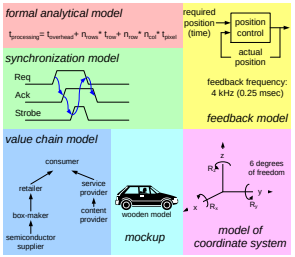
Basic Methods

-



Gerrit Muller

Buskerud University College

Frogs vei 41 P.O. Box 235, NO-3603 Kongsberg Norway

gaudisite@gmail.com

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.

version: 1.2 status: finished June 23, 2016

1 Introduction

The basic methods used by system 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 into 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 levels, 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 [7].

2 Viewpoint Hopping

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.

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 thousands[1] of these viewpoint changes.

The system description and implementation span a significant dynamic range. At the highest abstraction level a system can be characterized by its core function and the key performance figure. Via multiple decomposition steps the description

[1] Based on observations of other architects and own experience.

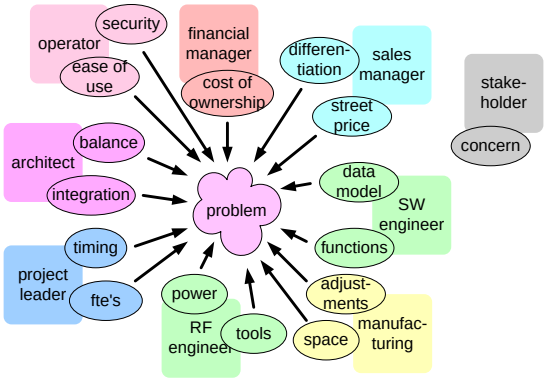


Figure 1: Small subset of stakeholders, concerns and viewpoints

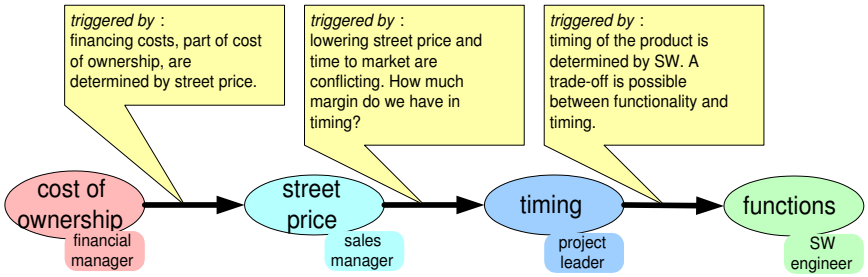


Figure 2: Short example of viewpoint hopping

is detailed to units that can be engineered. The implementation shows orders of magnitude more details. The source description of today's products is in the order of millions lines of code[2]. The source description expands via synthesis into an order magnitude more wires, gates, transistors and bytes. The amount of states in actual operation is again orders of magnitude larger.

Figure 3 shows this dynamic range. At the left hand abstraction levels in the creation life-cycle are shown. Both for hardware and software the typical entities at the different layers of abstraction are shown.

The viewpoints and dynamic range of abstraction create 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

[2]In 2003 the software figures for MR scanners, wafersteppers and televisions are all between 1 and 10 million lines of source code. Source code in the broad sense: all formalized definitions, that are created by humans and maintained and tested. Generated code is not counted.

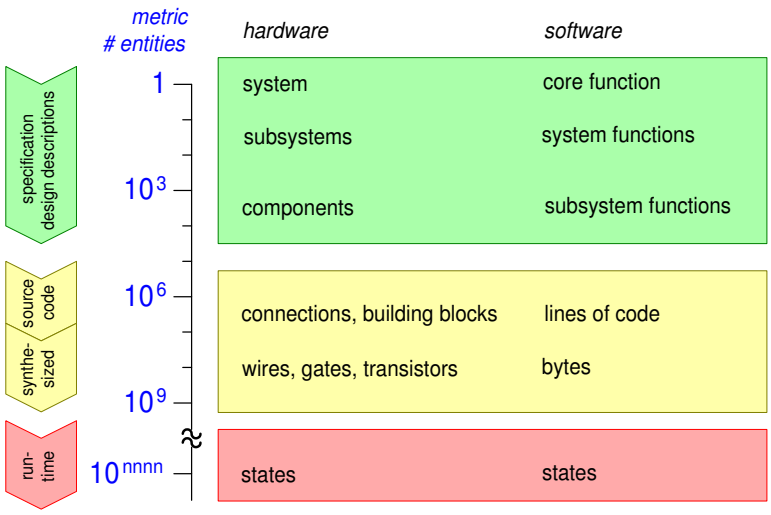


Figure 3: Dynamic range of description and implementation in a product

judging is required to reach a result in a limited amount of time and effort. See Figure 4 for these 2 modes of scanning.

