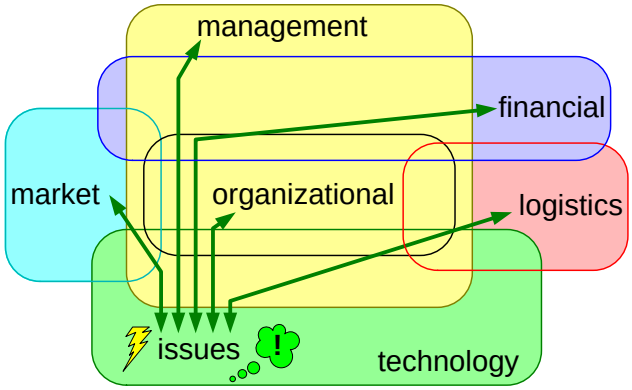


Figure 27.1: Architectural issues related to managerial viewpoints

common characteristics

+ action-oriented
+ solution rather than problem
+ impatient, busy
+ want facts not beliefs
+ operate in a political context
+ bottom-line oriented:
  profit, return on investment,
  market share, etc.

highly variable characteristics

? technology knowledge
  from extensive to shallow

? style from power play to
  inspirational leadership

Figure 27.2: Characteristics of managers in higher management teams

Architects must have a good understanding of their target audience. Figure 27.2 characterizes the managers in management teams. Their main job is to run a healthy business, which explains many of these characterizations: *action oriented*, *solution rather than problem*, *impatient, busy*, *bottom-line oriented:* profit, return on investment, market share, et cetera, and *want facts not believes*. These managers operate with many people all with their own personal interests. This means that managers *operate in a political context* (something which architects like to ignore).

Some characteristics of management teams depend on the company culture. For example, the amount of technology know-how can vary from extensive to shallow. Or, for example, the management style can vary from power play to inspirational leadership.

27.2 Preparation

Presentations to higher management teams must always be prepared with multiple people: a small preparation team. The combined insights of the preparation team enlarge the coverage of important issues, both technical as well as business. the combined understanding of the target audience is also quite valuable. Figure 27.3 shows how to prepare the content of the presentation as well as how to prepare for the audience.

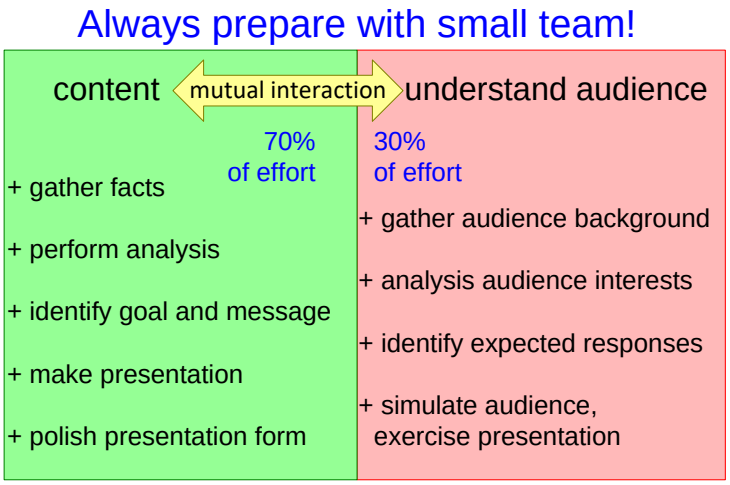


Figure 27.3: How to prepare

The content of the presentation must be clear, address the main issues, and convey the message, see also 27.3. The message must have substance for managers, which means that it should be *fact based*. The first steps are *gathering facts* and *performing analysis*. Based on these facts the *goal* and *message* of the presentation must be articulated. All this information must be combined in a *presentation*. When the presentation content is satisfactory the form must be polished (templates, colors, readability, et cetera). Although this has been described as a sequential process, the normal incremental spiral approach should be followed, going through these steps in 2-3 passes.

The members of management teams operate normally in a highly political context, mutually as well as with people in their context. This politics interferes significantly with the decision making. The political situation should be mapped by the preparation team, the political forces must be identified and understood. This is done by *analyzing the audience*, their *background* and their *interests*. The preparation team can gain a lot of insight by discussing the *expected responses* of the management team. At some moment the preparation team can *simulate* (role-play) the management team in a proof-run of the presentation. The understanding of the

audience must be used to select and structure the content part of the presentation. This activity should be time-multiplexed with the content preparation; 70% of the time working on content, 30% of the time for reflection and understanding of the audience.

27.3 The presentation material

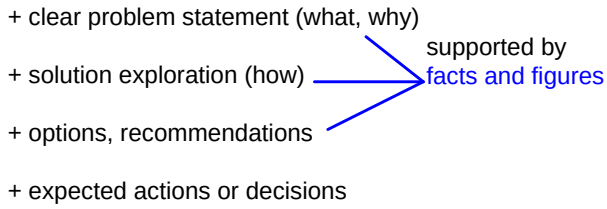


Figure 27.4: Recommended content

Figure 27.4 provides guidelines for the contents of the presentation. A clear *problem statement* and an *exploration of solution(s)* should address the technical issues as well as the translation to the business consequences. Normally a range of options are *provided* The options are *compared* and *recommendations* are provided. Note that options that are unfavorable from architectural point of view are nevertheless options. It is the challenge for the architect to articulate why these options are bad and should not be chosen. Architect enable and streamline the decision making by providing clear recommendations and by indicating what *actions* or *decisions* are required.

All content of the presentation should be to the point, *factual* and *quantified*. Quantified does not mean certain, often quite the opposite, future numbers are estimates based on many assumptions. The reliability of the information should be evident in the presentation. Many facts can be derived from the past. *Figures* from the past are useful to "calibrate" future options. Deviations from trends in the past are suspect and should be explained.

The presentation material should cover more than is actually being presented during the presentation itself. Some supporting data should be present on the sheets, without mentioning the data explicitly during the presentation. This allows the audience to assess the validity of the presented numbers, without the need to zoom in on all the details.

It is also useful to have additional backup material available with more in depth supporting data. This can be used to answer questions or to focus the discussion: speculation can be prevented by providing actual data.

