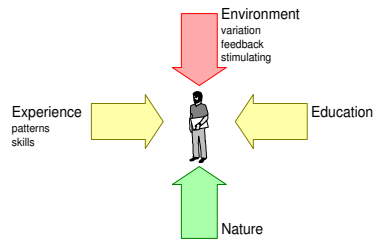


# Decomposing the Architect; What are Critical Success Factors?



**Gerrit Muller**

Embedded Systems Institute

Den Dolech 2 (Laplace Building 0.10) P.O. Box 513, 5600 MB Eindhoven The Netherlands

`gerrit.muller@embeddedsystems.nl`

## Abstract

System architects are scarce. If we want to search or educate potential system architects, then it is useful to know factors that determine the success of system architects. In this presentation we look at 4 areas: nature, education, environment and experience. We will make these areas more specific by quantification and illustration.

### **Distribution**

This article or presentation is written as part of the Gaudí project. The Gaudí project philosophy is to improve by obtaining frequent feedback. Frequent feedback is pursued by an open creation process. This document is published as intermediate or nearly mature version to get feedback. Further distribution is allowed as long as the document remains complete and unchanged.

All Gaudí documents are available at:  
<http://www.gaudisite.nl/>

version: 1.2

status: draft

July 1, 2011

# 1 Introduction

One of the big challenges of today is: How do we get more, and effective, system architects? At Philips and the Embedded Systems Institute we have been very successful in teaching the non-technical aspects of systems architecting to people. This course, called SARCH, has been given 36 times (May 2006) to about 570 participants. We also provide the Embedded Systems Architecting course (ESA), that has been given more than 20 times to more than 300 participants, which addresses the technical broadening of designers. We identified a number of missing steps in between these courses: addressing multi-disciplinary design. We fill this hole by "single aspect" courses that address one aspect at the time, for instance, performance or reliability. The performance oriented course, that has been given 7 times to about 100 people, is also successful. The next course that we developed to fill this hole is the Multi-Objective System Architecting and Design (MOSAD) course. The evaluation after 3 courses revealed a problem: the participants are satisfied, but the teacher is not satisfied. The dissatisfaction of the teacher is that the participants pick up many submethods and techniques provided in the course, but they struggle to integrate this into an architecting approach.

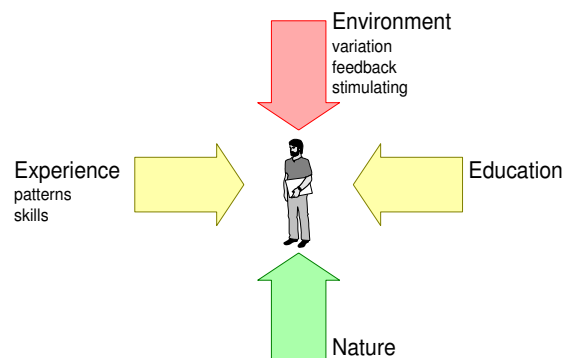


Figure 1: Decomposing Contributing Factors

This conclusion triggered the analysis of critical success factors for system architects. We decomposed these factors into four categories: *education*, *experience*, *environment*, and *nature*, as shown in Figure 1. We will discuss these four categories in separate sections. We will start with a section about the architect, to create a baseline for the further analysis.

## 2 What is an Architect?

System architects need a wide range of knowledge, skills and experience to be effective. Figure 2 shows a typical development of a system architect.

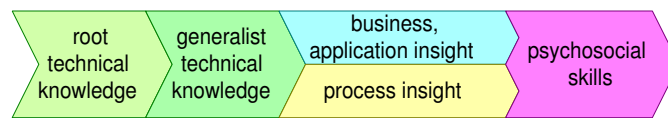


Figure 2: Typical Development of a System Architect

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.

When the awakening system architect has reached a technological breadth, it will become obvious that most problems have a root cause outside of technology. Two main parallel streams are opened:

- The business side: the market, customers, value, competition, logistics, service aspects
- The process side: who is doing what and why

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.

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.

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 his roots in in detailed technical knowledge. This root is important for the generalist himself, since it provides him with an anchor and a frame of reference. It is vital in the communication with other specialists, because it gives the generalist credibility.

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 which benefit from a generalist profile. For instance the function 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

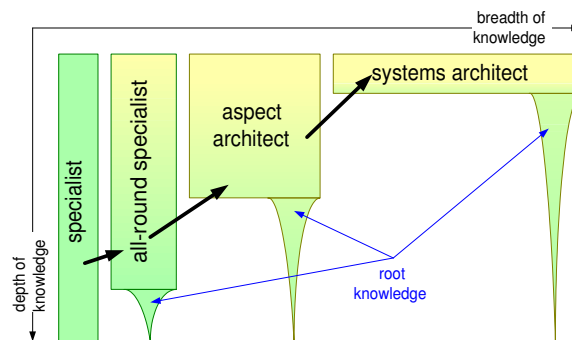


Figure 3: Growth in technical breadth, intermediate functions from specialist to system architect

a specialist to a generalist is of course not a binary transition. Figure 3 shows a more gradual spectrum from specialist to system architect. The arrows show that intermediate functions exist in larger product developments, which are natural stepping stones for the awakening architect.

Examples of aspect architects are:

- subsystem architects
- SW, mechanics or electronics architects

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, however the limited scope reduces the required breadth to a hopefully realistic level.