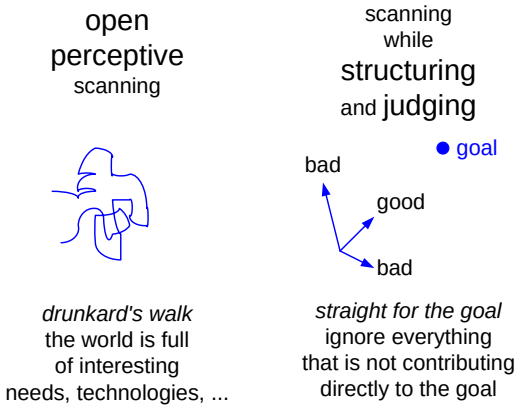


Figure 4: Two modes of scanning by an architect

The scanning approach taken by the architect can be compared with *simulated annealing methods* for optimization. An interesting quote from the book "Numerical Recipes in C; The Art of Scientific Computing"[6], about comparing optimization methods is:

> 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...

The exploration space is not exclusively covered by the architect(s), engineers will cover a large part of this space. The architect will focus mostly on the higher abstraction layers, *but* sufficient sampling of the lower layers is important to keep the higher layers meaningful.

3 Decomposition and Integration

The architect applies a reduction strategy by means of decomposition over and over. Decomposition is a very generic principle, that can be applied for many different problem and solution dimensions. Martin [4] uses the phrase development layers when the decomposition principle is applied on a system.

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 separating 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. In practice it is observed that the synthesis or integration part is more difficult and time consuming.

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. Thomas Gilb stresses the importance of quantification and estimation, see for instance [2].

The precision of the quantification increases during the project. Figure 5 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:

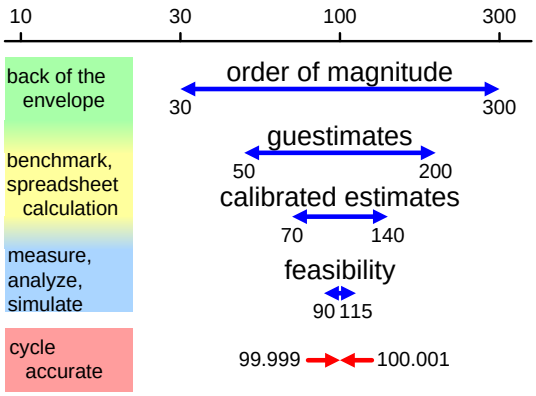


Figure 5: Successive quantification refined

- 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 5 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 6 shows a graphical example of an “overlay” budget for a waferstepper. 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 waferstepper. The budget is used to provide requirements for subsystems and components. 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.

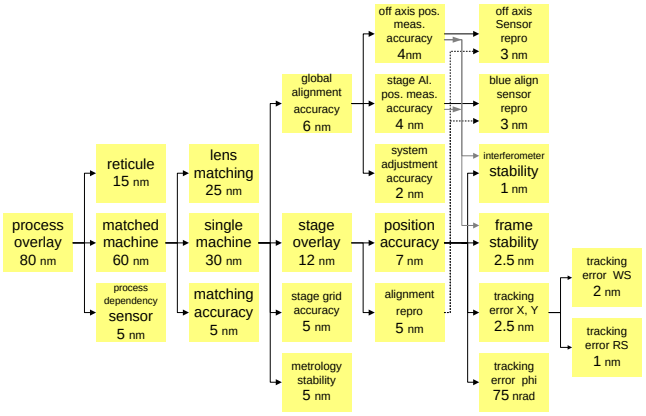


Figure 6: Example of a quantified understanding of overlay in a waferstepper

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.

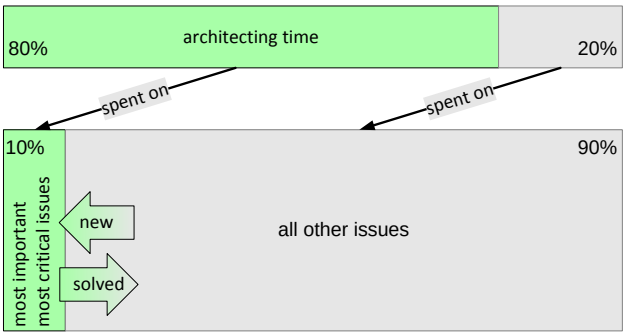


Figure 7: The architect focuses on important and critical issues, while monitoring the other issues

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 7 visualizes the time distribution of the architect: 80% of the time is spent on 10% of the issues. The well known 80/20 rule matches well with my own observations of how system

architects spend their time. The number of important issues that are addressed in the 80% of the time is also based on personal observations.

