

System Modeling and Analysis: a Practical Approach

logo
TBD

Gerrit Muller

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

Abstract

This book provides a practical approach to model systems. This approach is based on the modeling of many system and context aspects in small and simple models. These small models have to be used together to support the creation process of systems. It extends the notion of threads of reasoning and build on methods such as key driver graphs. A fast iteration over top-down models and bottom-up models is promoted.

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/>

Contents

Introduction	v
I Overview	1
1 Modeling and Analysis Overview	2
1.1 Introduction	2
1.2 Overview of Modeling and Analysis Approach	4
1.3 Acknowledgements	7
II Fundamentals of Technology	8
2 Introduction to System Performance Design	9
2.1 Introduction	9
2.2 What if	9
2.3 Problem Statement	12
2.4 Summary	13
2.5 Acknowledgements	13
3 Modeling and Analysis Fundamentals of Technology	15
3.1 Introduction	15
3.2 Computing Technology Figures of Merit	16
3.3 Caching in Web Shop Example	19
3.4 Summary	24
4 Modeling and Analysis: Measuring	25
4.1 introduction	25
4.2 Measuring Approach	27
4.3 Summary	41
4.4 Acknowledgements	42

III	System Model	43
5	Modeling and Analysis: System Model	44
5.1	Introduction	44
5.2	Stepwise approach to system modeling	45
5.3	Example system modeling of web shop	46
5.4	Discussion	54
5.5	Summary	55
5.6	Acknowledgements	56
6	Modeling and Analysis: Budgeting	57
6.1	Introduction	57
6.2	Budget-Based Design method	58
6.3	Summary	66
6.4	Acknowledgements	66
IV	Application and Life Cycle Model	68
7	Modeling and Analysis: Life Cycle Models	69
7.1	Introduction	69
7.2	Life Cycle Modeling Approach	70
8	Simplistic Financial Computations for System Architects.	79
8.1	Introduction	79
8.2	Cost and Margin	80
8.3	Refining investments and income	81
8.4	Adding the time dimension	83
8.5	Financial yardsticks	86
8.6	Acknowledgements	88
9	The application view	89
9.1	Introduction	89
9.2	Customer stakeholders and concerns	90
9.3	Context diagram	91
9.4	Entity relationship model	92
9.5	Dynamic models	92
V	Integration and Reasoning	95
10	Modeling and Analysis: Reasoning	96
10.1	Introduction	96

10.2 From Chaos to Order	98
10.3 Approach to Reason with Multiple Models	100
10.4 Balancing Chaos and Order	110
10.5 Life Cycle of Models	112
10.6 Summary	114
10.7 Acknowledgements	115
 VI Analysis	 116

Introduction

This book provides a practical approach to model systems. This approach is based on the modeling of many system and context aspects in small and simple models. These small models have to be used together to support the creation process of systems.

The book follows the structure of the course *Modeling and Analysis*. Each course module is a *Part* with a number of Chapters:

part 1 Overview

part 2 Fundamentals of Technology

part 3 System Model

part 4 Application and Life Cycle Model

part 5 Integration and Reasoning

part 6 Analysis

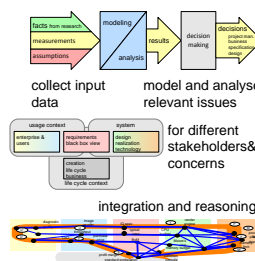
At this moment the book is in its early infancy. Only one running case is used at the moment, a web shop. Most articles are updated based on feedback from readers and students. The most up to date version of the articles can always be found at [4]. The same information can be found here in presentation format. Chapters can be read as autonomous units.

Part I

Overview

Chapter 1

Modeling and Analysis Overview



1.1 Introduction

At the beginning of the creation of a new product the problem is often ill-defined and only some ideas exist about potential solutions. The architecting effort must change this situation in the course of the project into a well articulated and structured understanding of both the problem and its potential solutions. Figure 1.1 shows that basic methods and an architecting method enable this architecting effort. We will zoom on modeling and analysis as support for the architecting effort.

Modeling and analysis supports the architecting in several ways during the project life cycle, see Figure 1.2. Early modeling and analysis efforts help to understand the problem and solution space. When the project gets more tangible the purpose of modeling shifts to exploration of specification and design alternatives. For some problems it is rewarding to optimize the solution by means of models. When the realization gets up and running, then model and realization can be compared for verification purposes.

The insight that the goal of modeling changes during the project life cycle implicates that the type of model depends on project phase. We should realize that every model that we create has a goal. These goals evolve and hence models evolve. The model itself will not achieve the goal. We have to actively pursue the goal, for instance by applying techniques, to reach this goal.

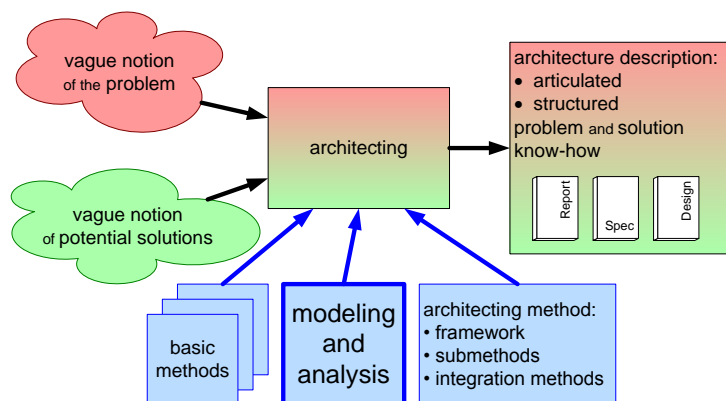


Figure 1.1: An architecting method supports the architect in his process to go from a vague notion of the problem and a vague notion of the potential solutions to a well articulated and structured architecture description. Modeling and Analysis supports the architecting effort.



Type of model depends on project phase

Models have a goal

Goals evolve and models evolve

Techniques are used to reach this goal

Figure 1.2: Modeling and Analysis supports:

The purpose of modeling is to support the architecting effort. Figure 1.3 makes this purpose more explicit: the purpose of modeling is to support the project to get the right specification, design and to take the right decisions to achieve the project goals. *Right* specification and design is assessed in terms of customer satisfaction, risk level, meeting project constraints (cost, effort, time), and business viability (profit margin). In order to get to this point we need information, modeling results, with sufficient *accuracy*, *working range*, and *credibility*. These modeling results are based on the inputs to modeling:

facts from investigations, such as searching supplier documentation, customer or stakeholder interviews, market research et cetera.

measurements of components, previous systems and the context

assumptions whenever facts or measurements are missing or too expensive.

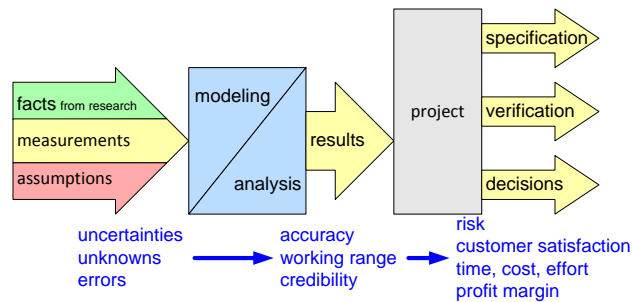


Figure 1.3: Purpose of Modeling

All these inputs have their uncertainties, may contain unknowns or may even be erroneous.

1.2 Overview of Modeling and Analysis Approach

The approach uses a simple model of the system and its context as shown in Figure 1.4. The system is modeled as black box, often called *system requirements*. Both functional or behavioral models can be used. However, we will focus mostly on the so-called *non-functional requirements* in the black box view. The internals of the system are also modeled: the design, the realization, and technology choices and consequences.