The use of demonstrators and the show of artifacts (components, mock-ups)

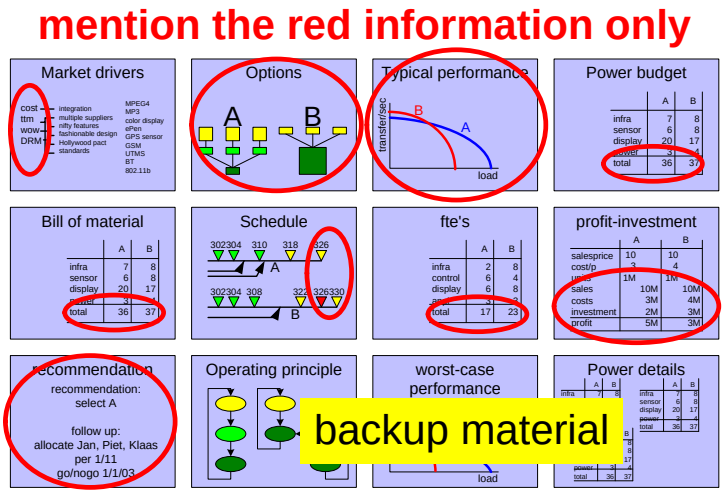


Figure 27.5: Mentioned info, shown info and backup info

makes the presentation more lively. The demonstrations should be short and attractive (from customer point of view), while illustrating the value and technological possibilities and issues.

poor form can easily distract from purpose and content

presentation material	presenter's appearance
+ professional	+ well dressed
+ moderate use of color and animations	+ self confident but open
+ readable	
+ use demos and show artifacts	but stay yourself, stay authentic

Figure 27.6: Form is important

Architects prefer to focus on the content, form is supportive to transfer the content. However architects should be aware that managers can be distracted by the form of a presentation, potentially spoiling the entire meeting by small issues. Figure 27.6 gives a number of recommendations with respect to the form of the presentation and the appearance of the presenter.

The presentation material (slides, demonstrators, video, drawings, et cetera)

has to look professional. Slides will use color and other presentation features. However, moderation in the use of colors, animations and other presentation features is recommended; an overload of these colors and features does not look professional and will distract the audience from the actual content. Information on the slides has to be readable: use large enough fonts and use sufficient contrast with the background. Pay special attention to quality and readability, when copy-pasting information from other sources. Sometimes it is better to recreate a high quality table or graph than to save effort by copy-pasting an unreadable table or graph.

The appearance of the presenter can also make or break the presentation. The presenter should give sufficient attention to clothes and overall appearance. Don't exaggerate this, you should stay yourself and still be authentic. Other people immediately sense it when the appearance is too exaggerated, which is also damaging for your image.

27.4 The Presentation

do not	*do*
- preach beliefs	+ quantify, show figures and facts
- underestimate technology knowledge of managers	+ create faith in your knowledge
- tell them what they did wrong	+ focus on objectives
- oversell	+ manage expectations

Figure 27.7: Don't force your opinion, understand the audience

Figures 27.7 and 27.8 show in the *don't* column a number of pitfalls for an architect when presenting to higher management teams. The preferred interaction pattern is given in the *do* column.

The pitfalls in Figure 27.7, *preaching believes*, *underestimating know-how of managers*, and *telling managers what they did wrong*, are caused by insufficient understanding of the target audience. In these cases the opinion of the architect is too dominant, opinions work counterproductive. *Overselling* creates a problem for the future: expectations are created that can not be met. The consequence of overselling is loss of credibility and potentially lack of support in tougher times. Architects must *manage* the *expectations* of the audience.

When presenting the architect tries to achieve multiple objectives:

- Create awareness of the problem and potential solutions by *quantification* and by *showing figures and facts*.

- Show architecting competence in these areas, with the message being: “you, the manager, can delegate the technical responsibility to me”. This creates *faith* in the *architect’s know-how*.
- Facilitate decision making by translating the problem and solution(s) in business consequences, with the *focus on objectives*.

This means that sufficient technological content need to be shown, at least to create faith in the architect’s competence. Underestimation of the managerial know-how is arrogant, but mostly very dangerous. Some managers have a significant historic know-how, which enable them to assess strengths and weaknesses quickly. Providing sufficient depth to this type of manager is rewarding. The less informed manager does not need to fully understand the technical part, but at least should get the feeling that he or she understands the issues.

do not	*do*
- let one of the managers hijack the meeting	+ maintain the lead
- build up tensions by withholding facts or solutions	+ be to the point and direct
- be lost or panic at unexpected inputs or alternatives	+ acknowledge input, indicate consequences (facts based)

Figure 27.8: How to cope with managerial dominance

The impatience and action orientation of managers makes them very dominant, with the risk that they take over the meeting or presentation. Figure 27.8 shows a number of these risks and the possible counter measures:

Managers hijacking the meeting can be prevented by maintaining the lead as presenter.

Build up tensions by withholding facts or solutions, but be to the point and direct. For example, it can be wise to start with a summary of the main facts and conclusions, so that the audience know where the presentation is heading.

Be lost or panic at unexpected inputs or alternatives. Most managers are fast and have a broad perspective that helps them to come with unforeseen options. Acknowledge inputs and indicate the consequences of alternatives as far as you can see them (fact based!).

An example of an unexpected input might be to outsource a proposed development to a low-cost country. The outsourcing of developments of core components might require lots of communication and traveling, creating costs. Such consequence has to be put on the table, but refrain from concluding that it is (im)possible.

27.5 Exercise

The SARCH course [12] on System Architecting contains an exercise, where the participants can apply then lessons learned by giving a presentation to a (simulated) management team. The presenter gives his presentation for the participants and the teacher, who play the role of this higher management team.