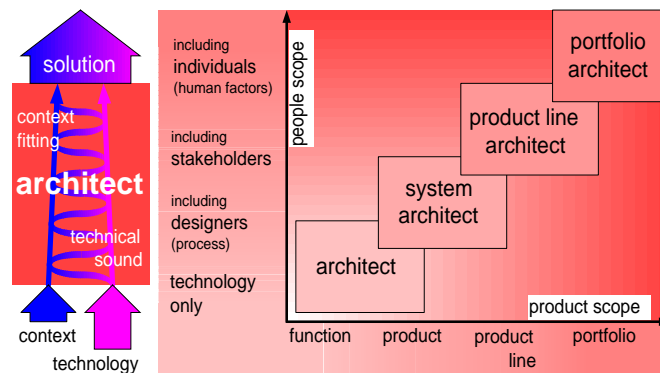


Figure 4: Different Architecting Scopes

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, that is 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.

Figure 4 shows that the scope of architects widely varies. The common denominator for all these architects is the bridge function between context and technology (or problem and solution). An architect needs sufficient know-how to understand the context as well as the technology, in order to design a solution, which fits in the context and is technical sound at the same time.

In general increasing the product scope of an architect coincides with an increase in people scope at the same time.

### 3 Education

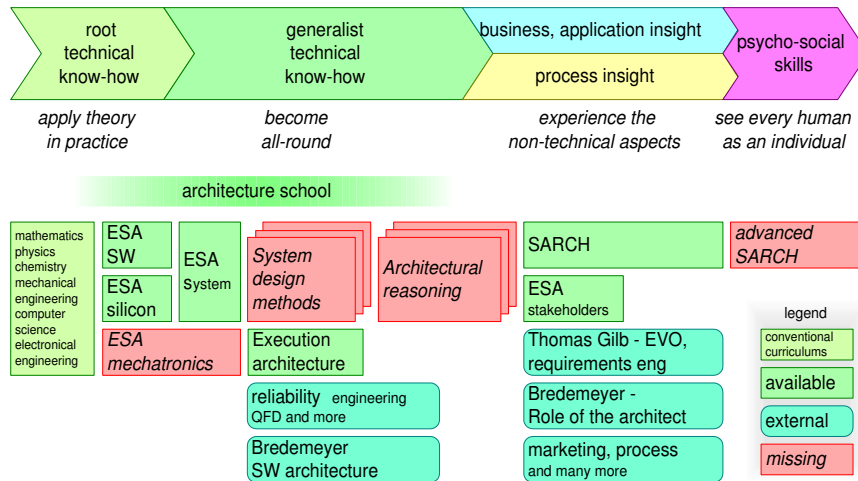


Figure 5: Proposed Curriculum for System Architects

A curriculum proposal for architects is shown in Figure 5. At the top of the figure the growth path of a system architect is shown. Below the courses or course subjects are shown which fit in the architect career path. Note that this is not a unified list for all architects. Instead it is a palette of courses, where the architect must select the courses which best fit his current needs. In color coding is indicated if courses are available internal or external.

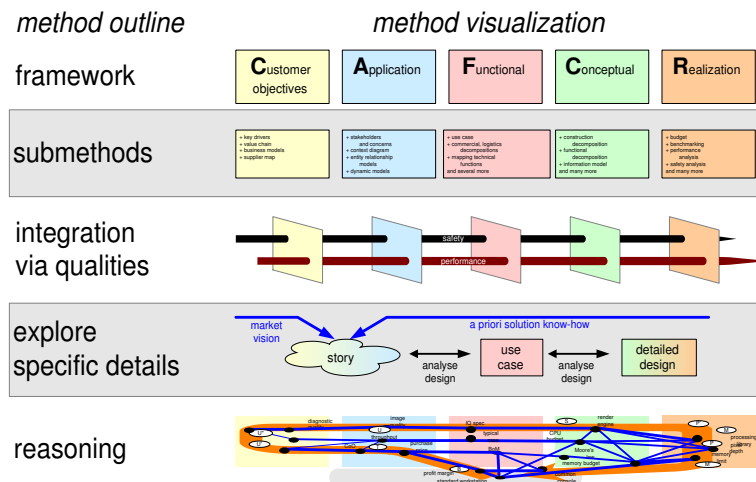


Figure 6: The outline of a CAFCR based architecting method

Figure 6 shows the overall outline of an architecting method, as it is being used in the MOSAD or *Architectural Reasoning* course. The right hand side shows the visualization of the steps of the method. The *framework* is a decomposition into five views, the “CAF<sub>CR</sub>” model, *Customer Objectives, Application, Functional, Conceptual, and Realization* views.

Per view in the decomposition a collection of *submethods* is given. The collections of submethods are open-ended. The collection is filled by borrowing relevant methods from many disciplines.

A decomposition in itself is not useful without the complementing integration. *Qualities* are used as *integrating* elements. The decomposition into qualities is orthogonal to the “CAF<sub>CR</sub>” model.

The decomposition into CAF<sub>CR</sub> views and into qualities both tend to be rather *abstract, high level* or *generic*. Therefore, a complementary approach is added to *explore specific details*: story telling. Story telling is the starting point for specific case analysis and design studies.

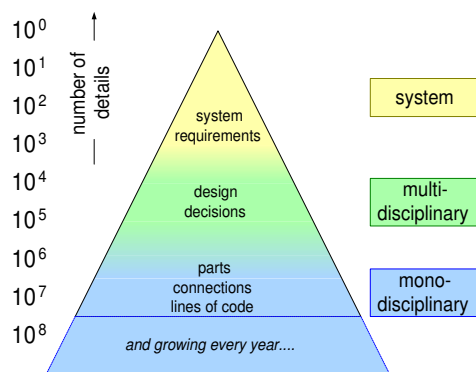


Figure 7: Connecting System Design to Detailed Design

These approaches are combined into a thread of *reasoning*: valuable insights in the different views in relation to each other. The basic working methods of the architect and the decompositions should help the architect to maintain the overview and to prevent drowning in the tremendous amount of data and relationships. The stories and detailed case and design studies should help to keep the insights factual.