6 Modeling

modeling is one of the most fundamental tools of an architect. Gilb [2] defines a model as

> A model is an artificial representation of an idea or a product. The representation can be in any useful format including specifications, drawings and physical representations. (Gilb glossary of concepts)

Lieberman [3] explains the difficulty of making good models for human use.

In summary we can say that models are used to obtain insight and understanding, and that models serve a clear purpose. 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.

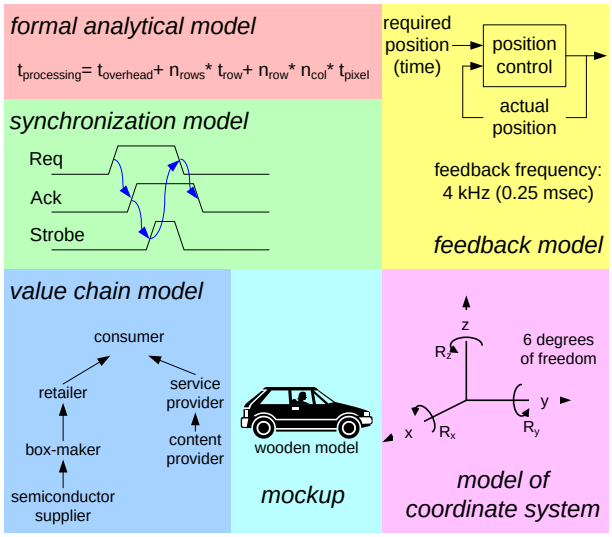


Figure 8: Some examples of models

Models exist in a very rich variety. A few examples of models are shown in Figure 8.

Models have many different manifestations. Figure 9 shows some of the different types of models, expressed in a number of adjectives.

Models can be *mathematical* (a mature field, see for instance [1]) expressed in formulas, they can be *linguistic*, expressed in words, or they can be *visual*,

mathematical linguistic	visual
formal	informal
quantitative	qualitative
detailed	global
concrete	abstract
accurate	approximate
executable	read only
←rational—	—intuitive→

Figure 9: Types of models

captured in diagrams. A model can be formal, where notations, operations and terms are precisely defined, or it can be 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

All "W" questions are an important tool for the architect. Figure 10 shows the useful starting words for questions to be asked by an architect.

Why	Who
What	When
How	Where

Figure 10: The starting words for questions by the 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 11 shows that the recursion with **why** questions broadens the scope, and recursion with **how** questions opens more details in a smaller scope.

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 cooper-

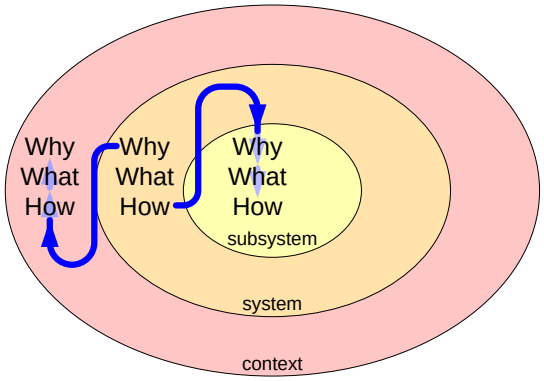


Figure 11: Why broadens scope, How opens details

ation 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, see section 5. The need for problem solving and decision making techniques is clearly recognized in companies like Philips and ASML. Quality improvement programs in these companies include education in these techniques. The approach described in this section is based on the techniques promoted by the quality improvement programs.

An approach to make these decisions is the flow depicted in Figure 12. 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 13 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 artic-