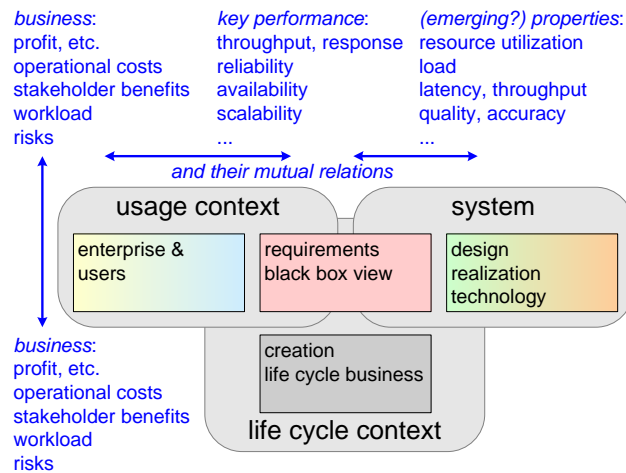


Figure 1.4: What to Model?

The purpose of modeling is to support the project in its architecting effort. The

project purpose is always to realize a system in its context. A good system is a system that fits in its context and that is appropriate for its purpose. Figure 1.4 shows that we will model the *usage* context and the *life cycle* context. The *usage* context is often an enterprise with business processes and human users. The life cycle context starts with the creation or project life cycle, but it continues for the entire operational life of the system. Often a service provider is the stakeholder taking care of the system life cycle, from infrastructure to maintenance services and possibly even support for higher level application services.

Many models can be made for the system and its context, see Figure 1.4 for some examples. However, to achieve the modeling goal we are often interested in the mutual relations between black box view and design, system and context, et cetera. We stress again, modeling is a means to support the process.

day 1	1. overall approach intro, overall approach, exercise overall approach
	2. input facts, data, uncertainties quantification, measurements, modeling, validation, technology background, lifecycle and business input sources
day 2	3. system modeling purpose, approaches, patterns, modularity, parametrization, means, exploration, visualization, micro-benchmarking, characterization, performance as example
	4. application, life-cycle modeling reiteration of modeling approach (see module 3), applied on customer application and business, and life cycle
day 3	5. integration and reasoning relating key driver models to design models, model based threads of reasoning, FMEA-like approach, modeling in project life-cycle
	6. analysis, using models sensitivity, robustness, worst case, working range, scalability, exceptions, changes

Figure 1.5: Program of Modeling and Analysis Course

The structure of the course and the supporting book follows the six modules shown in Figure 1.5.

overall approach providing an introduction and an overview of the overall approach to modeling and analysis.

input facts, data, uncertainties where and how to get quantified input data? We discuss measurements and the basis for dealing with uncertainties and errors. We also provide figures of merit for computer technology.

system modeling via a number of examples we show how the system and design

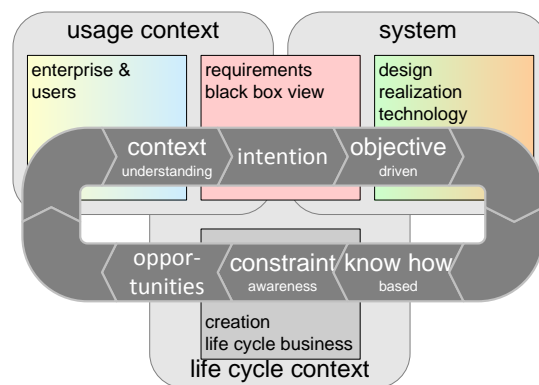


Figure 1.7: Iteration over viewpoints

thinking and bottom-up fact finding are continuously alternated. Top-down provides context understanding, intentions and objectives. Bottom-up provides know-how, constraints and opportunities.

1.3 Acknowledgements

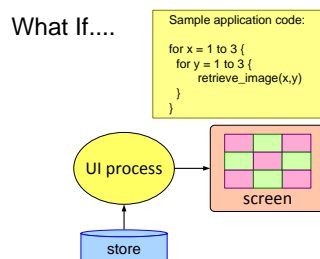
Dinesh Verma initiated the development of a Modeling and Analysis course. Russ Taylor kept asking for structure and why, and provided feedback.

Part II

Fundamentals of Technology

Chapter 2

Introduction to System Performance Design



2.1 Introduction

This article discusses a typical example of a performance problem during the creation of an additional function in an existing system context. We will use this example to formulate a problem statement. The problem statement is then used to identify ingredients to address the problem.

2.2 What if ...

Let's assume that the application asks for the display of $3 \cdot 3$ images to be displayed "instantaneously". The author of the requirements specification wants to sharpen this specification and asks for the expected performance of feasible solutions. For this purpose we assume a solution, for instance an image retrieval function with code that looks like the code in Figure 2.1. How do we predict or estimate the expected performance based on this code fragment?

If we want to estimate the performance we have to know what happens in the system in the `retrieve_image` function. We may have a simple system, as shown in

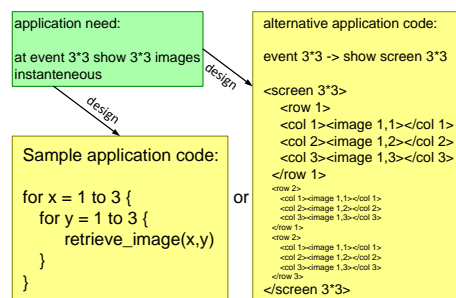


Figure 2.1: Image Retrieval Performance

Figure 2.2, where the `retrieve_image` function is part of a *user interface* process. This process reads image data directly from the hard disk based store and renders the image directly to the screen. Based on these assumptions we can estimate the performance. This estimation will be based on the disk transfer rate and the rendering rate.

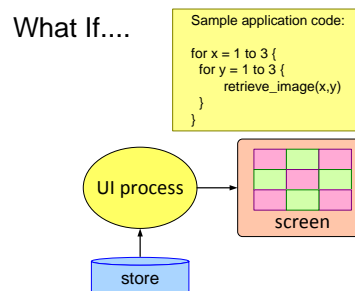


Figure 2.2: Straight Forward Read and Display

However, the system might be slightly more complex, as shown in Figure 2.3. Instead of one process we now have multiple processes involved: database, user interface process and screen server. Process communication becomes an additional contribution to the time needed for the image retrieval. If the process communication is image based (every call to `retrieve_image` triggers a database access and a transfer to the screen server) then $2 \cdot 9$ process communications takes place. Every process communication costs time due to overhead as well as due to copying image data from one process context to another process context. Also the database access will contribute to the total time. Database queries cost a significant amount of time.

The actual performance might be further negatively impacted by the overhead costs of the meta-information. Meta-information is the describing information of the image, typically tens to hundreds of attributes. The amount of data of meta-information, measured in bytes, is normally orders of magnitude smaller than the

What If....

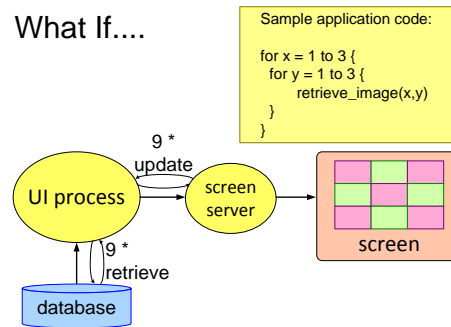


Figure 2.3: More Process Communication

amount of pixel data. The initial estimation ignores the cost of meta-information, because the of amount of data is insignificant. However, the chosen implementation does have a significant impact on the cost of meta-information handling. Figure 2.4 shows an example where the attributes of the meta-information are internally mapped on COM objects. The implementation causes a complete “factory” construction for every attribute that is retrieved. The cost of such a construction is $80\mu\text{sec}$. With 100 attributes per image we get a total construction overhead of $9 \cdot 100 \cdot 80\mu\text{s} = 72\text{ms}$. This cost is significant, because it is in the same order of magnitude as image transfer and rendering operations.

