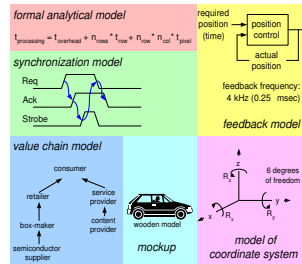


Basic Working Methods of a System Architect

-



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

The challenge for the architect is to cover a wide range of subjects, with many unknowns and uncertainties, while decisions are required all the time. The basic working methods, such as viewpoint hopping, modelling, handling uncertainties and WWHWWW questions are described.

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: 1.5

status: concept

July 1, 2011

1 Introduction

The basic working methods of the architects are covered by a limited set of very generic patterns:

- Viewpoint hopping, looking at the problem and (potential) solutions from many points of view, see section 2.
- Decomposition, breaking up a large problem in smaller problems, introducing interfaces and the need for integration, see section 3.
- Quantification, building up understanding by quantification, from order of magnitude numbers to specifications with acceptable confidence level, see section 4.
- Decision making when lots of data is missing, see section 5.
- Modelling, as means of communication, documentation, analysis, simulation, decision making and verification, see section 6.
- Asking Why, What, How, Who, When, Where questions, see section 7.
- Problem solving approach, see section 8.

Besides these methods the architect needs lots of “soft” skills, to be effective with the large amount of different people involved in creating the system. See [5], [2] and [3] for additional descriptions of the work and skills of the architect.

2 Viewpoint hopping

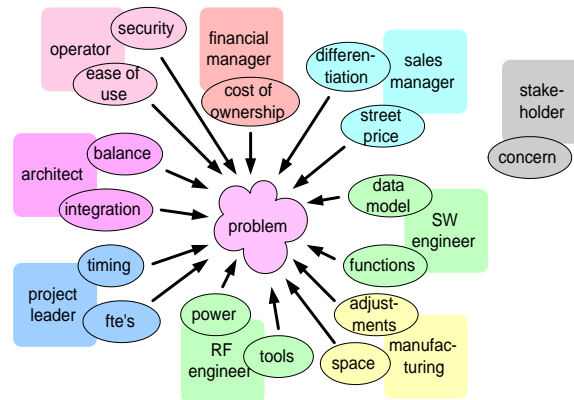


Figure 1: Small subset of viewpoints

The architect is looking towards problems and (potential) solutions from many different viewpoints. A small subset of viewpoints is visualized in figure 1, where the viewpoints are shown as stakeholders with their concerns.

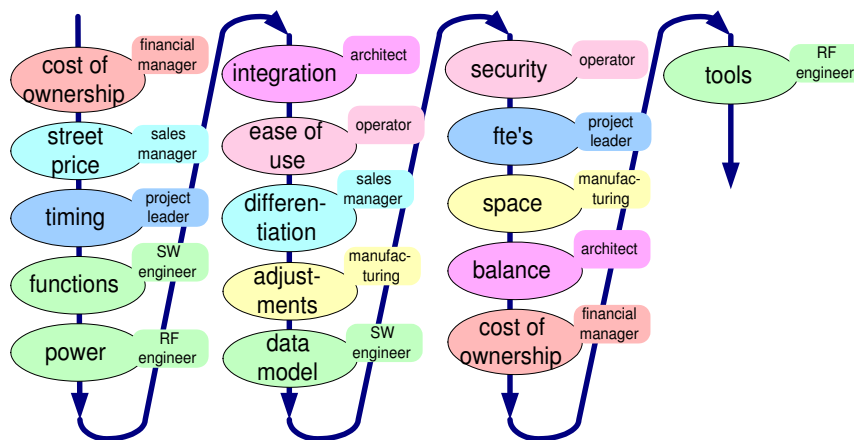


Figure 2: Viewpoint Hopping

The architect is interested in an overall view on the problem, where all these viewpoints are present simultaneously. The limitations of the human brains force the architect to create an overall view by quickly alternating the individual viewpoints. The order in which the viewpoints are alternated is chaotic: problems or opportunities in one viewpoint trigger the switch to a related viewpoint. Figure 2 shows a very short example of viewpoint hopping. This example sequence can take

anywhere from minutes to weeks. In a complete product creation project the architect makes thousands¹ of these viewpoint changes.