The translation of system requirements 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 7 shows this dynamic range as a pyramid with the system at the top and the millions of technical details at the bottom.

The combination of Figures 6 and 7 brings us to a very common organizational

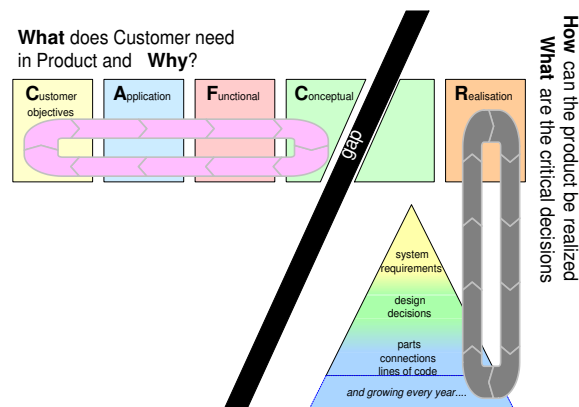


Figure 8: Organizational Problem: Disconnect

problem: the disconnect between customer oriented reasoning (breadth, CAFCR) and technical expertise (depth, the mono-disciplinary area in the pyramid). Figure 8 shows this disconnect.

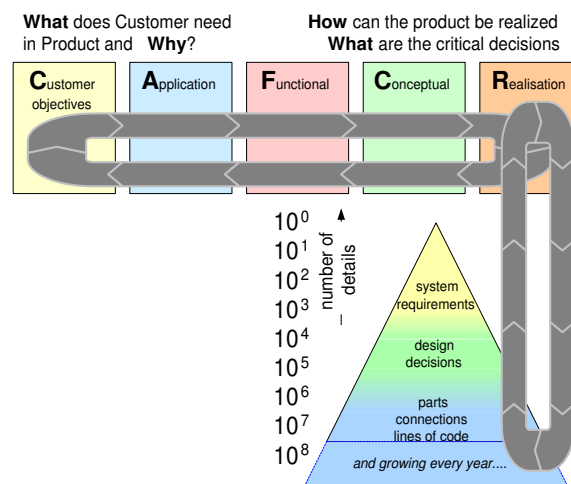


Figure 9: Architect: Connecting Problem and Technical Solution

Our definition of the work of an architect places this role as a bridge between these two worlds, as shown in Figure 9. In essence the architect must combine and balance breadth and depth iterations.

We should realize that this architect role is quite a stretching proposition. The architect is stretched in customer, application and business direction and at the same time the same architect is expected to be able to discuss technological details at nuts and bolts level. By necessity the architect will function most of the time at higher abstraction levels, time does and brain capacity don't allow the architect to

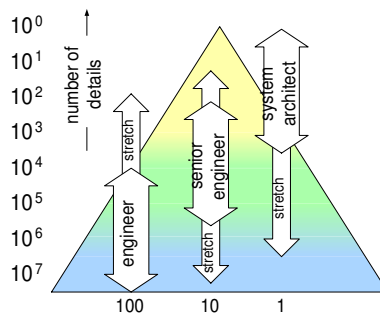


Figure 10: Major Bottleneck: Mental Dynamic Range

spend all time at detailed design level. Figure 10 shows that different people fill different spots in the abstraction hierarchies. For communication purposes and to get a healthy system design the roles must have sufficient overlap. This means that all players need to be stretched regularly beyond their natural realm of comfort.

The MOSAD course provides means to address:

- the breadth of systems architecting
- the depth of technological design
- the connection of breadth and depth

If we look back at the first editions of the MOSAD course, then we see that participants have the tendency to either go for breadth or for depth. But exploring both breadth and depth, and even more challenging connecting breadth and depth appears to be very difficult.

## 4 Nature

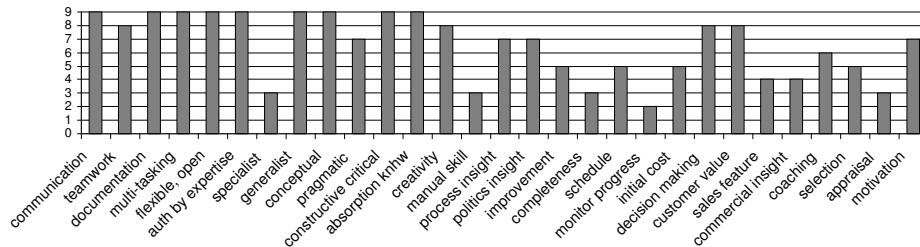


Figure 11: Profile of an "Ideal" System Architect

The profile of the "ideal" system architect shows a broad spectrum of required skills, as shown in Figure 11. A more complete description of this profile and the skills in this profile can be found at[3]. 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, which is based on technical leadership, where the architect provides direction (*know-how* and *reasoning power*) as well as moderates the integration (*interpersonal skills*).

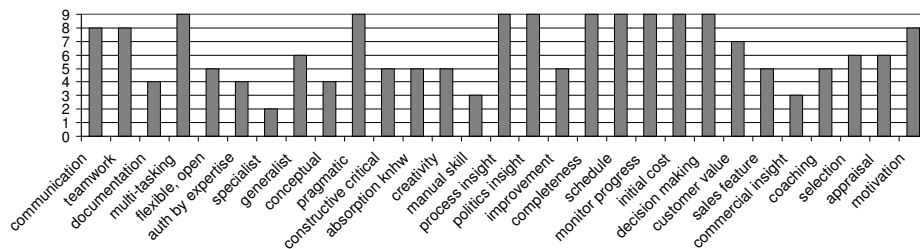


Figure 12: For Comparison: Profile of a Project Leader

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 allows for the formation of architecture teams, which as a team are close to this profile.