+ Bring a clear architecture message to

+ a Management team at least 2 hierarchical levels higher

+ with 10 minutes for presentation including discussion
  (no limitation on number of slides)

* architecture message =
  technology options in relation with market/product

* address the concerns of the management stakeholders:
  translation required from technology issues into
  business consequences (months, fte's, turnover, profit, investments)

Figure 27.9: Exercise presentation to higher management

Figure 27.9 shows the description of this exercise. The group of participants is divided in 4 teams of about 4 people, preferably from the same domain. These teams have somewhat less than 2 hours for the preparation of the presentation. The exercise is explained to them several days before and the teams are also formed days before. This enables the team to determine a subject and message in a background process, during lunch and in the breaks. Determining the subject and message requires quite some elapsed time. It is highly recommended to take a subject from *real-life*: "What you always wanted to tell topmanagement".

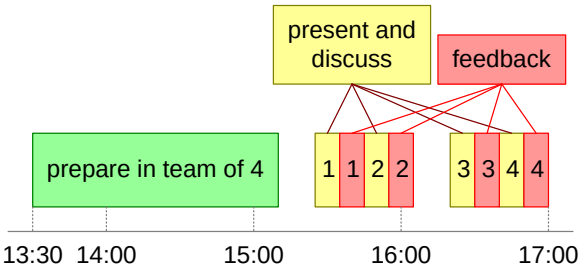


Figure 27.10: Schedule of the presentation exercise

Figure 27.10 shows the schedule of the exercise. Every presentation is 10 minutes sharp, **including** the interaction with the management team. Directly after

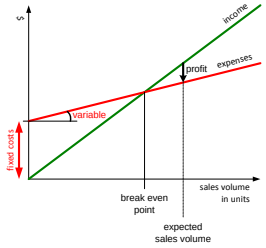
the presentation feedback is given by the participants as well as by the teacher. This feedback should follow the normal feedback guidelines: mentioning the strong points, before discussing the options for improvement. The teacher must ensure that sufficient feedback is given, the material in this exercise can be used as guideline.

The limited preparation time implies that the result will also be limited. The form will be limited (handwritten flipovers) and most of the historical data will be made up.

The teacher should stimulate the complete group to really participate in the role play, it can also be a lot of fun.

Chapter 28

Simplistic Financial Computations for System Architects.



28.1 Introduction

Many system architects shy away from the financial considerations of the product creation. In this document a very much simplified set of models is offered to help the architect in exploring the financial aspects as well. This will help the architect to make a ”sharper” design, by understanding earlier the financial aspects.

The architect should always be aware of the many simplifications in the models presented here. Interaction with real financial experts, such as controllers, will help to understand shortcomings of these models and the finesses of the highly virtualized financial world.

In Section 28.2 a very basic cost and margin model is described. Section 28.3 refines the model at the cost side and the income side. In Section 28.4 the time dimension is added to the model. Section 28.5 provides a number of criteria for making finacial decisions.

28.2 Cost and Margin

The simplest financial model looks only at the selling price (what does the customer pay), the cost price (how much does the manufacturing of the product actually cost). The difference of the selling price and the cost price is the margin. Figure 28.1 shows these simple relations. The figure also adds some annotations, to make the notions more useful:

- the cost price can be further decomposed in material, labor and other costs
- the margin (”profit per product”) must cover all other company expenses, such as research and development costs, before a real profit is generated
- most products are sold as one of the elements of a value chain. In this figure a retailer is added to show that the street price, as paid by the consumer, is different from the price paid by the retailer[1].

The annotation of the other costs, into transportation, insurance, and royalties per product, show that the model can be refined more and more. The model without such a refinement happens to be rather useful already.

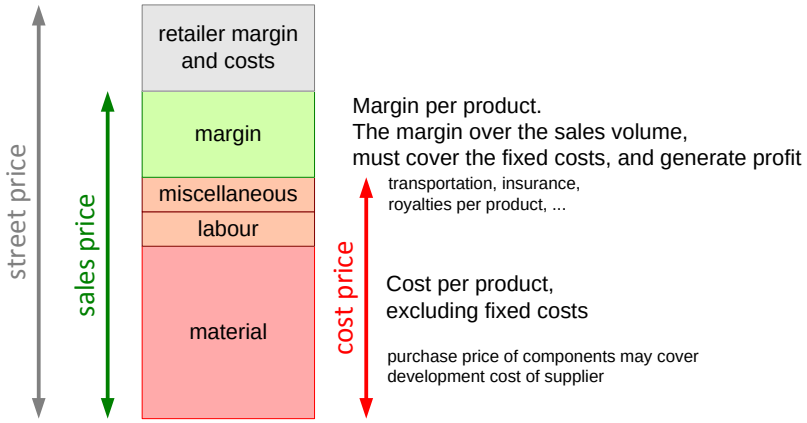


Figure 28.1: The relation between sales price, cost price and margin per product

The translation of margin into profit can be done by plotting income and expenses in one figure, as shown in Figure 28.2, as function of the sales volume. The slope of the expenses line is proportional with the costs per product. The slope of the income line is proportional with the sales price. The vertical offset of the expenses line are the fixed organizational costs, such as research, development, and overhead costs. The figure shows immediately that the sales volume must exceed the break even point to make a profit. The profit is the vertical distance between expenses and income for a given sales volume. The figure is very useful to obtain insight in

the robustness of the profit: variations in the sales volume are horizontal shifts in the figure. If the sales volume is far away from the break even point than the profit is not so sensitive for the the volume.