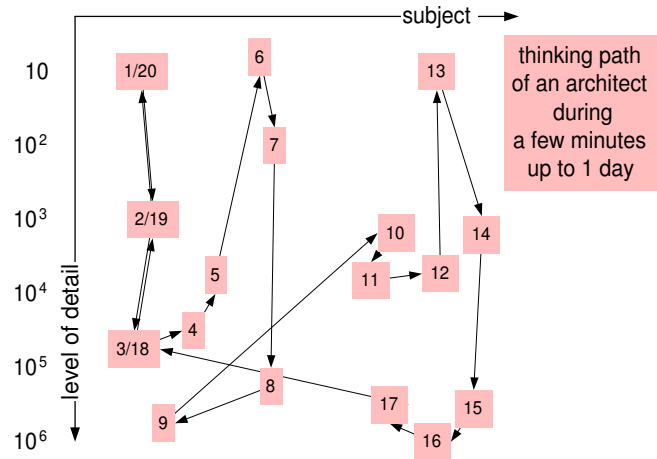


Figure 3: The seemingly random exploration path

Viewpoint hopping is happening quite fast in the head of the architect. Besides changing the viewpoint the architect is also zooming in and out with respect to the level of detail. The dynamic range of the details taken into account is many orders of magnitude. Exploring different subjects and different levels of detail together can be viewed as an exploration path. The exploration path followed by the architect (in the architect's head) appears to be quite random. Figure 3 shows an example of an exploration path happening inside the architects head.

The plane used to show the exploration path has one axis with *subjects*, which can be stakeholders, concerns, functions, qualities, design aspects, et cetera, while the other axis is *the level of detail*. A very coarse (low level of detail) is for example the customer key driver level (for instance cost per placement is 0.1 milli-cent/placement). Examples at the very detailed level are lines of code, cycle accurate simulation data, or bolt type, material and size.

Both axis span a tremendous dynamic range, creating a huge space for exploration. Systematic scanning of this space is way too slow. An architect is using two techniques to scan this space, that are quite difficult to combine: open perceptive scanning and scanning while structuring and judging. The open perceptive mode is needed to build understanding and insight. Early structuring and judging is dangerous because it might become a self-fulfilling prophecy. The structuring and judging is required to reach a result in a limited amount of time and effort. See figure 4 for these 2 modes of scanning.

The scanning approach taken by the architect can be compared with *simulated*

¹Based on observations of other architects and own experience.

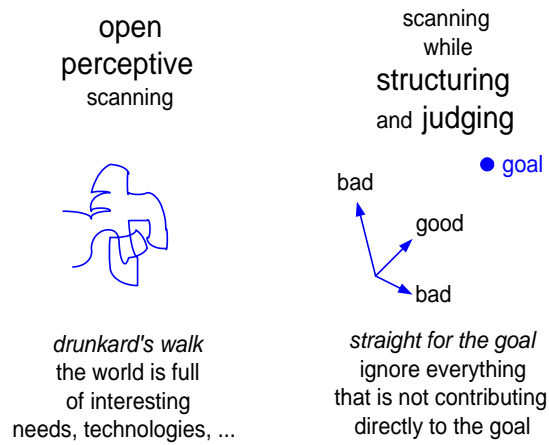


Figure 4: Two modes of scanning by an architect

annealing methods for optimization[4]. An interesting quote from this book, comparing optimization methods:

Although the analogy is not perfect, there is a sense in which all of the minimization algorithms thus far in this chapter correspond to rapid cooling or quenching. In all cases, we have gone greedily for the quick, nearby solution: From the starting point, go immediately downhill as far as you can go. This, as often remarked above, leads to a local, but not necessarily a global, minimum. Nature’s own minimization algorithm is based on a quite different procedure...