For comparison the profile of a project leader is shown in Figure 12. A project leader is totally focused on the result. This requires project management skills, which is the core discipline for project leaders. The multi-tasking ability is an important prerequisite for the project leader. If this ability is missing the person runs a severe risk on a burn out.

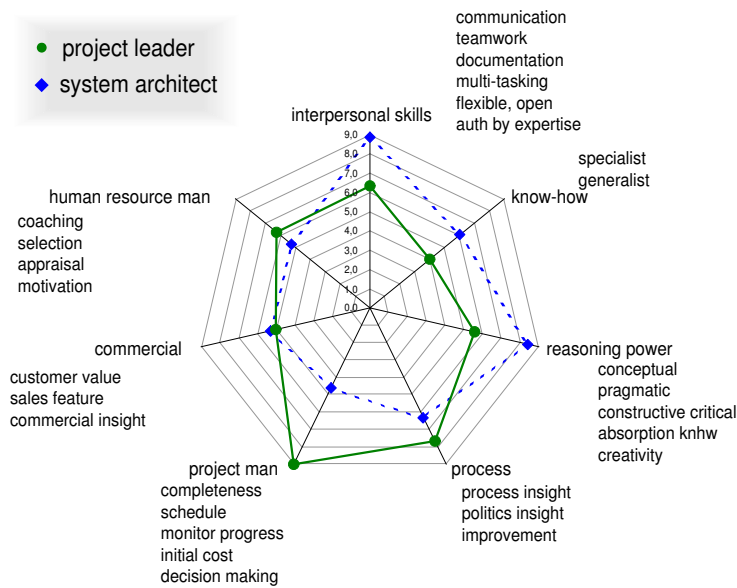


Figure 13: Project Leader versus System Architect

The comparison is further visualized in Figure 13, where the more detailed skills from Figures 11 and 12 are grouped together.

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

Figure 14: Most Discriminating Characteristics

In practice the characteristics shown in Figure 14 are quite discriminating when selecting (potential) system architects: The first reduction step, when searching for architects, is to select the generalists only. This step reduces the input stream with one order of magnitude. The next step is to detect those people which need time and concentration to make progress. These people become unnerved in the job of the system 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 these people become

system architect nevertheless they are in sever danger of stress and burn out, hence it is also the benefit of the person itself to fairly asses the multi-tasking characteristic.

The attitude of the (potential) architect is important for the long term effectiveness. Roughly two attitudes can be distinguished: architects that ask for formal power and architects that operate on the basis of build-up authority. Building up authority requires know-how and visible contribution to projects. We have observed that architects asking for formal power 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 costs a lot of energy, but it pays back in the long term.

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.

## 5 Experience

The effectiveness of an architect depends on experience. In all years of being an engineer, designer and architect, a lot of different needs in different contexts with different solutions with different complicating challenges pass by. If all these events are processed by the (potential) architect, then a frame of reference is created that is very valuable for future architecting work.

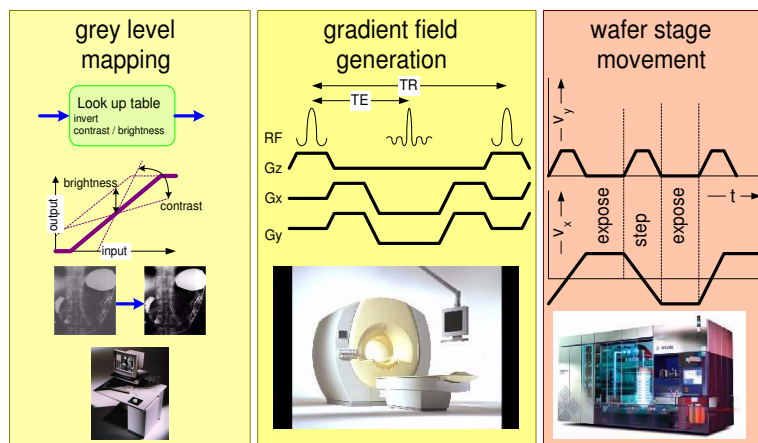


Figure 15: Example: Trapezoid Pattern

In this section we will illustrate the experience factor by means of a few architecture patterns that repeatedly popped up in completely different domains. For this purpose we look at the *Trapezoid Pattern*, as shown in Figure 15. One of the very common technical problems is the actuation by software of some physical entity, for instance for positioning, moving or displaying. In these cases the software often has to create set-points for one parameter, where this parameter is constant at different levels for some time and switches linearly from one level to another level. For instance, a sample table is at a given position (constant), moves with a constant velocity to the next position, and then stays at the next position for some time (constant). This same behavior is also present in the actuation of gradient fields in MRI scanners, and in the grey level mapping in imaging displays (although the last example uses grey levels as running parameters instead of time).