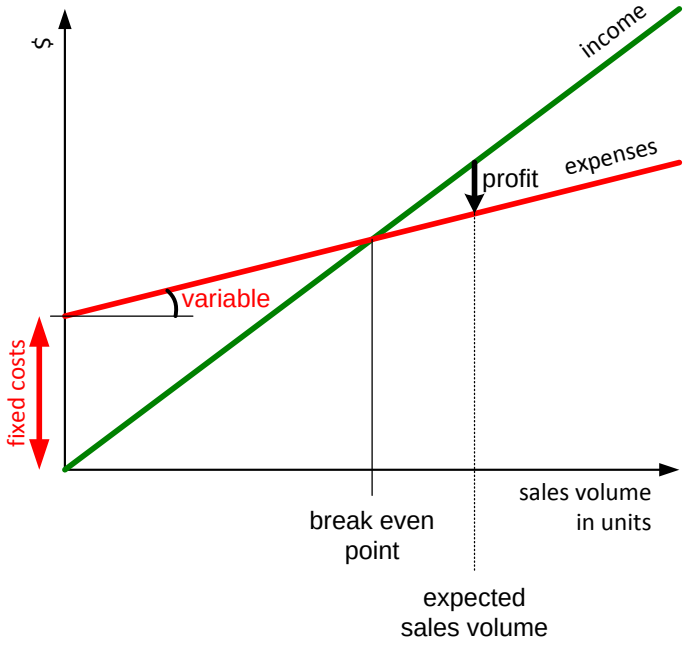


Figure 28.2: Profit as function of sales volume

28.3 Refining investments and income

The investments as mentioned before may be much more than the research and development costs only, depending strongly on the business domain. Figure 28.3 shows a decomposition of the investments. The R&D investments are often calculated in a simple way, by using a standard rate for development personnel that includes overhead costs such as housing, infrastructure, management and so on. The investment in R&D is then easily calculated as the product of the amount of effort in hours times the rate (=standardized cost per hour). The danger of this type of simplification is that overhead costs become invisible and are not managed explicitly anymore.

Not all development costs need to be financed as investments. For outsourced developments an explicit decision has to be made about the financing model:

- the supplier takes a risk by making the investments, but also benefits from

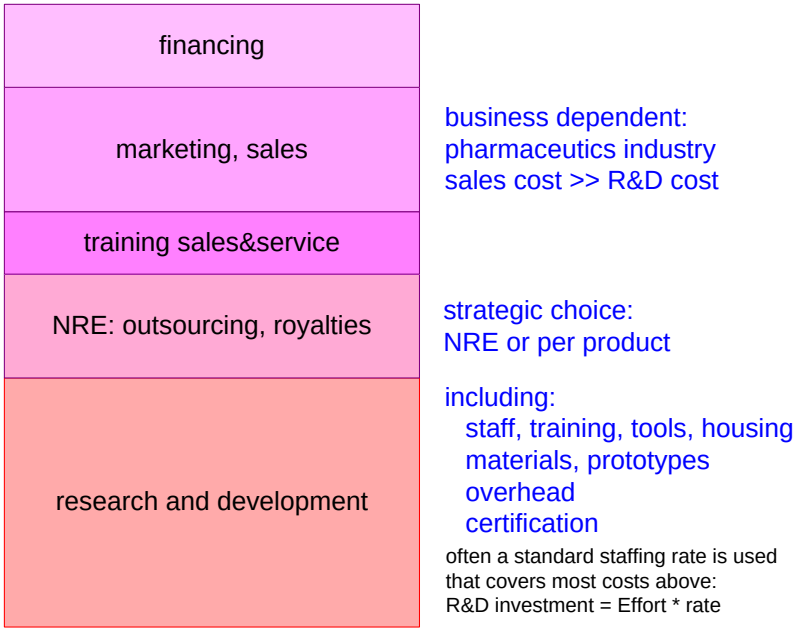


Figure 28.3: Investments, more than R&D

larger sales volumes

- the company pays the investment, the so called Non Recurring Engineering (NRE) costs. In this case the supplier takes less risks, but will also benefit less from larger sales volumes.

If the supplier does the investment than the development costs of the component are part of the purchasing price and become part of the material price. For the NRE case the component development costs are a straightforward investment.

Other investments to be made are needed to prepare the company to scale all customer oriented processes to the expected sales volume, ranging from manufacturing and customer support to sales staff. In some business segments the marketing costs of introducing new products is very significant. For example, the pharmaceutical industry spends 4 times as much money on marketing than on R&D. The financial costs of making investments, such as interest on the capital being used, must also be taken into account.

We have started by simplifying the income side to the sales price of the products. The model can be refined by taking other sources of income into account, as shown in Figure 28.4. The options and accessories are sold as separate entities, generating a significant revenue for many products. For many products the base products are sold with a loss. This loss is later compensated by the profit on options and accessories.

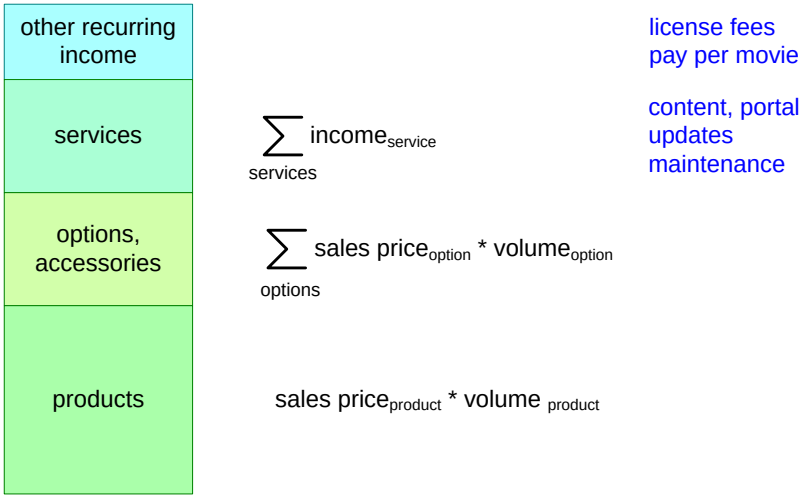


Figure 28.4: Income, more than product sales only

Many companies strive for a business model where a recurring stream of revenues is created, for instance by providing services (access to updates or content), or by selling consumables (ink for prink jet printers, lamps for beamers, et cetera).

One step further is to tap the income of other players of the value chain. Example is the license income for MPEG4 usage by service and content providers. The chip or box supplier may generate additional income by partnering with the downstream value chain players.

28.4 Adding the time dimension

All financial parameters are a function of time: income, expenses, cash-flow, profit, et cetera. The financial future can be estimated over time, for example in table form as shown in Figure 28.5. This table shows the investments, sales volume, variable costs, income, and profit (loss) per quarter. At the bottom the accumulated profit is shown.