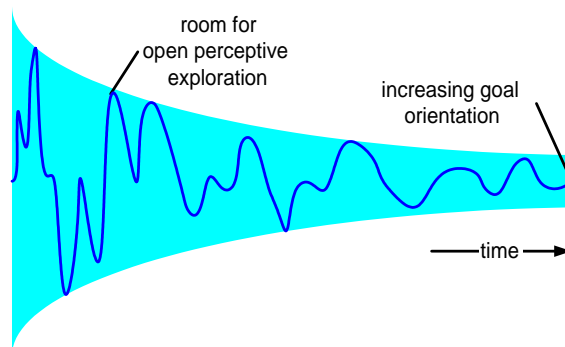


Figure 5: Combined open perceptive scanning and goal-oriented scanning

See also figure 5 for the combined scanning path. The perceptive mode is used more early in the project, while at the end of the project the goal oriented mode is dominant.

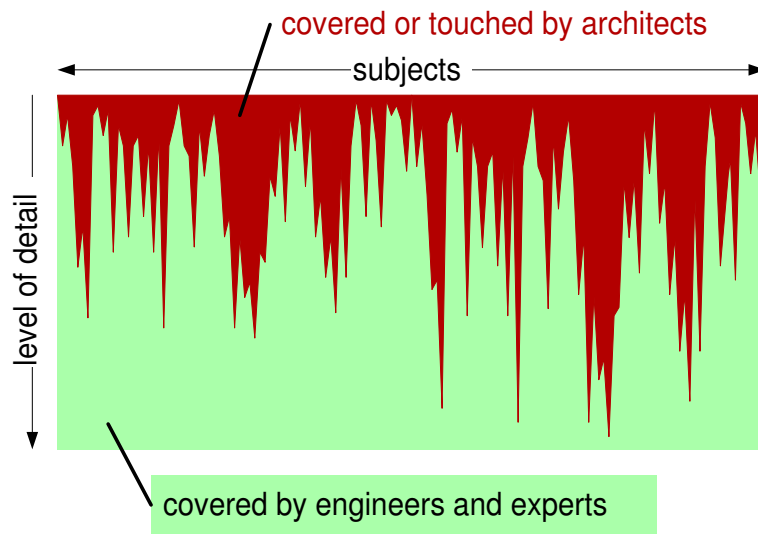


Figure 6: The final coverage of the problem and solution space by architect and engineers

The coverage of the problem and solution space is visualized in figure 6. Note that the area covered or touched by the architect(s) is not exclusively covered, engineers will also cover or touch that area partially. The architect needs experience to learn when to dig deeper and when to move on to next subjects. Balancing depth and breadth is still largely an art.

3 Decomposition and integration

The architect applies a reduction strategy by means of decomposition over and over, as shown in figure 7. Decomposition is a very generic principle. Decomposition can be applied for many different problem and solution dimensions, as will be shown in the later sections.

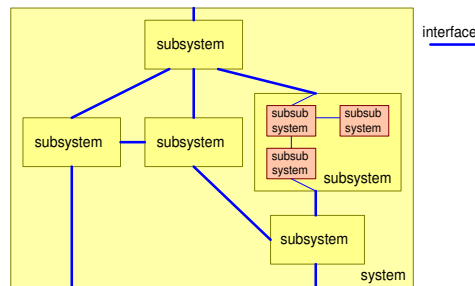


Figure 7: Decomposition, interface management and integration

Whenever something is decomposed the resulting components will be decoupled by interfaces. The architect will invest time in interfaces, since these provide a convenient method to determine system structure and behavior, while decoupling the inside of these components from their external behavior.

The true challenge for the architect is to design decompositions, that in the end will support an integration of components into a system. Most effort of the architect is concerned with the integrating concepts, how do multiple components work together?

Many stakeholders perceive the decomposition and the interface management as the most important contribution. The synthesis or integration part is more difficult and time consuming, and will be perceived as the main contribution by the architect.

4 Quantification

The architect is continuously trying to improve his understanding of problem and solution. This understanding is based on many different interacting insights, such as functionality, behavior, relationships et cetera. An important factor in understanding is the **quantification**. Quantification helps to get grip on the many vague aspects of problem and solution. Many aspects can be quantified, much more than most designers are willing to quantify.