In the system a chain of transformations takes place to get from a high level software representation towards an actual physical behavior by physical objects. Figure 16 shows such a chain of three steps: *computation*, *conversion*, and *actuation*. Note that this chain is often more complex in real systems with more software steps (controller algorithms, corrections), more electronic steps (controllers, amplifiers), and more mechanical steps (motors, transmission). The high level software representation that is the starting point is unconstrained (high precision in time as well as

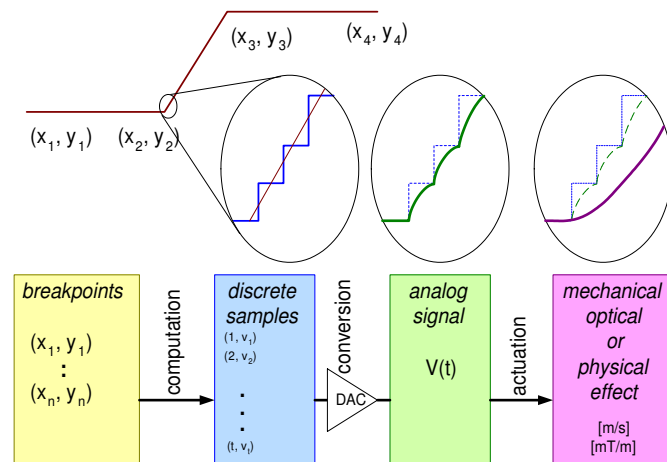


Figure 16: From SW input to physical Effect

in value). The most common representation is break-point based: the coordinates, where the running parameter changes the linear behavior, are specified.

The conversion and actuation steps have their own particular transfer functions. These steps may introduce additional delays, noise, variations et cetera. The virtual model in the high level software does not take this into account or makes (calibrated) assumptions.

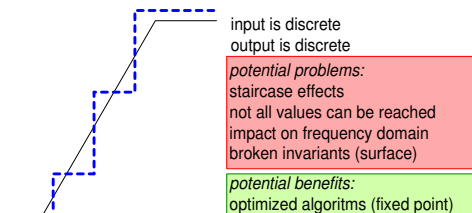


Figure 17: Discretization effects

The *computation* step transforms the unconstrained representation into a constrained sampled list of values. This transformation is a discretization in two directions: time and value, see Figure 17. This discretization may introduce system level problems:

**Staircase effects** the linear shape is approximated by many staircase-like steps.

The question is how this software output is transformed into the actual physical actuation and if artifacts will be observable in the physical performance.

**Not all values can be reached** . Normally the digital to analog conversion is a bottleneck in the values that can be reached. This conversion can be very

much limited in low cost solutions (8-bits, 256 values) to limited (16-bit, 65536 values). The time-values are also limited, varying from sub-microsecond for more expensive solutions to milliseconds for simple low-cost controls. The consequence of this limitation is that the physical reality may differ in a systematic way from the virtual model in the high level software. For example the high level software may have determined that at moment  $t = 3.14159$  the system should be at position  $x = 2.718281$ , while actually the system is controlled to stop at  $t = 3.1$ ,  $x = 2.7$ .

**Impact on frequency domain** The staircase approximation of linear behavior introduces many higher frequencies in the frequency domain. Many of the higher frequency artifacts are filtered out in the analog and physical part of the chain. However, due to aliasing-like problems the system performance might degrade in unexpected ways.

**Broken invariants (surface)** The high level software model in many systems is based on invariants. For instance, if we control velocity linear, then we expect that we know the position as the integral of velocity. Discretization, at lower software level, will violate the higher level assumption. If the model assumes we move with  $v = 3.14159m/s$ , while we actually move with  $v = 3.1m/s$ , then the position will deviate significant. Interestingly, the low level software can compensate for this error by modulating the value: 58% of the time  $v = 3.1m/s$  and 42% of the time  $v = 3.2m/s$ . These solutions work, but introduce again their own next level of problems and artifacts. In this example the frequency of the modulation may introduce unexpected physical behavior, such as vibrations.

A priori use of the need for discretization can also turn into a benefit. Especially the consequent use of integer representations (with some pragmatic normalization, such as  $255 = 5Volt$ ) reduces processor load, memory use and may increase system performance.

Discretization problems, the artifacts introduced by discretization, the measures against artifacts are also universally applicable. However, the exact consequence and the right countermeasure are domain dependent.

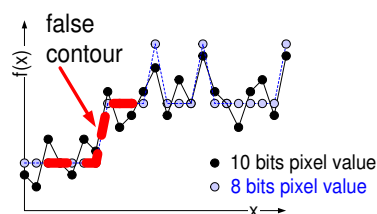


Figure 18: Example of Discretization Problem

As example of discretization problems Figure 18 shows a typical image quality problem that popped up during the integration phase of a Medical Imaging Workstation. The pixel value  $x$ , corresponding to the amount of X-ray dose received in the detector, has to be transformed into a grey value  $f(x)$  that is used to display the image on the screen. Due to discretization of the pixel values to 8 bits *false contours* become visible. For the human eye an artefact is visible between pixels that are mapped on a single grey value and neighboring pixels that are mapped on the next higher grey value. It is the levelling effect caused by the discretization that becomes visible as false contour. This artefact is invisible if the natural noise is still present. Concatenation of multiple processing steps can strongly increase this type of artifacts.