The cost price and sales price per unit are assumed to be constant in this example, respectively 20k$ and 50k$. The formulas for variable costs, income and profit are very simple:

$$variable\ costs = sales\ volume * cost\ price$$

$$income = sales\ volume * sales\ price$$

$$profit = income - (investments + variable\ costs)$$

	Y1 Q1	Y1 Q2	Y1 Q3	Y1 Q4	Y2 Q1	Y2 Q2	Y2 Q3
investments	100k$	400k$	500k$	100k$	100k$	60k$	20k$
sales volume (units)	-	-	2	10	20	30	30
material & labour costs	-	-	40k$	200k$	400k$	600k$	600k$
income	-	-	100k$	500k$	1000k$	1500k$	1500k$
quarter profit (loss)	(100k$)	(400k$)	(440k$)	200k$	500k$	840k$	880k$
cumulative profit	(100k$)	(500k$)	(940k$)	(740k$)	(240k$)	600k$	1480k$

cost price / unit = 20k$
sales price / unit = 50k$

variable cost = sales volume * cost price / unit
income = sales volume * sales price / unit
quarter profit = income - (investments + variable costs)

Figure 28.5: The Time Dimension

Figure 28.6 shows the cumulative profit from Figure 28.5 as a graph. This graph is often called a ”hockey” stick: it starts with going down, making a loss, but when the sales increase it goes up, and the company starts to make a profit. Relevant questions for such a graph are:

- when is profit expected?
- how much loss can be permitted in the beginning?
- what will the sustainable profit be in later phases?

These questions can also be refined by performing a simple sensitivity analysis. Figure 28.7 shows an example of such an analysis. Two variations of the original plan are shown:

- a development delay of 3 months
- an intermediate more expensive product in the beginning, followed by a more cost optimized product later

The delay of 3 months in development causes a much later profitability. The investment level continues for a longer time, while the income is delayed. Unfortunately development delays occur quite often, so this delayed profitability is rather common. Reality is sometimes worse, due to loss of market share and sales price erosion. This example brings two messages:

- a go decision is based on the combination of the profit expectation and the risk assessment
- development delays are financially very bad

The scenario starting with a more expensive product is based on an initial product cost price of 30k$. The 20k$ cost price level is reached after 1 year. The benefit of an early product availability is that market share is build up. In

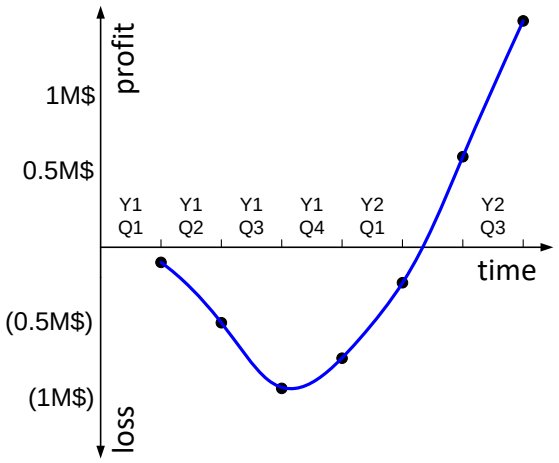


Figure 28.6: The “Hockey” Stick

this example the final market share in the first example is assumed to be 30 units, while in the latter scenario 35 units is used. The benefits of this scenario are mostly risk related. The loss in the beginning is somewhat less and the time to profit is somewhat better, but the most important gain is be in the market early and to reduce the risk in that way. An important side effect of being early in the market is that early market feedback is obtained that will be used in the follow on products.

In reality, a company does not develop a single product or system. After developing an initial product, it will develop successors and may be expand into a product family. Figure reffig:SFCmultipleDevelopments shows how the cumulative profits are stacked, creating an integral hockey stick for the succession of products. In this graph the sales of the first product is reduced, while the sales of the second product is starting. This gradual ramp-up and down is repated for the next products. The sales volume for the later products is increasing gradually.

28.5 Financial yardsticks

How to assess the outcome of the presented simple financial models? What are *good* scenarios from financial point of view? The expectation to be profitable is not sufficient to start a new product development. One of the problems in answering these questions is that the financial criteria appear to be rather dynamic themselves. A management fashion influences the emphasis in these criteria. Figure 28.9 shows a number of metrics that have been fashionable in the last decade.

The list is not complete, but it shows the many financial considerations that play a role in decision making.

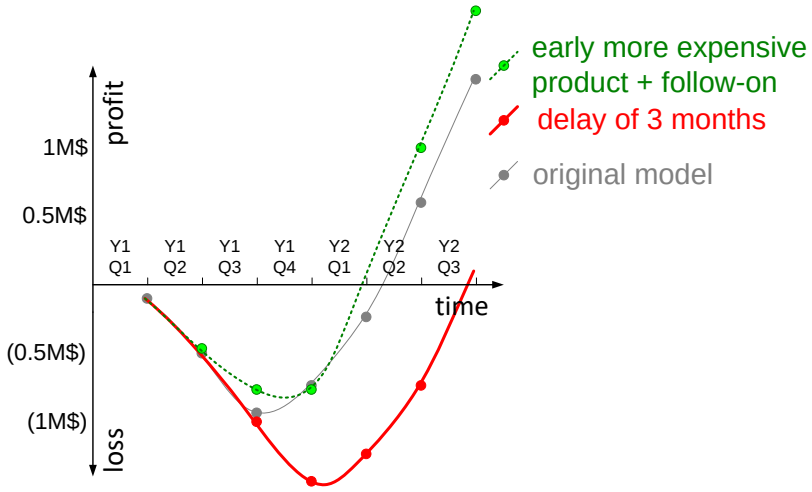


Figure 28.7: What if ...?

Return On Investments is a metric from the point of view of the shareholder or the investor. The decision these stakeholders make is: what investment is the most attractive.

Return On Net Assets (RONA) is basically the same as ROI, but it looks at all the capital involved, not only the investments. It is a more integral metric than ROI.

turnover / fte is a metric that measures the efficiency of the human capital. Optimization of this metric results in a maximum added value per employee. It helps companies to focus on the core activities, by outsourcing the non-core activities.

market ranking (share, growth) has been used heavily by the former CEO of General Electric, Jack Welch. Only business units in rank 1, 2 or 3 were allowed. Too small business units were expanded aggressively if sufficient potential was available. Otherwise the business units were closed or sold. The growth figure is related to the shareholder value: only growing companies create more shareholder value.