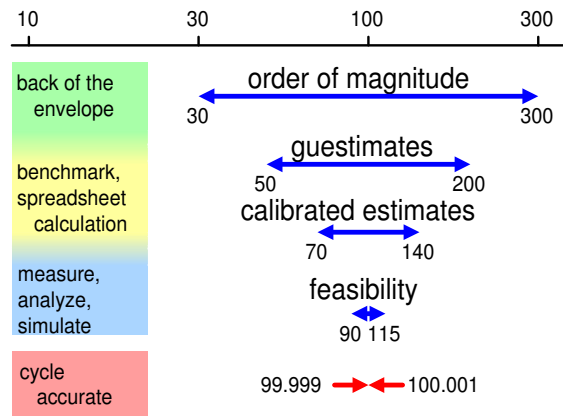


Figure 8: Successive quantification refined

The precision of the quantification increases during the project. Figure 8 shows the stepwise refinement of the quantification. In first instance it is important to get a feeling for the problem by quantifying orders of magnitude. For example:

- How large is the targeted customer population?
- What is the amount of money they are willing and able to spend?
- How many pictures/movies do they want to store?
- How much storage and bandwidth is needed?

The order of magnitude numbers can be refined by making back of the envelop calculations, making simple models and making assumptions and estimates. From this work it becomes clear where the major uncertainties are and what measurements or other data acquisitions will help to refine the numbers further.

At the bottom of figure 8 the other extreme of the spectrum of quantification is shown, in this example cycle accurate simulation of video frame processing results in very accurate numbers. It is a challenge for an architect to bridge these worlds.

Figure 9 shows an example how the quantification evolves in time. The dotted red line represents the required performance as defined in the specification. The

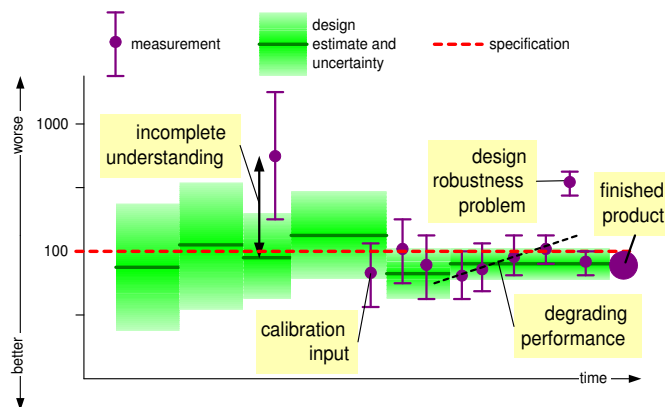


Figure 9: Example of the evolution of quantification in time

shaded area indicates the “paper” value, with its accuracy. The measurements are shown as dots with a range bar. A large difference between paper value and measurement is a clear indication of missing understanding. Later during the implementation continuous measurements monitor the expected outcome, in this example a clear degradation is visible. Large jumps in the measurements are an indication of a design which is not robust (small implementation changes cause large performance deviations).

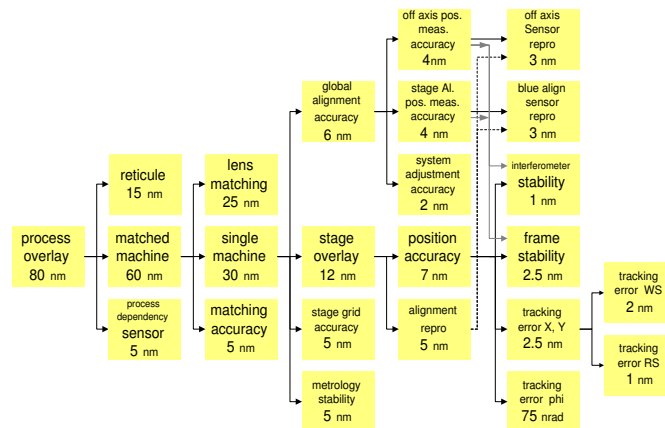


Figure 10: Example of a quantified understanding of overlay in a wafer stepper

Figure 10 shows a graphical example of an “overlay” budget for a wafer stepper. This figure is taken from the *System Design Specification* of the ASML TwinScan system, although for confidentiality reasons some minor modifications have been applied. This budget is based on a model of the overlay functionality in the wafer stepper. The budget is used to provide requirements for subsystems and compo-