What If....

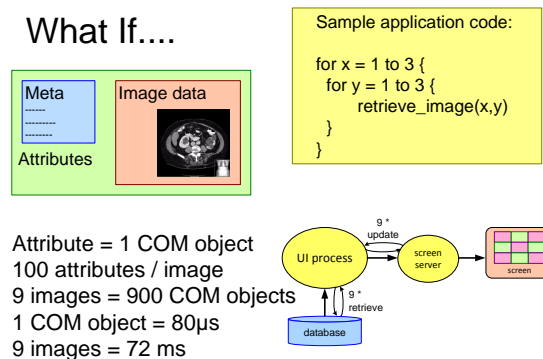


Figure 2.4: Meta Information Realization Overhead

Figure 2.5 shows I/O overhead as a last example of potential hidden costs. If the granularity of I/O transfers is rather fine, for instance based on image lines, then the I/O overhead becomes very significant. If we assume that images are 512^2 , and if we assume $t_{I/O} = 1\text{ms}$, then the total overhead becomes $9 \cdot 512 \cdot 1\text{ms} \approx 4.5\text{s}$!

What If....

Sample application code:

```
for x = 1 to 3 {  
  for y = 1 to 3 {  
    retrieve_image(x,y)  
  }  
}
```

- I/O on line basis (512^2 image)

$$9 * 512 * t_{I/O}$$

$$t_{I/O} \approx 1ms$$

- . . .

Figure 2.5: I/O overhead

2.3 Problem Statement

Sample application code:

```
for x = 1 to 3 {  
  for y = 1 to 3 {  
    retrieve_image(x,y)  
  }  
}
```

can be:

fast, but very local
slow, but very generic
slow, but very robust
fast and robust

...

*The emerging properties (behavior, performance)
cannot be seen from the code itself!*

*Underlying platform and neighbouring functions
determine emerging properties mostly.*

Figure 2.6: Non Functional Requirements Require System View

In the previous section we have shown that the performance of a new function cannot directly be derived from the code fragment belonging to this function. The performance depends on many design and implementation choices in the SW layers that are used. Figure 2.6 shows the conclusions based on the previous *What if* examples.

Figure 2.7 shows the factors outside our new function that have impact on the overall performance. All the layers used directly or indirectly by the function have impact, ranging from the hardware itself, up to middleware providing services. But also the neighboring functions that have no direct relation with our new function have impact on our function. Finally the environment including the user have impact on the performance.

Figure 2.8 formulates a problem statement in terms of a challenge: How to understand the performance of a function as a function of underlying layers and surrounding functions expressed in a manageable number of parameters? Where the size and complexity of underlying layers and neighboring functions is large (tens, hundreds or even thousands man-years of software).

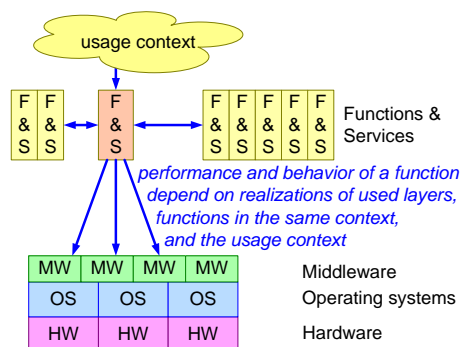
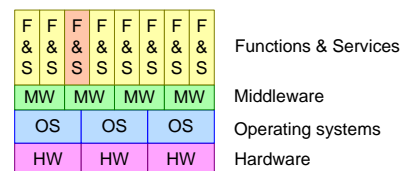


Figure 2.7: Function in System Context



Performance = Function (F&S, other F&S, MW, OS, HW)
MW, OS, HW >> 100 Manyyear : very complex

Challenge: How to understand MW, OS, HW
with only a few parameters

Figure 2.8: Challenge

2.4 Summary

We have worked through a simple example of a new application level function. The performance of this function cannot be predicted by looking at the code of the function itself. The underlying platform, neighboring applications and user context all have impact on the performance of this new function. The underlying platform, neighboring applications and user context are often large and very complex. We propose to use models to cope with this complexity.

2.5 Acknowledgements

The diagrams are a joined effort of Roland Mathijssen, Teun Hendriks and Gerrit Muller. Most of the material is based on material from the EXARCH course created by Ton Kostelijnk and Gerrit Muller.

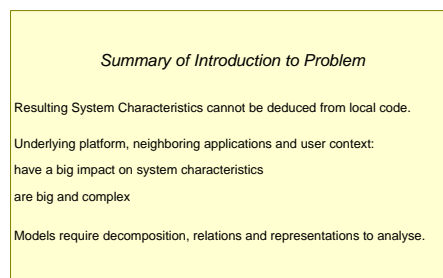
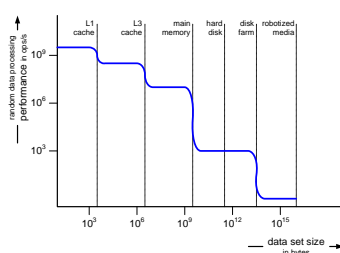


Figure 2.9: Summary of Problem Introduction

Chapter 3

Modeling and Analysis Fundamentals of Technology



3.1 Introduction

Figure 3.1 provides an overview of the content. In this article we discuss generic know how of computing technology. We will start with a commonly used decomposition and layering. We provide *figures of merit* for several generic computing functions, such as storage and communication. Finally we discuss caching as example of a technology that is related to storage figures of merit. We will apply the caching in a web shop example, and discuss design considerations.

<i>content of this presentation</i>
generic layering and block diagrams
typical characteristics and concerns
figures of merit
example of picture caching in web shop application

Figure 3.1: Overview Content *Fundamentals of Technology*

When we model technology oriented design questions we often need feasibility answers that are assessed at the level of non functional system requirements. Figure 3.2 shows a set of potential technology questions and the required answers at system level.

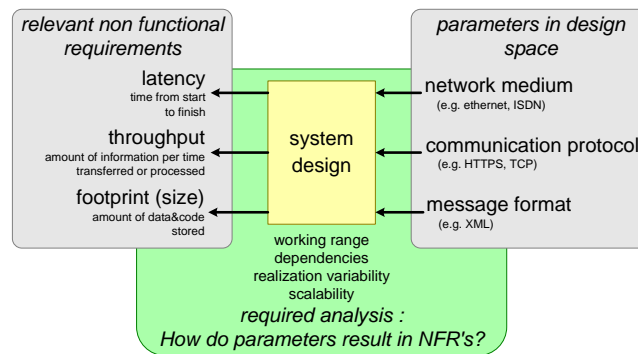


Figure 3.2: What do We Need to Analyze?

From design point of view we need, for example, information about the working range, dependencies, variability of the actual realization, or scalability.

3.2 Computing Technology Figures of Merit

In information and communication systems we can distinguish the following generic technology functions:

storage ranging from short term volatile storage to long term persistent storage.

Storage technologies range from solid state static memories to optical disks or tapes.

communication between components, subsystems and systems. Technologies range from local interconnects and busses to distributed networks.

processing of data, ranging from simple control, to presentation to compute intensive operations such as 3D rendering or data mining. Technologies range from general purpose CPUs to dedicated I/O or graphics processors.

presentation to human beings, the final interaction point with the human users. Technologies range from small mobile display devices to large “cockpit” like control rooms with many flat panel displays.