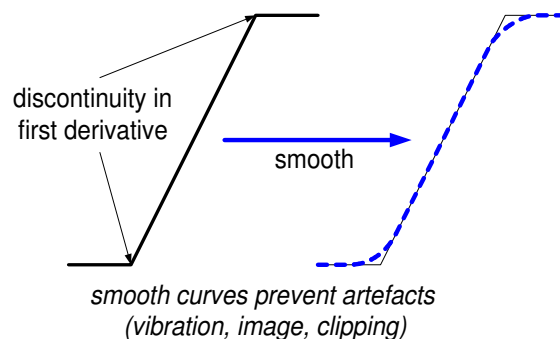


Figure 19: Example of Generic Smoothing Consideration

An example of a pattern that builds further on this transformation chain is shown in Figure 19. Physical systems in general start to show artifacts with discontinuous inputs. The linear approximation used in the trapezoid pattern has a discontinuity in the derivative. For example, if we control velocity, then the acceleration jumps at the break-point. A solution for this discontinuity is to *smooth* the input function, for instance by a low-pass filter. Note that most analog and mechanical systems are already natural low-pass filters. Despite the low-pass characteristic of the later part of the chain artifacts might still be induced by the discontinuity. These remaining artifacts can be further removed by using an explicit low-pass filter in the high level software model. Again this is an example of a pattern that is universally applied in multiple domains.

The example showed a small subset of patterns that an architect experiences. This subset as it has been discussed here is highly technical. However, in real life technical patterns and organizational patterns are experienced concurrently. For example in the trapezoid example also a number of organizational patterns pop up, related to mono-disciplinary experts and multi-disciplinary design, and system integration.

In Figure 20 the career of an architect is shown with the repeated encounters

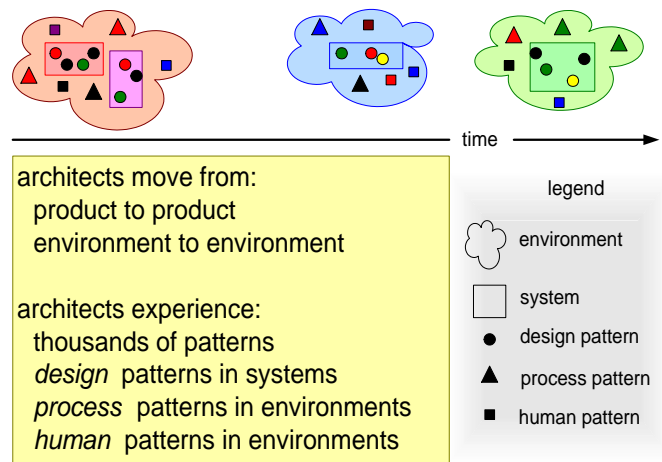


Figure 20: Architects Collect a Rich Set of Patterns

of patterns in different products and in different environments. We estimate that an experiences architect encounters (and files and uses) thousands of patterns. All these patterns form a frame of reference for the architect as an individual. This frame of reference helps the architect to assess new architectures very quickly. Potential problem areas are identified and design issues are weighted very fast, thanks to this frame of reference.

## 6 Environment

The business process for an organization which creates and builds systems consisting of hardware and software can be decomposed in 4 main processes as shown in figure 21. This process decomposition model is more extensively discussed in[2].

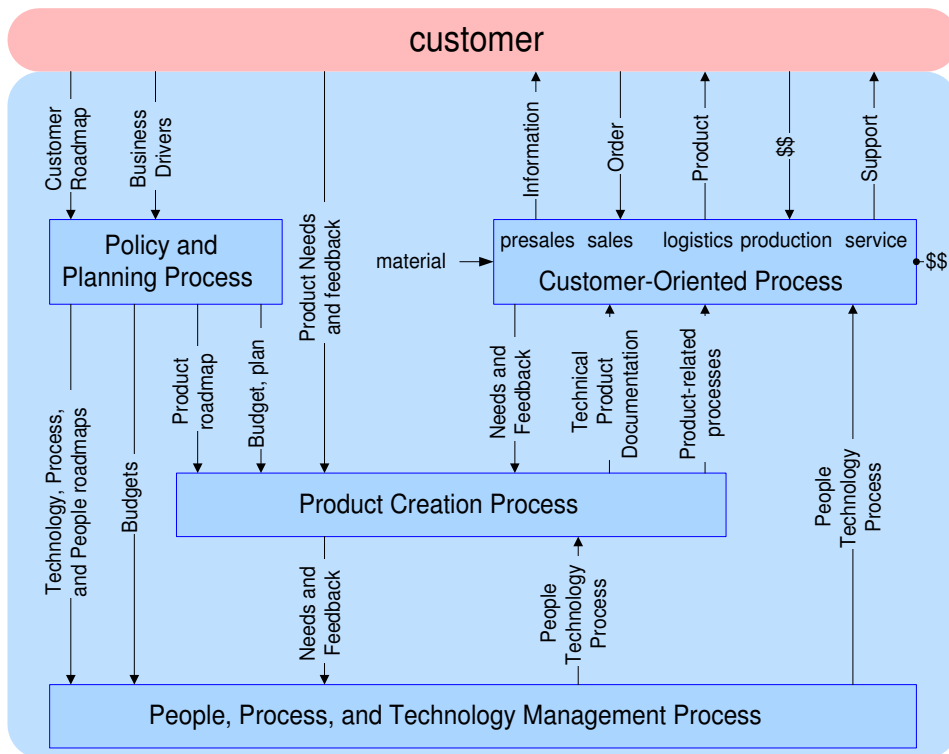


Figure 21: Simplified decomposition of the Business

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 cash flow 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 cash flow 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.