R&D investment / sales is a metric at company macro level. For high-tech companies 10% is commonly used. Low investments carry the risk of insufficient product innovation. Higher investments may not be affordable.

cashflow is a metric of the actual liquid assets that are available. The profit of a company is defined by the growth of all assets of a company. In fast growing

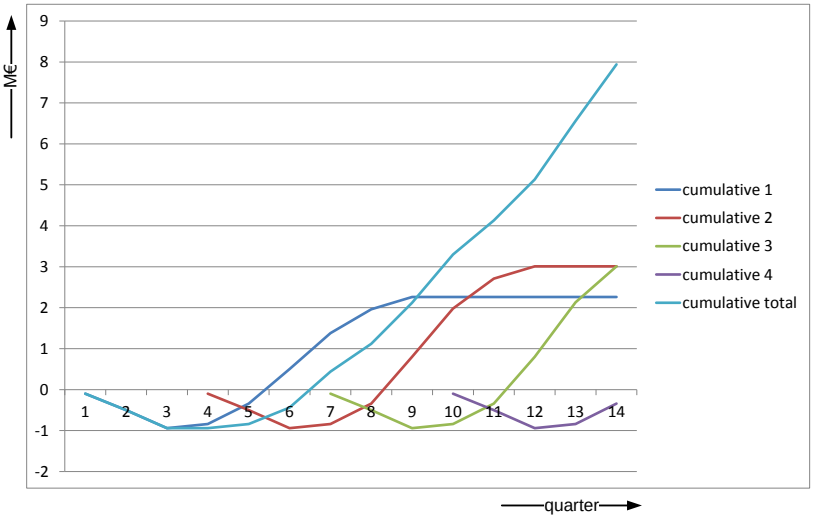


Figure 28.8: Stacking Multiple Developments

companies a lot of working capital can be unavailable in stocks or other non salable assets. Fast growing, profit making, companies can go bankrupt by a negative cash-flow. The crisis of Philips in 1992 was caused by this effect: years of profit combined with a negative cash-flow.

28.6 Acknowledgements

William van der Sterren provided feedback and references. Hans Barella, former CEO of Philips medical Systems, always stressed the importance of Figure 28.2, and especially the importance of a robust profit. Ad van den Langenberg pointed out a number of spelling errors.

Return On Investments (ROI)

Net Present Value

Return On Net Assets (RONA) leasing reduces assets, improves RONA

turnover / fte outsourcing reduces headcount, improves this ratio

market ranking (share, growth) "only numbers 1, 2 and 3 will be profitable"

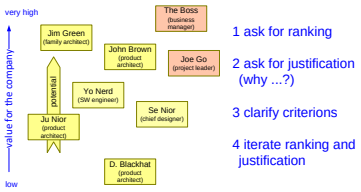
R&D investment / sales in high tech segments 10% or more

cash-flow fast growing companies combine profits with negative cash-flow, risk of bankruptcy

Figure 28.9: Fashionable financial yardsticks

Chapter 29

How to appraise or assess an architect?



29.1 Introduction

The responsibilities of system architects are ill defined. Either the responsibilities overlap significantly with other players in the Product Creation Process, or the responsibilities are very abstract and vague (not specific and measurable), see [17].

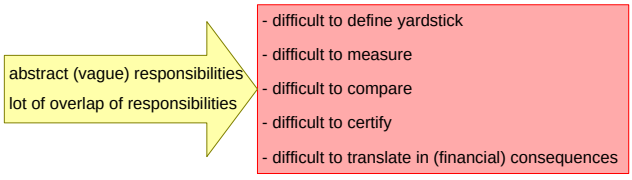


Figure 29.1: The function of an architect is difficult to evaluate

Figure 29.1 provides the problem statement: *How to asses the architect*, when it is difficult to define a yardstick, measurements, comparisons, or certifications due

to the ill defined responsibilities. The financial remuneration, which is normally based on measurements and comparisons also becomes very difficult.

Section 29.2 formulates the success criterions for architects. These criterions are used in section 29.3 to describe an assessment method.

29.2 When is the architect successful?

In [17] the deliverables, responsibilities and activities of the system architect are discussed. Figure 29.2 summarizes this article. The deliverables of the architect are abstract paperwork or electronic information, no tangible modules or systems. The primary responsibilities are not easily measured: how sound (balanced, well decomposed, consistent, et cetera) is the system specification and design? The architect is spending most of his time on activities which do not result in one of the deliverables and most of the activities do not directly contribute to the primary responsibilities. However all of these activities are indispensable for the role of the architect and together ensure the architecture quality.

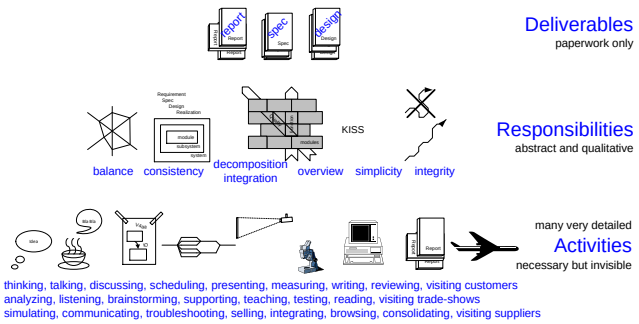


Figure 29.2: Tangible deliverables based upon many invisible activities

Figure 29.3 shows the architecting function and the criterions for successful architecting. Architecting is the transformation of problem and solution know how and often an already existing architecture into a new architecture. This process takes place in the context of many stakeholders, with their expectations, needs, concerns and constraints. The architecting is done by the product creation team (project leader, engineers, product manager and the system architect), although the architect should take the lead in this process.

The architect has played his role successful if the 2 criterions which are shown are fulfilled:

- the resulting architecture satisfies the stakeholders
- the architect has enabled the product creation team by leading the architecting process.