Figure 3.3 shows these four generic technologies in the typical layering of a *Service Oriented Architecture* (SOA). In such an architecture the repositories, the bottom-tier of this figure, are decoupled from the business logic that is being

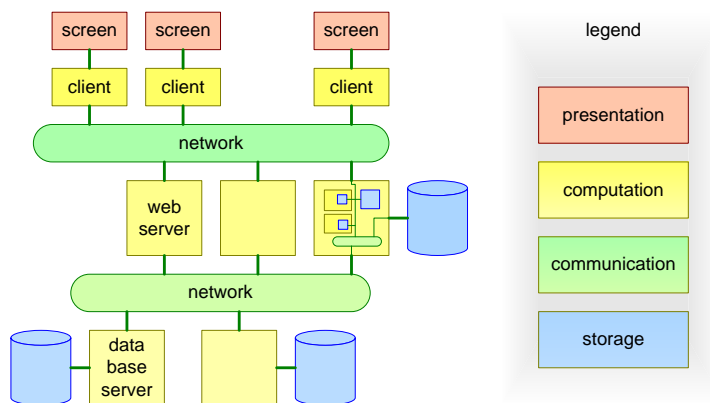


Figure 3.3: Typical Block Diagram and Typical Resources

handled in the middle layer, called *web server*. The client tier is the access and interaction layer, which can be highly distributed and heterogeneous.

The four generic technologies are recursively present: within a web-server, for example, communication, storage and processing are present. If we would zoom in further on the CPU itself, then we would again see the same technologies.

		latency	capacity
processor cache	<i>L1 cache</i>	sub ns	n kB
	<i>L2 cache</i>		
	<i>L3 cache</i>	ns	n MB
fast volatile	<i>main memory</i>	tens ns	n GB
persistent	<i>disks</i>		n*100 GB
	<i>disk arrays</i>	ms	
	<i>disk farms</i>		n*10 TB
archival	<i>robotized optical media tape</i>	>s	n PB

Figure 3.4: Hierarchy of Storage Technology *Figures of Merit*

For every generic technology we can provide *figures of merit* for several characteristics. Figure 3.4 shows a table with different storage technologies. The table provides typical data for latency and storage capacity. Very fast storage technologies tend to have a small capacity. For example, L1 caches, static memory as part of the CPU chip, run typically at processor speeds of several GHz, but their capacity

is limited to several kilobytes. The much higher capacity main memory, solid state dynamic RAM, is much slower, but provides Gigabytes of memory. Non solid state memories use block access: data is transferred in chunks of many kilobytes. The consequence is that the access time for a single byte of information gets much longer, milliseconds for hard disks. When mechanical constructions are needed to transport physical media, such as robot arms for optical media, then the access time gets dominated by the physical transport times.

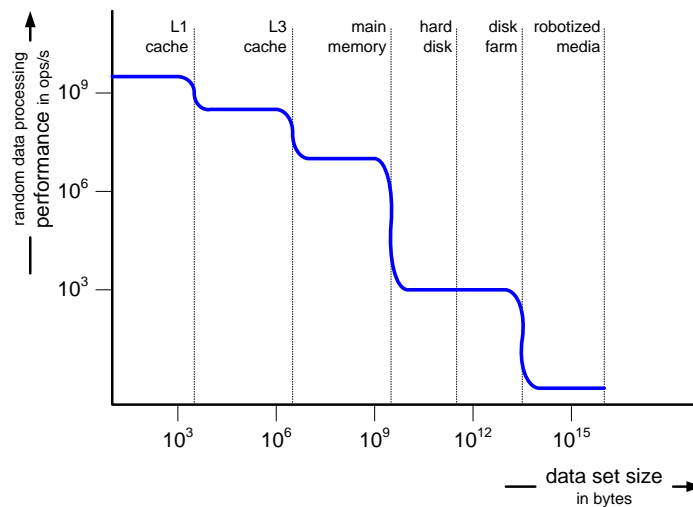


Figure 3.5: Performance as Function of Data Set Size

Figure 3.5 shows the same storage figures of merit in a 2-dimensional graph. The horizontal axis shows the capacity or the maximum data set size that we can store. The vertical axis shows the latency if we access a single byte of information in the data set in a random order. Note that both axes are shown as a logarithmic scale, both axes cover a dynamic range of many orders of magnitude! The resulting graph shows a rather non-linear behavior with step-like transitions. We can access data very fast up to several kilobytes; the access time increases significantly when we exceed the L1 cache capacity. This effect repeats itself for every technology transition.

The communication figures of merit are shown in the same way in Figure 3.6. In this table we show *latency*, *frequency* and *distance* as critical characteristics. The latency and the distance have a similar relationship as latency and capacity for storage: longer distance capabilities result in longer latencies. The frequency behavior, which relates directly to the transfer capacity, is different. On chip very high frequencies can be realized. Off chip and on the printed circuit board these high frequencies are much more difficult and costly. When we go to the long-

		latency	frequency	distance
on chip	connection	sub ns	n GHz	n mm
	network	n ns	n GHz	n mm
PCB level		tens ns	n 100MHz	n cm
Serial I/O		n ms	n 100MHz	n m
network	LAN	n ms	100MHz	n km
	WAN	n 10ms	n GHz	global

Figure 3.6: Communication Technology Figures of Merit

distance networks optical technologies are being used, with very high frequencies.

3.3 Caching in Web Shop Example

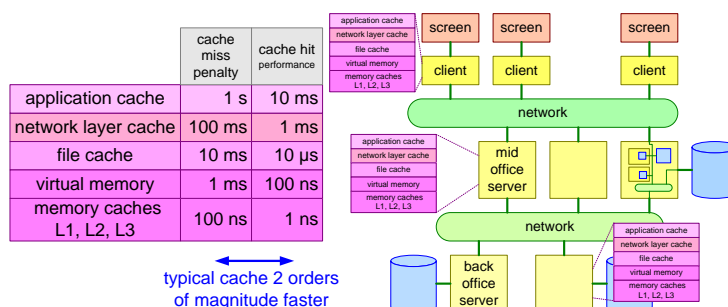


Figure 3.7: Multiple Layers of Caching

The speed differences in storage and communication often result in the use of a cache design pattern. The cache is a local fast storage, where frequently used data is stored to prevent repeated slow accesses to slow storage media. Figure 3.7 shows that this caching pattern is applied at many levels within a system, for example:

network layer cache to avoid network latencies for distributed data. Many communication protocol stacks, such as http, have local caches.

file cache as part of the operating system. The file cache caches the stored data itself as well as directory information in main memory to speed up many file operations.

- storage location
- cache size
- chunk size
- format

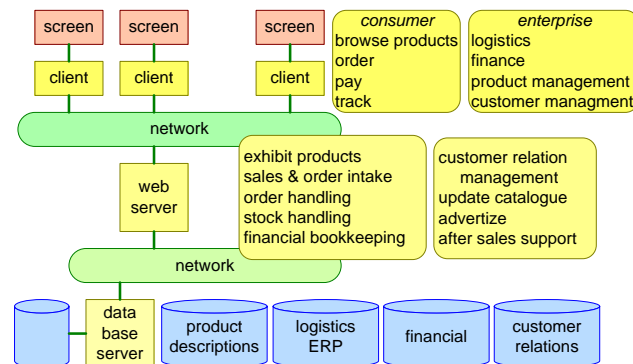


Figure 3.9: Example Web Shop

As an example of caching we look at a web shop, as shown in Figure 3.9. Customers at client level should be able to browse the product catalogue, to order products, to pay, and to track the progress of the order. Other stakeholders at client level have logistics functions, financial functions, and can do product and customer management. The web server layer provides the logic for the exhibition of products, the sales and order intake, the order handling, the stock handling, and the financial bookkeeping. Also at the web server layer is the logic for customer relation management, the update of the product catalogue, the advertisements, and the after sales support. The data base layer has repositories for product descriptions, logistics and resource planning, customer relations, and financial information.