**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.

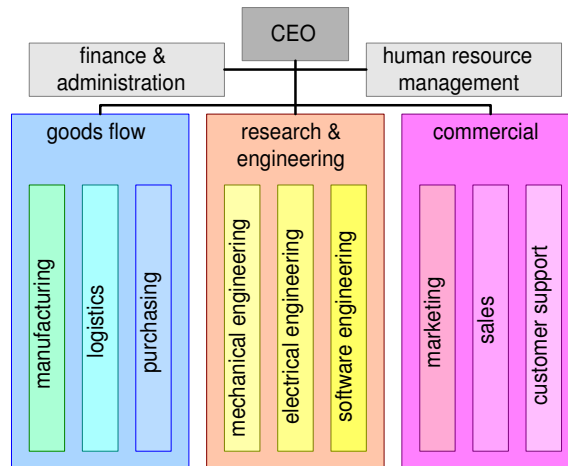


Figure 22: Line Organization Stovepipe

The challenge for companies is to organize themselves in a way that support these 4 different types of processes. Rather common is that the People and Technology Management Process is mapped on the line organization, see Figure 22. This figure also shows a common problem of hierarchical organization structures: the organizational entities become (over)specialized stovepipes.

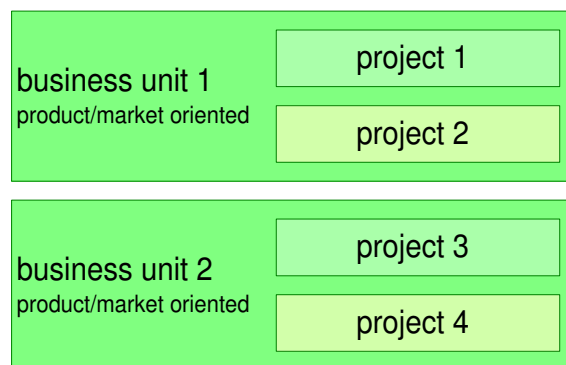


Figure 23: Business Organization Stovepipe

The *Product Creation Process* maps often on a business oriented project organi-

zation, as shown in Figure 23. The stovepipe problem is here also present, although the stovepipes are now in the product/market direction.

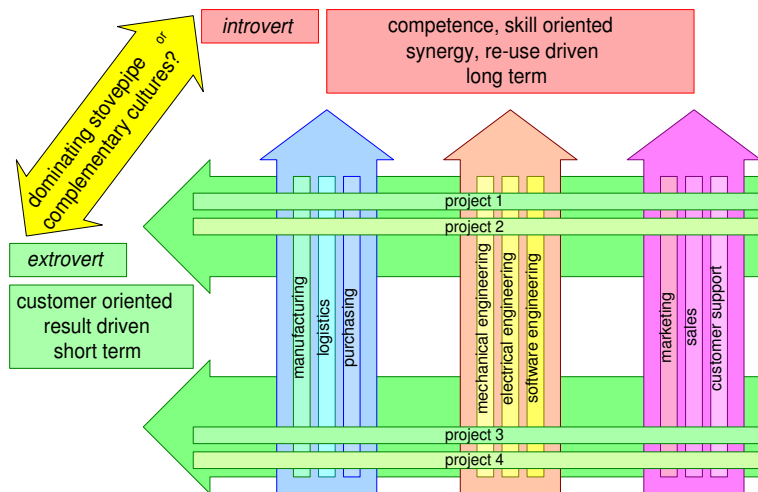


Figure 24: Different Concerns

The combination of both organization models results in a matrix organization, where the two types of organizations have different concerns. The line organization is competence and skill oriented, looking for synergy and re-use opportunities. The line organization typically has a long term focus, but an introvert perspective. The business organization is customer oriented and result driven. The business organization typically has a short term focus, but an extrovert perspective.

Figure 25 positions the *System Architecture Process* in the simplified process decomposition. The System Architecture Process bridges the Policy and Planning Process and the Product Creation Process. The roadmaps made in the policy and planning process are the shared understanding of direction of the company:

- It positions the products in the market and within the product portfolio.
- It shows the relations between products, such as re-use of technology.
- It positions the product in the technology life-cycle.
- It relates products and technology to the (long lead) development of people and process

The System Architecture Process is the process that:

- Gathers input for the Policy and Planning Process
- Brings in technical overview and common sense in the Policy and Planning Process and the Product Creation Process

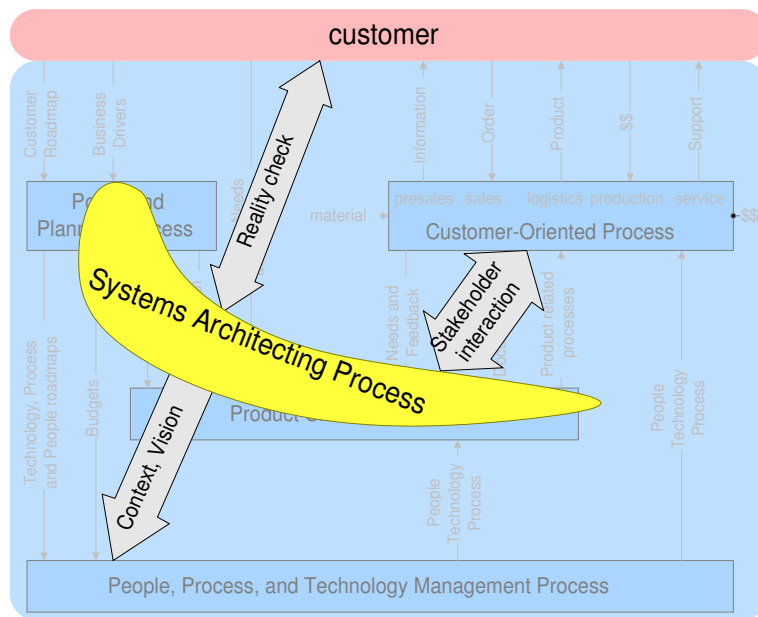


Figure 25: Positioning System Architecting

- Transfers the intention of the Policy and Planning Process into the Product Creation Process
- Performs the system level Requirement analysis, Specification, Design and Verification
- Maintains the consistency, integrity and balance.

systems engineering as discipline  
 job rotation  
 stimulate architect exposure  
 stretch all engineers  
 cultivate customer & market oriented culture  
 share and invest in future exploration and vision

Figure 26: What Can We Do to Improve the Environment?