nents. The actual contributions to the overlay are measured during the design and integration process, on functional models or prototypes. These measurements provide early feedback of the overlay design. If needed the budget or the design is changed on the basis of this feedback.

5 Coping with uncertainty

The architect has to make decisions all the time, while most substantiating data is still missing. On top of that some of the available data will be false, inconsistent or interpreted wrong.

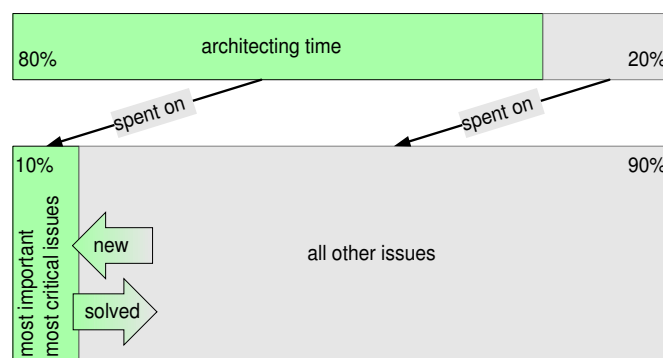


Figure 11: The architect focuses on important and critical issues, while monitoring the other issues

An important means in making decisions is building up insight, understanding and overview, by means of structuring the problems. The understanding is used to determine important (for the product use) and critical (with respect to technical design and implementation) issues. The architect will pay most attention to these *important* and *critical* issues. The other issues are monitored, because sometimes minor details turn out to be important or critical issues. Figure 11 visualizes the time distribution of the architect: 80% of the time is spent on 10% of the issues.

The architect will, often implicitly, work on the basis of a top 10 issue list, the ten most relevant (important, urgent, critical) issues. Figure 12 shows an example of such a “worry”-list.

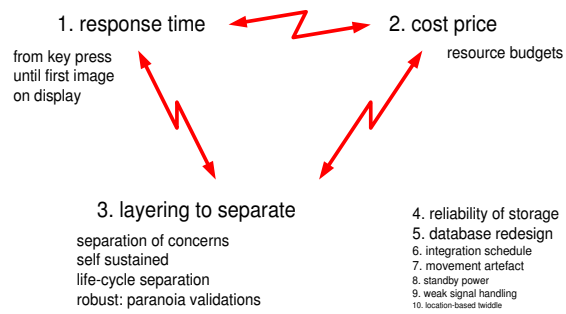


Figure 12: Example worry list of an architect

6 Modelling

Modelling is one of the most fundamental tools of an architect.

A **model** is
a **simplified** representation of
part of the **real world** used for:

communication, documentation
analysis, simulation,
decision making, verification

In summary models are used to obtain insight and understanding, facilitating communication, documentation, analysis, simulation, decision making, verification. At the same time the architect is always aware of the (over)simplification applied in every model. A model is very valuable, but every model has its limitations, imposed by the simplifications.