We will zoom in on the product browsing by the customers. During this browsing customers can see pictures of the products in the catalogue. The originals of these pictures reside in the product catalogue repository in the data base layer. The web server determines when and how to show products for customers. The actual pictures are shown to many customers, who are distributed widely over the country.

The customers expect a fast response when browsing. Slow response may result in loss of customer attention and hence may cause a reduced sales. A picture cache at the web server level decreases the load at web server level, and at the same time improves the response time for customer browsing. It also reduces the server load of the data base.

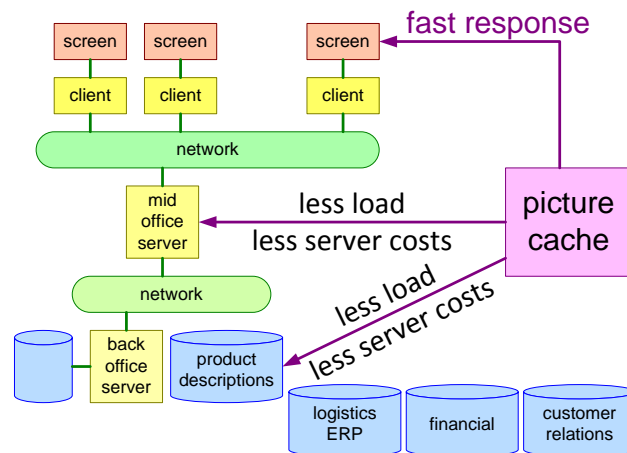


Figure 3.10: Impact of Picture Cache

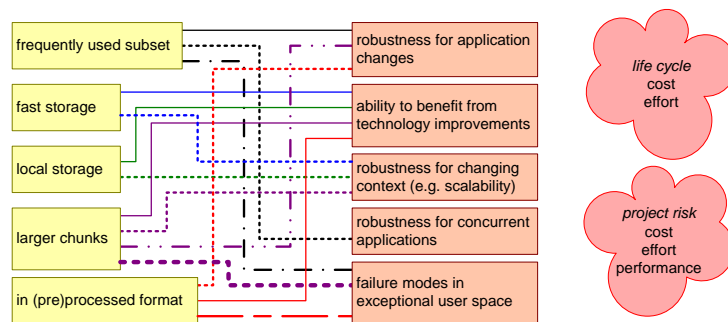


Figure 3.11: Risks of Caching

So far, the caching appears to be a no-brainer: improved response, reduces server loads, what more do we want? However, Figure 3.11 shows the potential risks of caching, caused mostly by increased complexity and decreased transparency. These risks are:

- The robustness for application changes may decrease, because the assumptions are not true anymore.
- The design becomes specific for this technology, impacting the ability to benefit from technology improvements.
- The robustness for changing context (e.g. scalability) is reduced
- The design is not robust for concurrent applications

- Failure modes in exceptional user space may occur

All of these technical risks translate in project risks in terms of cost, effort and performance.

3.4 Summary

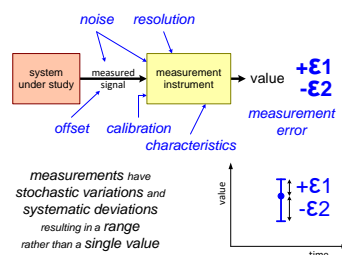
<i>Conclusions</i> Technology characteristics can be discontinuous Caches are an example to work around discontinuities Caches introduce complexity and decrease transparency
<i>Techniques, Models, Heuristics of this module</i> Generic block diagram: Presentation, Computation, Communication and Storage Figures of merit Local reasoning (e.g. cache example)

Figure 3.12: Summary

Figure 3.12 shows a summary of this paper. We showed a generic block diagram with *Presentation*, *Computation*, *Communication* and *Storage* as generic computing technologies. Technology characteristics of these generic technologies have discontinuous characteristics. At the transition from one type of technology to another type of technology a steep transition of characteristics takes place. We have provided *figures of merit* for several technologies. Caches are an example to work around these discontinuities. However, caches introduce complexity and decrease the transparency of the design. We have applied local reasoning graphs to discuss the reasons of introduction of caches and the related design parameters. later we applied the same type of graph to discuss potential risks caused by the increased complexity and decreased transparency.

Chapter 4

Modeling and Analysis: Measuring



4.1 introduction

Measurements are used to calibrate and to validate models. Measuring is a specific knowledge area and skill set. Some educations, such as Physics, extensively teach experimentation. Unfortunately, the curriculum of studies such as software engineering and computer sciences has abstracted away from this aspect. In this paper we will address the fundamentals of modeling.

Figure 4.1 shows the content of this paper. The crucial aspects of measuring are integrated into a measuring approach, see the next section.

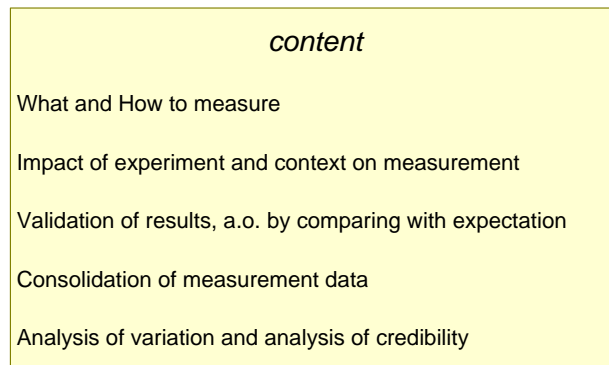


Figure 4.1: Presentation Content

4.2 Measuring Approach

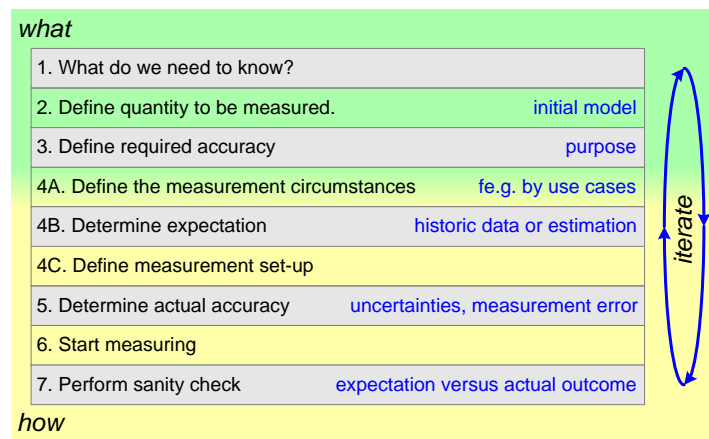


Figure 4.2: Measuring Approach: What and How

The measurement approach starts with *preparation and fact finding* and ends with *measurement and sanity check*. Figure 4.2 shows all steps and emphasizes the need for iteration over these steps.

- 1. What do we need?** What is the problem to be addressed, so what do we need to know?
- 2. Define quantity to be measured** Articulate as sharp as possible what quantity needs to be measured. Often we need to create a mental model to define this quantity.
- 3. Define required accuracy** The required accuracy is based on the problem to be addressed and the purpose of the measurement.
- 4A. Define the measurement circumstances** The system context, for instance the amount of concurrent jobs, has a big impact on the result. This is a further elaboration of step 1 *What do we need?*.
- 4B. Determine expectation** The experimentator needs to have an expectation of the quantity to be measured to design the experiment and to be able to assess the outcome.
- 4C. Define measurement set-up** The actual design of the experiment, from input stimuli, measurement equipment to outputs.