Until now we have sketched the organizational and process environment in which the system architect operates. A complex environment that is full of human factors, such as conflicting interests and complementing (or opposing?) characters.

The natural growth direction in this environment is specialization. In some organizations the security or standardization efforts hurt the architecting effectiveness. For example, we have seen organizations where customer key drivers, cost of ownership models, and market roadmaps are marketing confidential. The gap as described in Figure 8 is here imposed by the organization.

Figure 26 shows what we can do to improve the environment from system architecting perspective.

**Systems engineering as discipline** Conventional disciplines are technology oriented, for instance: mechanical, electrical, and software engineering. However, systems engineering has grown into a discipline itself. Most organizations have a severe lack of systems engineers and systems architects. Organizational ownership for systems engineering as a discipline counter-balances the natural tendency towards specialization.

**Job rotation** is one of the means to broaden employees. The cultivation of a systems attitude requires such a broadening, it is a prerequisite to become systems engineer

**Stimulate architect exposure** to help them overcome their introvert nature and to help them bridge the gap between managers and architects.

**stretch all engineers** The broadening mentioned before should not be limited to (potential) system architects. The extremely challenging job of a system architect becomes somewhat more feasible if the engineers are at least system-aware.

**cultivate customer and market oriented culture** Especially in large organizations the distance from local organizational concerns to customers and market can become large. System architects suffer tremendously from introvert organizations, because the architect has to connect the customer and market needs to technological decisions.

**share and invest in future exploration and vision** Good architects base their work on a vision. Some investment is needed to create a vision and to keep the vision up-to-date. A vision becomes much more powerful if it is shared throughout the organization.

## 7 Discussion and Conclusions

This paper was triggered by the not yet satisfactory results of our newly developed MOSAD course. Analysis of the critical success factors for system architects provides us with the following insights:

- Only a limited set of technical educated people have a personality profile (the *nature* component) that fits with the architecting role.
- System architecting education for people that do **not** fit in this architect profile is, nevertheless, a good investment. System aware designers ease the job of the system architect.
- Environmental issues, such as organization and processes, have a big impact on the effectiveness of architects.
- Architects need to be stimulated and supported to break through roadblocks imposed by the environment.
- To integrate and use multi-disciplinary design techniques a broad frame of reference is needed. Such a frame of reference helps to position, relate and weight issues, and to identify risks. Without the ability to quickly determine value, relevance and criticality, designers drown in the practical infinite space of problems and solutions.
- A frame of reference grows over time and is the result of experience. This process can be supported by explicit reflection, for instance triggered by a mentor or intervention by peers.

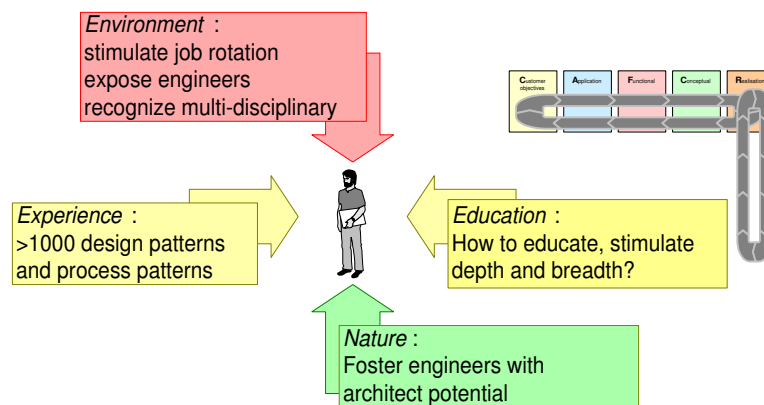


Figure 27: Conclusion

Figure 27 summarizes the conclusions:

**Education** How do we stimulate and educate breadth and depth synthesis?

**Nature** People with architecting genes are scarce; We have to foster and stimulate those people that fit in the architecting profile.

**Experience** plays a very critical role in cultivating architects. Good architects have a very rich frame of reference with thousands of patterns.

**Environment** has a big impact on architect effectiveness. Stimulation of job rotation helps to enrich the frame of reference. By exposing engineers to multi-disciplinary aspects the awareness for system issues increases The environment (management, rewarding system) must recognize the value of multi-disciplinary design.

## 8 Acknowledgements

Louis Stroucken detected a painful copy-paste error and provided other useful feedback.

## References

- [1] Gerrit Muller. The system architecture homepage. <http://www.gaudisite.nl/index.html>, 1999.
- [2] Gerrit Muller. Process decomposition of a business. <http://www.gaudisite.nl/ProcessDecompositionOfBusinessPaper.pdf>, 2000.
- [3] Gerrit Muller. Function profiles; the sheep with 7 legs. <http://www.gaudisite.nl/FunctionProfilesPaper.pdf>, 2001.

## History

**Version: 1.2, date: September 11, 2006 changed by: Gerrit Muller**

- exchanged Figures 11 and 12
- extended the text about power versus authority

**Version: 1.1, date: June 12, 2006 changed by: Gerrit Muller**

- added kind of conclusion to the education section

**Version: 1.0, date: June 7, 2006 changed by: Gerrit Muller**

- created a text version
- changed status to draft

**Version: 0, date: January 31, 2006 changed by: Gerrit Muller**

- Created, no changelog yet