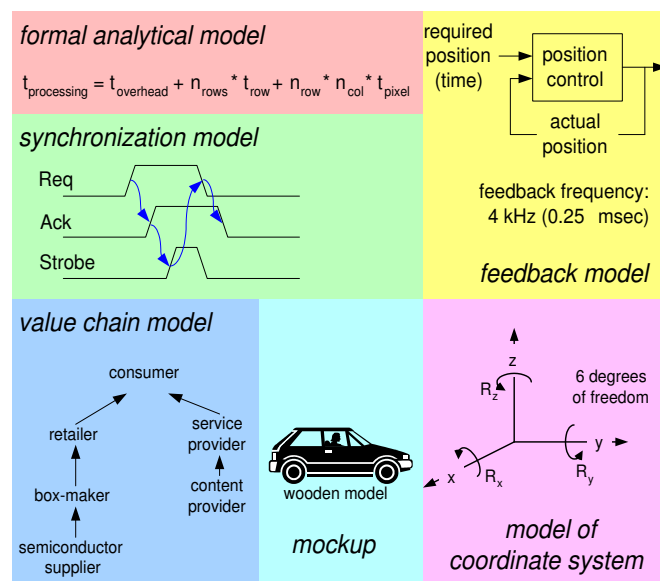


Figure 13: Some examples of models

Models exist in a very rich variety, an arbitrary small selection of models is shown in figure 13.

Models have many different manifestations. Figure 14 shows some of the different types of models, expressed in a number of adjectives.

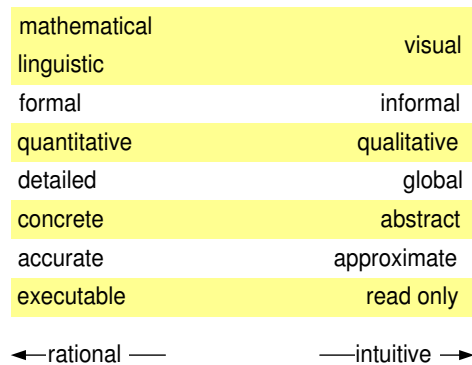


Figure 14: Types of models

Models can be *mathematical*, expressed in formulas, they can be *linguistic*, expressed in words or they can be *visual*, captured in diagrams. A model can be formal, where notations, operations and terms are precisely defined, or informal using plain English and sketches. Quantitative models use meaningful numbers, allowing verification and judgements. Qualitative models show relations and behavior, providing understanding. Concrete models use tangible objects and parameters, while abstract models express mental concepts. Some models can be executed (as a simulation), while other models only make sense for humans reading the model.

7 WWHWWW questions

Why	Who
What	When
How	Where

Figure 15: The starting words for questions by the architect

All “W” questions are an important tool for the architect. Figure 15 shows the useful starting words for questions to be asked by an architect.

Why, what and how are used over and over in architecting. Why, what and how are used to determine objectives, rationale and design. This works highly recursively, a design has objectives and a rationale and results in smaller designs that again have objectives and rationales. Figure 16 shows that the recursion with **why** questions broadens the scope, and recursion with **how** questions opens more details in a smaller scope.

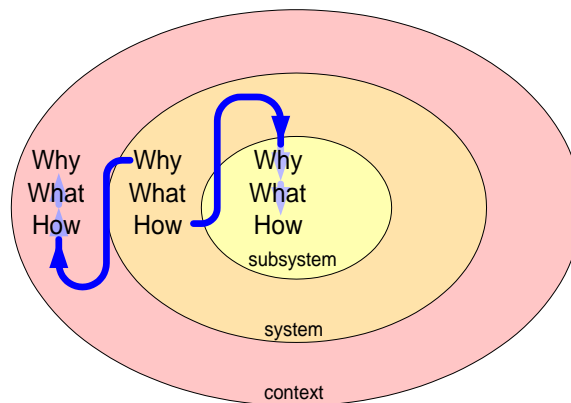


Figure 16: Why broadens scope, How opens details

Who, where and when are used somewhat less frequently. Who, where and when can be used to build up understanding of the context, and are used in cooperation with the project leader to prepare the project plan.

8 Decision Making Approach in Specification and Design

Many specification and design decisions have to be taken during the product creation process. For example, functionality and performance requirements need to be defined, and the way to realize them has to be chosen. Many of these decisions are interrelated and have to be taken at a time when many uncertainties still exist.