Note that the steps 4A, 4B and 4C mutually influence each other.

- 5. Determine actual accuracy** When the set-up is known, then the potential measurement errors and uncertainties can be analyzed and accumulated into a total actual accuracy.
- 6. Start measuring** Perform the experiment. In practice this step has to be repeated many times to “debug” the experiment.
- 7. Perform sanity check** Does the measurement result makes sense? Is the result close to the expectation?

In the next subsections we will elaborate this approach further and illustrate the approach by measuring a typical embedded controller platform: ARM9 and VxWorks.

4.2.1 What do we need?

The first question is: “What is the problem to be addressed, so what do we need to know?” Figure 4.3 provides an example. The *problem* is the need for guidance for *concurrency design* and *task granularity*. Based on experience the designers know that these aspects tend to go wrong. The effect of poor *concurrency design* and *task granularity* is poor performance or outrageous resource consumption.

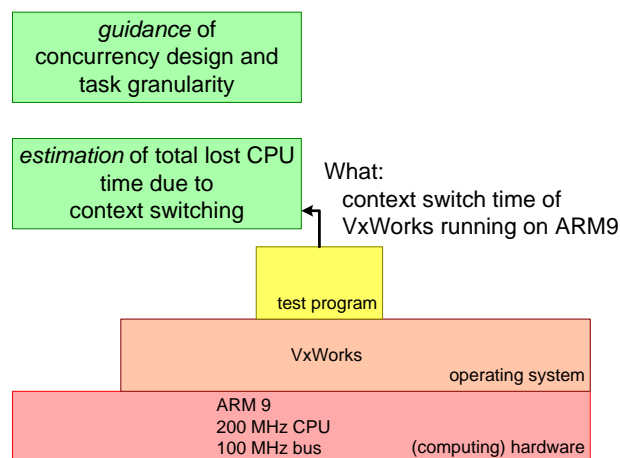


Figure 4.3: What do We Need? Example Context Switching

The designers know, also based on experience, that *context switching* is costly and critical. They have a need to estimate the total amount of CPU time lost due to context switching. One of the inputs needed for this estimation is the cost in CPU time of a single context switch. This cost is a function of the hardware platform, the operating system and the circumstances. The example in Figure 4.3 is based on

the following hardware: ARM9 CPU running internally at 200 MHz and externally at 100 MHz. The operating system is VxWorks. VxWorks is a real-time executive frequently used in embedded systems.

4.2.2 Define quantity to be measured.

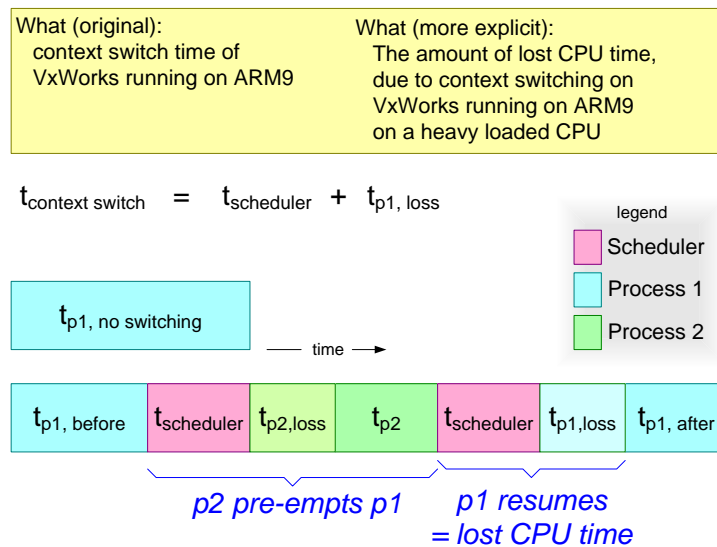


Figure 4.4: Define Quantity by Initial Model

As need we have defined the CPU cost of context switching. Before setting up measurements we have to explore the required quantity some more so that we can define the quantity more explicit. In the previous subsection we already mentioned shortly that the context switching time depends on the circumstances. The a priori knowledge of the designer is that context switching is especially significant in busy systems. Lots of activities are running concurrently, with different periods and priorities.

Figure 4.4 defines the quantity to be measured as the total cost of context switching. This total cost is not only the overhead cost of the context switch itself and the related administration, but also the negative impact on the cache performance. In this case the a priori knowledge of the designer is that a context switch causes additional cache loads (and hence also cache pollution). This cache effect is the term $t_{p1, \text{loss}}$ in Figure 4.4. Note that these effects are not present in a lightly loaded system that may completely run from cache.

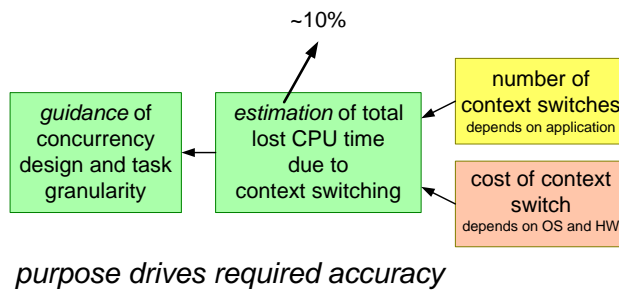


Figure 4.5: Define Required Accuracy

4.2.3 Define required accuracy

The required accuracy of the measurement is determined by the need we originally formulated. In this example the need is the ability to *estimate* the total lost CPU time due to context switching. The key word here is *estimate*. Estimations don't require the highest accuracy, we are more interested in the order of magnitude. If we can estimate the CPU time with an accuracy of tens of percents, then we have useful facts for further analysis of for instance task granularity.

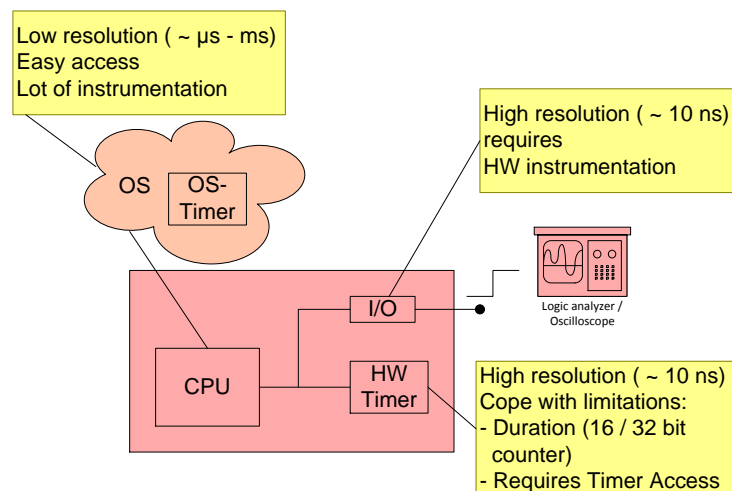


Figure 4.6: How to Measure CPU Time?

The relevance of the required accuracy is shown by looking at available measurement instruments. Figure 4.6 shows a few alternatives for measuring time on this type of platforms. The most easy variants use the instrumentation provided by the operating system. Unfortunately, the accuracy of the operating system timing is often very limited. Large operating systems, such as Windows and Linux, often

provide 50 to 100 Hz timers. The timing resolution is then 10 to 20 milliseconds. More dedicated OS-timer services may provide a resolution of several microseconds. Hardware assisted measurements make use of hardware timers or logic analyzers. This hardware support increases the resolution to tens of nanoseconds.