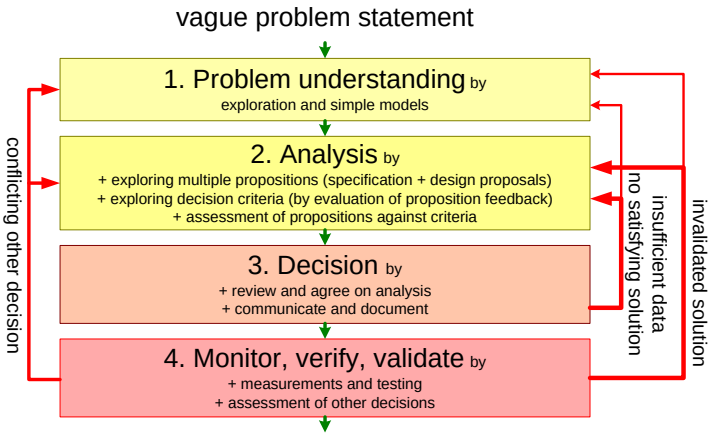


Figure 12: Flow from problem to solution

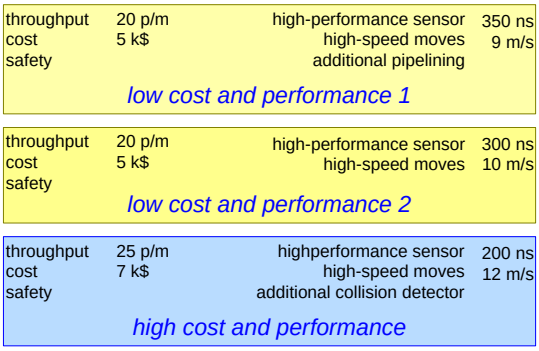


Figure 13: Multiple propositions

ulated 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 decision itself is documented and communicated[3]. 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 is based on many assumptions. A significant amount of work is to monitor the consequences and the implementation of the

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

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, because the solution is invalidated. An invalidated solution returns the process to the *understanding* step in case of serious mismatches ('apparently we don't understand the problem at all'). In case of smaller mismatches the process returns to the *analysis* step.

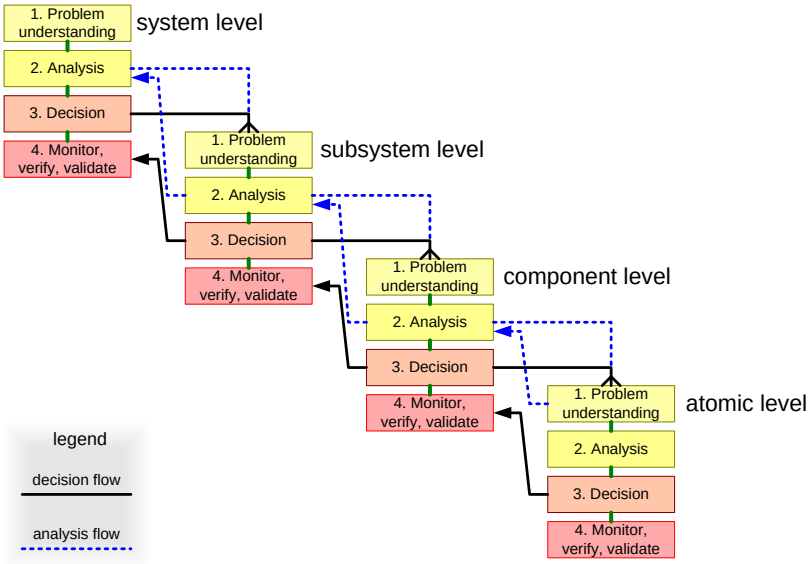


Figure 14: Recursive and concurrent application of flow

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 14. 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.

9 Acknowledgements "Basic methods"

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] Florian Cajori. *A history of Mathematical Notations*. The Open Court Publishing Company, 1928.

[2] Thomas Gilb. *Competitive Engineering: A Handbook For Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Elsevier Butterworth-Heinemann, London, 2005.

[3] Ben Lieberman. The art of modeling; part i: Constructing an analytical framework. `http://www.therationaledge.com/content/aug_03/f_modeling_bl.jsp`, 2003.

[4] James N. Martin. *Systems Engineering Guidebook*. CRC Press, Boca Raton, Florida, 1996.

[5] Gerrit Muller. The system architecture homepage. `http://www.gaudisite.nl/index.html`, 1999.

[6] 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.

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

History

Version: 1.2, date: April 6, 2003 changed by: Gerrit Muller

- minor textual updates
- updated figures decision flow
- changed status to finished

Version: 1.1, date: November 20, 2003 changed by: Gerrit Muller

- minor textual updates

- added source of architecting time distribution
- added source of decision making approach
- changed status to concept

Version: 1.0, date: October 10, 2003 changed by: Gerrit Muller

- added rational and intuitive to figure with types of models
- rewrite of Section ”Problem solving”
- deleted Figure ”Assessment of propositions”
- changed status to draft

Version: 0.2, date: October 3, 2003 changed by: Gerrit Muller

- removed language errors and rephrased many sentences upto Section ”Decomposition and Integration”
- simplified diagram explaining the dynamic range

Version: 0.1, date: July 31, 2003 changed by: Gerrit Muller

- reworked section ”Viewpoint hopping”
- reworked section ”Decomposition and integration”
- reworked section ”Quantification”

Version: 0, date: July 29, 2003 changed by: Gerrit Muller

- Started as a copy of ”Basic working methods of a system architect”