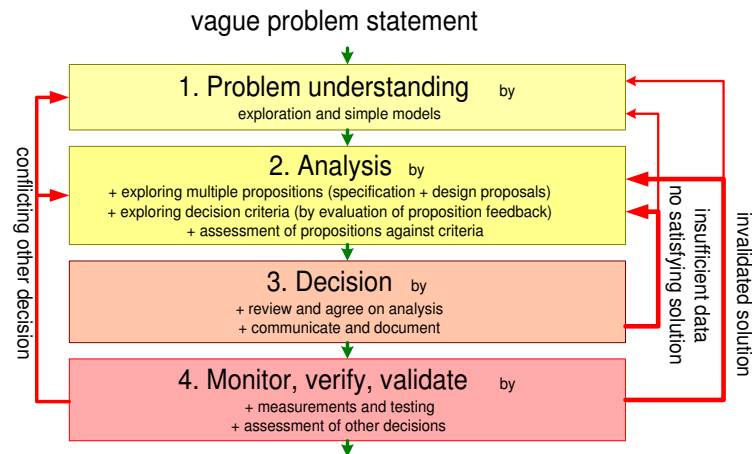


Figure 17: Flow from problem to solution

An approach to make these decisions is the flow depicted in Figure 17. The decision process is modeled in four steps. An understanding of the problem is created by the first step *problem understanding*, by exploration of problem and solution space. Simple models, in problem space as well as in solution space, help to create this understanding. The next step is to perform a somewhat more systematic *analysis*. The analysis is often based on *exploring multiple propositions*. The third step is the *decision* itself. The analysis results are reviewed, and the decision is documented and communicated. The last step is to *monitor, verify and validate* the decision.

The *analysis* involves multiple substeps: *exploring multiple propositions*, *exploring decision criteria* and *assessing the propositions against the criteria*. A proposition describes both specification (**what**) and design (*how*). Figure 18 shows an example of multiple propositions. In this example a high performance, but high cost alternative, is put besides two lower performing alternatives. Most criteria get articulated in the discussions about the propositions: “I think that we should choose proposition 2, because...”. The *because* can be reconstructed into a criterion.

The decision to chose a proposition is taken on the basis of the analysis results. A review of the analysis results ensures that these results are agreed upon. The

throughput	20 p/m	high-performance sensor	350 ns
cost	5 k\$	high-speed moves	9 m/s
safety		additional pipelining	
<i>low cost and performance 1</i>			
throughput	20 p/m	high-performance sensor	300 ns
cost	5 k\$	high-speed moves	10 m/s
safety			
<i>low cost and performance 2</i>			
throughput	25 p/m	highperformance sensor	200 ns
cost	7 k\$	high-speed moves	12 m/s
safety		additional collision detector	
<i>high cost and performance</i>			

Figure 18: Multiple propositions

decision itself is documented and communicated². In case of insufficient data or in absence of a satisfying solution we have to back track to the *analysis* step. Sometimes it is better to revisit the problem statement by going back to the *understanding* step.

Taking a decision requires a lot of follow up. The decision is in practice based on partial and uncertain data, and many assumptions. An significant amount of work is to monitor the consequences and implementation of the decision. Monitoring is partially a *soft skill*, such as actively listening to engineers, and partially a *engineering activity* such as measuring and testing. The consequence of a measurement can be that the problem has to be revisited, starting again with the understanding for serious mismatches (“apparently we don’t understand the problem at all”) or direct into the analysis for smaller mismatches.

The implementation of taken decisions can be disturbed by later decisions. This problem is partially tackled by requirements traceability, where known interdependencies are managed explicitly. In the complex real world the amount of dependencies is almost infinite, that means that the explicit dependability specifications are inherently incomplete and only partially understood. To cope with the inherent uncertainty about dependabilities, an open mind is needed when screening later decisions. A conflict caused by a later decision triggers a revisit of the original problem.

The same flow of activities is used recursively at different levels of detail, as shown in Figure 19. A *system* problem will result in a system design, where many design aspects need the same flow of problem solving activities for the subsystems. This process is repeated for smaller scopes until termination at problems that can be solved directly by an implementation team. The smallest scope of termination is denoted as *atomic* level in the figure. Note that the more detailed problem solving might have impact on the more global decisions.

²This sounds absolutely trivial, but unfortunately this step is performed quite poorly in practice.