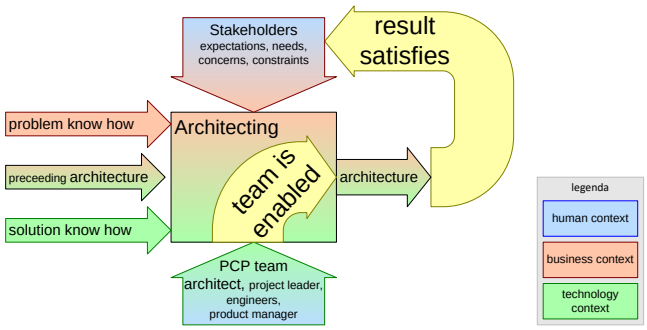


Figure 29.3: Criterions for successful architecting

29.3 How to assess the architect?

The criterions discussed in section 29.2 must be explored in order to facilitate the assessment of the architect. Most appraisal systems are based on formalized yardsticks, such as the (generic) function appraisal system, the (specific) job description and the (also specific) personal career development plan.

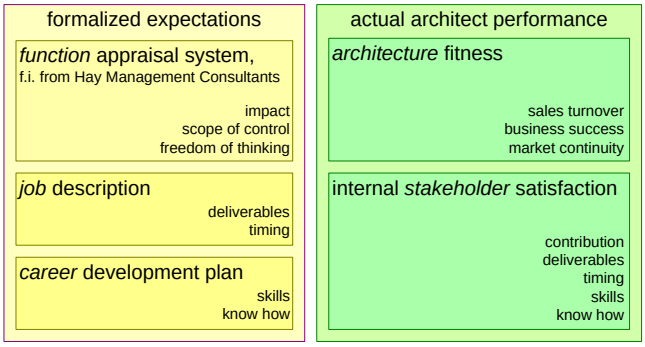


Figure 29.4: Yardsticks for architect assessment

Figure 29.4 shows the formal yardsticks at the left hand side. The main issues addressed in the yardsticks are also mentioned.

The function appraisal systems, such as defined by Hay Management, are based on parameters as *scope of control*, *impact* and *freedom of thinking*. The Hay management system is calibrated over multiple companies, domains and functions, by the active participation of the Hay Management company.

The experience is that the architect function does not easily fit in this method. ASML has defined all their functions in this system, with a multiple ladder approach and were able to fit the *system engineer* function in an acceptable way in this model. Other companies are struggling more with the architect function, due to

the problems described in section 29.1.

The reference for the individual appraisal is the specific *job description*, which defines the *deliverables* and the *timing*. Deliverables are a poor performance indicator, lots of paper is a sign of a bad architect! However a small amount of paper is not yet a sign of a good architect. Instead of measuring the deliverables the architecture fitness can be assessed, which in turn is a measure for the architecting contribution of the architect.

Complementary is the personal *career development plan*, which defines the desired *skills* and *know how*. The measurement of skills and know how can be done by assessing the internal stakeholder satisfaction.

The right hand side of figure 29.4 shows the actual architect performance, in terms of *architecture fitness* and internal *stakeholder* satisfaction. The architecture fitness is characterized by parameters such as *sales turnover*, *business success* and *market continuity*. The internal stakeholder satisfaction is characterized by the opinion of the stakeholders of the architects role in terms of *contribution*, *deliverables*, *timing*, *skills* and *know how*.

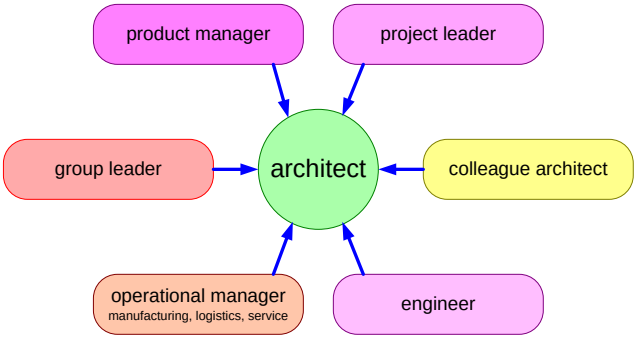


Figure 29.5: 360 degree assessment

An informal 360 degree approach can be used to ”measure” the internal stakeholder satisfaction with respect to the architect. A subset (3 to 6) of internal stakeholders is interviewed, where the performance of the architect is discussed in terms of *contribution*, *deliverables*, *timing*, *skills* and *know how*, see figure 29.5.

The stakeholders to be interviewed should have had sufficient interaction with the architect and should have complementary, somewhat overlapping viewpoints. By asking specific, but open questions, the role of the architect can be articulated.

Assessment is a relative act, in order to provide meaning to the input data, the data needs to be calibrated. This calibration can be done by comparing the architect being assessed with colleagues. It is useful to ask for a ranking with multiple colleagues, both architects and non architects. The ranking question asked to the interviewees has mostly a trigger function: by forcing a one dimensional comparison the performance in different dimensions has to be combined in a single

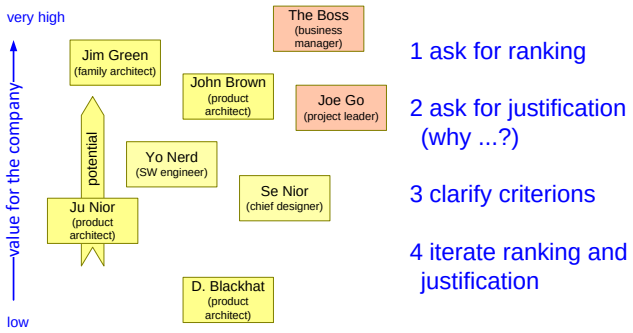


Figure 29.6: Ranking as trigger for discussions

assessment figure.

The relative position and the distance between ranked people will generate new questions: ”Why do you think that Yo Nerd has a greater value than Se Nior?". Also the differences in ranking between interviewees gives a lot of insight in the (often implicit) criterions which are used by the interviewees, for instance: ”Ju Nior is highly valued by the engineer for his excellent technical solutions, while the product manager criticizes him for not listening to the customer”.

Bibliography

[1] Mark Abraham. Define and price for your market starting at end market values! `http://www.sticky-marketing.net/articles/pricing-for-channels.htm`, 2001.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA, 2000.