4.2.4 Define the measurement circumstances

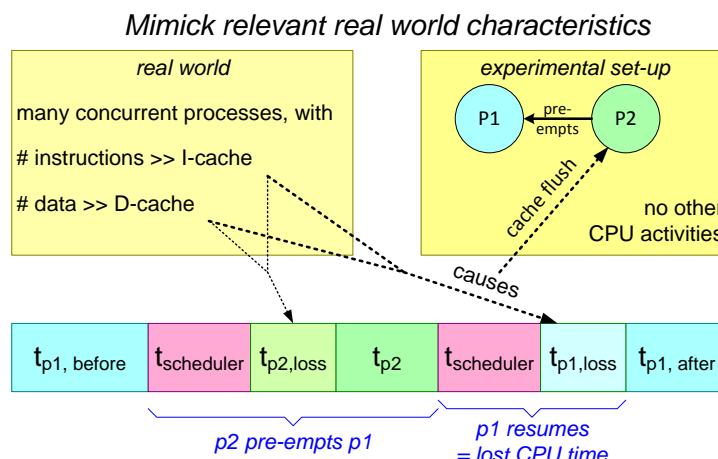


Figure 4.7: Define the Measurement Set-up

We have defined that we need to know the context switching time under *heavy load* conditions. In the final application *heavy load* means that we have lots of cache activity from both instruction and data activities. When a context switch occurs the most likely effect is that the process to be run is not in the cache. We lose time to get the process back in cache.

Figure 4.7 shows that we are going to mimick this cache behavior by flushing the cache in the small test processes. The overall set-up is that we create two small processes that alternate running: Process *P2* pre-empts process *P1* over and over.

4.2.5 Determine expectation

Determining the expected outcome of the measurement is rather challenging. We need to create a simple model of the context switch running on this platform. Figures 4.8 and 4.9 provide a simple hardware model. Figure 4.10 provides a simple software model. The hardware and software models are combined in Figure 4.11. After substitution with assumed numbers we get a number for the expected outcome, see Figure 4.12.

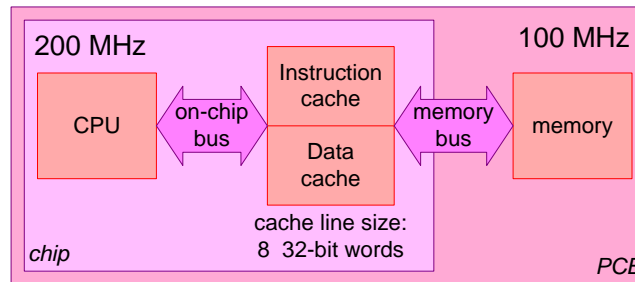


Figure 4.8: Case: ARM9 Hardware Block Diagram

Figure 4.8 shows the hardware block diagram of the ARM9. A typical chip based on the ARM9 architecture has anno 2006 a clock-speed of 200 MHz. The memory is off-chip standard DRAM. The CPU chip has on-chip cache memories for instruction and data, because of the long latencies of the off-chip memory access. The memory bus is often slower than the CPU speed, anno 2006 typically 100 MHz.

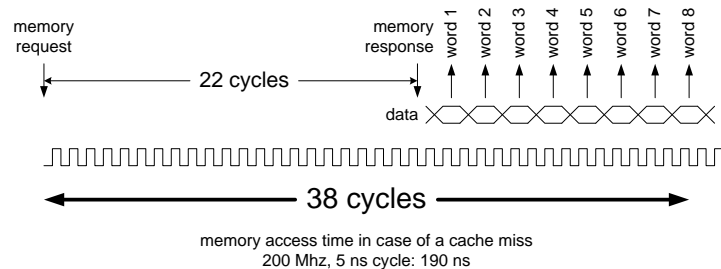


Figure 4.9: Key Hardware Performance Aspect

Figure 4.9 shows more detailed timing of the memory accesses. After 22 CPU cycles the memory responds with the first word of a memory read request. Normally an entire cache line is read, consisting of 8 32-bit words. Every word takes 2 CPU cycles = 1 bus cycle. So after $22 + 8 * 2 = 38$ cycles the cache-line is loaded in the CPU.

Figure 4.10 shows the fundamental scheduling concepts in operating systems. For context switching the most relevant process states are *ready*, *running* and *waiting*. A context switch results in state changes of two processes and hence in scheduling and administration overhead for these two processes.

Figure 4.11 elaborates the software part of context switching in five contributing activities:

- save state P1

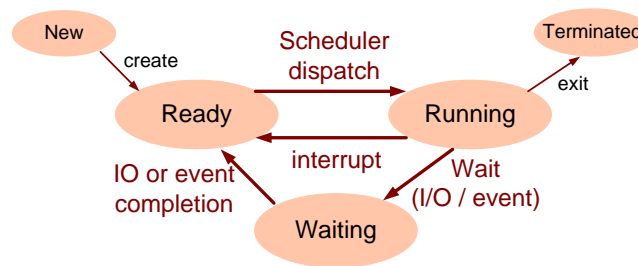


Figure 4.10: OS Process Scheduling Concepts

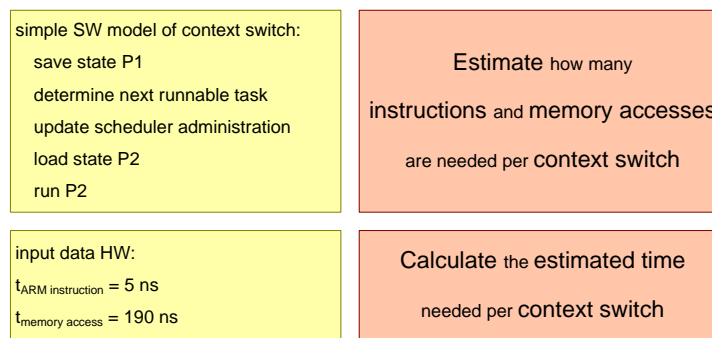


Figure 4.11: Determine Expectation

- determine next runnable task
- update scheduler administration
- load state P2
- run P2

The cost of these 5 operations depend mostly on 2 hardware depending parameters: the numbers of instruction needed for each activity and the amount of memory accesses per activity. From the hardware models, Figure 4.9, we know that as simplest approximation gives us an instruction time of 5ns ($= 1$ cycle at 200 MHz) and memory accesses of 190ns . Combining all this data together allows us to estimate the context switch time.

In Figure 4.12 we have substituted estimated number of instructions and memory accesses for the 5 operations. The assumption is that very simple operations require 10 instructions, while the somewhat more complicated scheduling operation requires scanning some data structure, assumed to take 50 cycles here. The estimation is now reduced to a simple set of multiplications and additions: $(10 + 50 + 20 +$

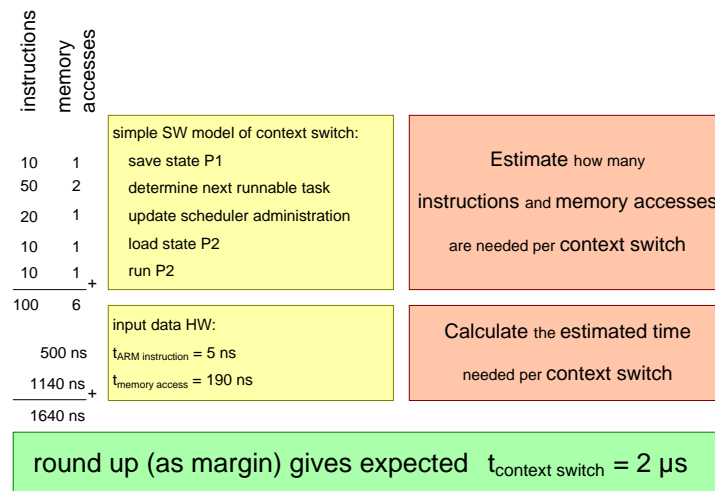


Figure 4.12: Determine Expectation Quantified

$10 + 10)instructions \cdot 5ns + (1 + 2 + 1 + 1 + 1)memoryaccesses \cdot 190ns$
 $= 500ns(instructions) + 1140ns(memoryaccesses) = 1640ns$ To add some margin for unknown activities we round this value to $2\mu s$.