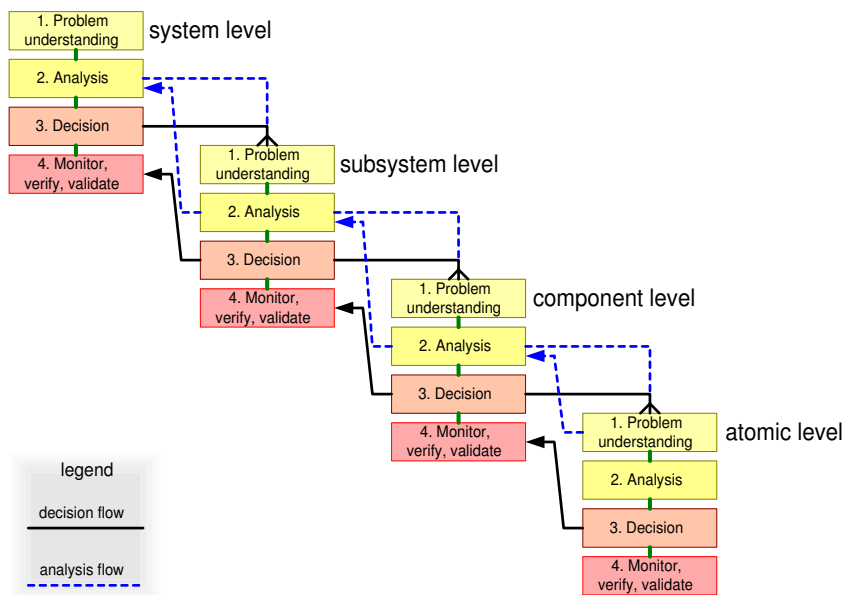


Figure 19: Recursive and concurrent application of flow

9 Acknowledgements

The team of composable architectures, with the following members Pierre America, Marcel Bijsterveld, Peter van den Hamer, Jürgen Müller, Henk Obbink, Rob van Ommering, and William van der Sterren within Philips Research provided valuable feedback for this article

References

- [1] Gerrit Muller. The system architecture homepage. <http://www.gaudisite.nl/index.html>, 1999.
- [2] Gerrit Muller. The role and task of the system architect. <http://www.gaudisite.nl/RoleSystemArchitectPaper.pdf>, 2000.
- [3] Gerrit Muller. Function profiles; the sheep with 7 legs. <http://www.gaudisite.nl/FunctionProfilesPaper.pdf>, 2001.
- [4] William H. Press, William T. Vetterling, Saul A. Teulosky, and Brian P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, Cambridge, England, 1992. Simulated annealing methods page 444 and further.

[5] Eberhardt Rechtin and Mark W. Maier. *The Art of Systems Architecting*. CRC Press, Boca Raton, Florida, 1997.

History

Version: 1.5, date: July 6, 2010 changed by: Gerrit Muller

- textual and layout adaptations

Version: 1.4, date: October 10, 2003 changed by: Gerrit Muller

- added rational and intuitive to figure with types of models

Version: 1.3, date: May 21, 2003 changed by: Gerrit Muller

- moved "problem solving approach" from chapter "Threads of Reasoning" to this chapter

Version: 1.2, date: September 3, 2002 changed by: Gerrit Muller

- updated figure example of the evolution of quantification in time
- updated figure about open perceptive and goal driven scanning

Version: 1.1, date: July 8, 2002 changed by: Gerrit Muller

- updated figure architecting time
- added quantification evolution in time

Version: 1.0, date: July 3, 2002 changed by: Gerrit Muller

- split figure viewpoint hopping in 2 figures: many viewpoints and sequence of viewpoints, text extended accordingly
- added coverage figure
- added from perceiving to goal driven figure
- added model types
- replaced figure "modelling" by inline text
- added section "decomposition and integration"
- added figure recursive use Why what how
- added figure architecting time
- added section "quantification"
- added acknowledgements

Version: 0, date: June 19, 2002 changed by: Gerrit Muller

- Created, no changelog yet