[3] Dana Bredemeyer and Ruth Malan. Role of the software architect. `http://www.bredemeyer.com/pdf_files/role.pdf`, 1999.

[4] Jean-Marc DeBaud and Klaus Schmid. A systematic approach to derive the scope of software product lines. In *21^{st} international Conference on Software Engineering: Preparing for the Software Century*, pages 34–47. ICSE, 1999.

[5] K. Frampton, J. M. Carroll, and J. A. Thom. What capabilities do IT architects say they need? In *10th United Kingdom Academy for Information Systems (UKAIS) Proceedings*, 2005.

[6] Thomas Gilb. *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Elsevier Butterworth-Heinemann, London, 2005.

[7] W. Huitt. Maslow's hierarchy of needs. *Educational Psychology Interactive. Valdosta, GA: Valdosta State University.*, 2004.

[8] Ivar Jacobson, Grady Booch, and James Rumbaugh. *The Unified Software Development Process*. Addison-Wesley, Reading, MA, 1999.

[9] Ivar Jacobson, Martin Griss, and Patrik Jonsson. *Software Reuse; Architecture, Process and Organization for Business Success*. ACM Press, New York, 1997.

[10] Klaus Kronlöf, editor. *Method Integration; Concepts and Case Studies*. John Wiley, Chichester, England, 1993. A useful introduction is given in Chapter 1, The Concept of Method Integration.

[11] Philip Kruchten. The software architect- and the software architecture team. In *Software Architecture; TC2 First Working IFIP Conference on Software Architecture (WICSA1)*, pages 565–583. IFIP, 1999. This article describes required skills for architect and architecture team; traps and pitfalls; Personality profile based on Myers-Briggs Type Indicator.

[12] Gerrit Muller. CTT course SARCH. `http://www.gaudisite.nl/SARCHcoursePaper.pdf`, 1999.

[13] Gerrit Muller. Product families and generic aspects. `http://www.gaudisite.nl/GenericDevelopmentsPaper.pdf`, 1999.

[14] Gerrit Muller. Requirements capturing by the system architect. `http://www.gaudisite.nl/RequirementsPaper.pdf`, 1999.

[15] Gerrit Muller. Roadmapping. `http://www.gaudisite.nl/RoadmappingPaper.pdf`, 1999.

[16] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

[17] Gerrit Muller. The role and task of the system architect. `http://www.gaudisite.nl/RoleSystemArchitectPaper.pdf`, 2000.

[18] Gerrit Muller. Architectural reasoning explained. `http://www.gaudisite.nl/ArchitecturalReasoningBook.pdf`, 2002.

[19] Gerrit Muller. CAFCR: A multi-view method for embedded systems architecting: Balancing genericity and specificity. `http://www.gaudisite.nl/ThesisBook.pdf`, 2004.

[20] Henk Obbink, Jürgen Müller, Pierre America, and Rob van Ommering. COPA: A component-oriented platform architecting method for families of software-intensive electronic products. `http://www.hitech-projects.com/SAE/COPA/COPA_Tutorial.pdf`, 2000.

[21] Eberhardt Rechtin and Mark W. Maier. *The Art of Systems Architecting*. CRC Press, Boca Raton, Florida, 1997.

[22] Carnegie Mellon Software Engineering Institute SEI. Software engineering management practices. `http://www.sei.cmu.edu/managing/managing.html`, 2000.

History

Version: 1.8, date: July 28, 2010 changed by: Gerrit Muller

- added ToolBoxSystemArchitect, ChangeManagementIntoducingSystems

Version: 1.7, date: July 10, 2010 changed by: Gerrit Muller
- added BusinessStrategyMethodsModels

Version: 1.6, date: July 7, 2010 changed by: Gerrit Muller
- added Dynamic Range of Abstraction Levels in Architecting

Version: 1.5, date: June 30, 2010 changed by: Gerrit Muller
- refactored chapter requirements into: CAFCR introduction, Fundamentals of Requirements Engineering, Key Driver How to, Requirements Elicitation and Selection.

Version: 1.4, date: June 26, 2010 changed by: Gerrit Muller
- added chapter "Product, Projects, and Services"
- added chapter "Systems Titles and Roles"

Version: 1.3, date: June 3, 2004 changed by: Gerrit Muller
- added chapter '"Simplistic Financial Computations"

Version: 1.2, date: May 28, 2004 changed by: Gerrit Muller
- divided the book in Parts
- removed the chapter about Medical Imaging

Version: 1.1, date: March 29, 2004 changed by: Gerrit Muller
- added "Architecting Styles"

Version: 1.0, date: June 12, 2003 changed by: Gerrit Muller
- added "How to appraise or assess an architect?"
- added refactoring notice to the abstract and introduction

Version: 0.9, date: January 21, 2003 changed by: Gerrit Muller
- minor changes

Version: 0.8, date: November 1, 2002 changed by: Gerrit Muller
- Added chapter "How to present architecture issues to higher management"

Version: 0.7, date: October 3, 2002 changed by: Gerrit Muller
- Added chapter "The role of roadmapping in the strategy process"
- added chapter "Market Product lifecycle consequences for architecting"
- changed order of chapters "Roadmapping" and "Requirements"

Version: 0.6, date: August 5, 2002 changed by: Gerrit Muller
- Changed title to "System Architecting"

Version: 0.5, date: June 22, 2001 changed by: Gerrit Muller
- corrected part of missing or wrong references and citations

Version: 0.4, date: February 6, 2001 changed by: Gerrit Muller
- Added chapter "Function Profiles; The sheep with 7 legs"

Version: 0.3, date: November 1, 2000 changed by: Gerrit Muller
- Added chapter "Role and Task of the System Architect"
- Added chapter "Role of Software in Complex Systems"

Version: 0.2, date: March 29, 2000 changed by: Gerrit Muller
- Added chapter "Product Families Business Analysis and Family Definition"

Version: 0.1, date: March 24, 2000 changed by: Gerrit Muller
- Added Preface; System Architecture: The Golden Bullet?

Version: 0, date: March 21, 2000 changed by: Gerrit Muller
- Created very preliminary bookstructure from available Gaudí articles and intermezzo's, no changelog yet