4.2.6 Define measurement set-up

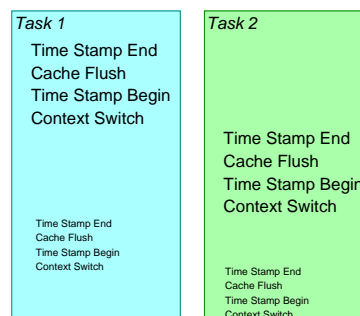


Figure 4.13: Code to Measure Context Switch

Figure 4.13 shows pseudo code to create two alternating processes. In this code time stamps are generated just before and after the context switch. In the process itself a cache flush is forced to mimick the loaded situation.

Figure 4.14 shows the CPU use as function of time for the two processes and the scheduler.

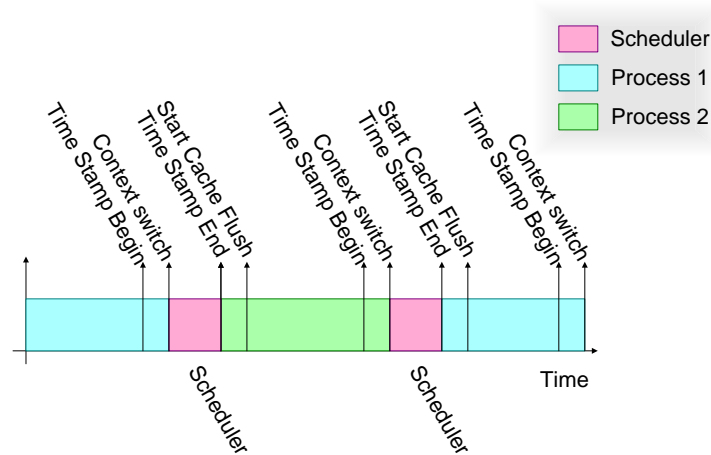


Figure 4.14: Measuring Context Switch Time

4.2.7 Expectation revisited

Once we have defined the measurement set-up we can again reason more about the expected outcome. Figure 4.15 is again the CPU activity as function of time. However, at the vertical axis the CPI (Clock cycles Per Instruction) is shown. The CPI is an indicator showing the effectiveness of the cache. If the CPI is close to 1, then the cache is rather effective. In this case little or no main memory accesses are needed, so the CPU does not have to wait for the memory. When the CPU has to wait for memory, then the CPI gets higher. This increase is caused by the waiting cycles necessary for the main memory accesses.

Figure 4.15 clearly shows that every change from the execution flow increases (worsens) the CPI. So the CPU is slowed down when entering the scheduler. The CPI decreases while the scheduler is executing, because code and data gets more and more from cache instead of main memory. When Process 2 is activated the CPI again worsens and then starts to improve again. This pattern repeats itself for every discontinuity of the program flow. In other words we see this effect twice for one context switch. One interruption of $P1$ by $P2$ causes two context switches and hence four dips of the cache performance.

4.2.8 Determine actual accuracy

Measurement results are in principle a range instead of a single value. The signal to be measured contains some noise and may have some offset. Also the measurement instrument may add some noise and offset. Note that this is not limited to the analog world. For instance concurrent background activities may cause noise as well as offsets, when using bigger operating systems such as Windows or Linux.

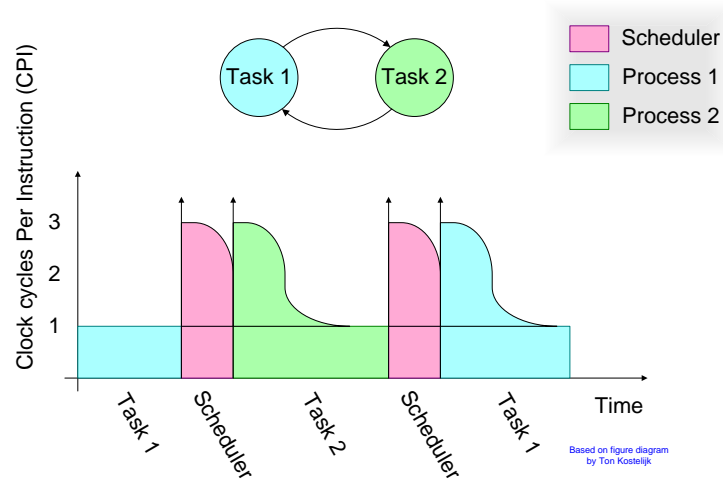


Figure 4.15: Understanding: Impact of Context Switch

The (limited) resolution of the instrument also causes a measurement error. Known systematic effects, such as a constant delay due to background processes, can be removed by calibration. Such a calibration itself causes a new, hopefully smaller, contribution to the measurement error.

Note that contributions to the measurement error can be stochastic, such as noise, or systematic, such as offsets. Error accumulation works differently for stochastic or systematic contributions: stochastic errors can be accumulated quadratic $\varepsilon_{total} = \sqrt{\varepsilon_1^2 + \varepsilon_2^2}$, while systematic errors are accumulated linear $\varepsilon_{total} = \varepsilon_1 + \varepsilon_2$.

Figure 4.17 shows the effect of error propagation. Special attention should be paid to subtraction of measurement results, because the values are subtracted while the errors are added. If we do a single measurement, as shown earlier in Figure 4.13, then we get both a start and end value with a measurement error. Subtracting these values adds the errors. In Figure 4.17 the provided values result in $t_{duration} = 4 + / - 4\mu s$. In other words when subtracted values are close to zero then the error can become very large in relative terms.

The whole notion of measurement values and error ranges is more general than the measurement sec. Especially models also work with ranges, rather than single values. Input values to the models have uncertainties, errors et cetera that propagate through the model. The way of propagation depends also on the nature of the error: stochastic or systematic. This insight is captured in Figure 4.18.

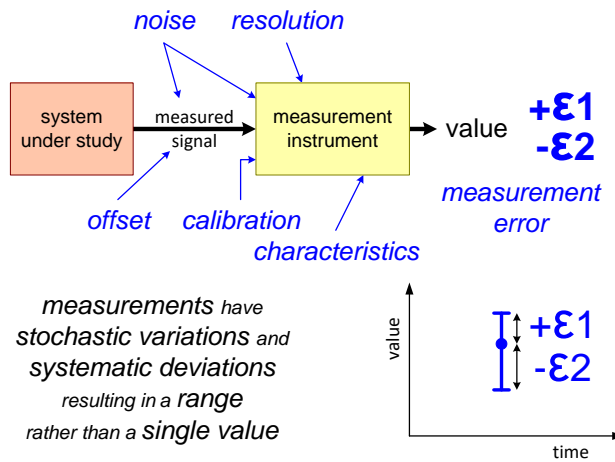


Figure 4.16: Accuracy: Measurement Error

$$t_{\text{duration}} = t_{\text{end}} - t_{\text{start}}$$

$$t_{\text{start}} = 10 \pm 2 \mu\text{s}$$

$$t_{\text{end}} = 14 \pm 2 \mu\text{s}$$

$$t_{\text{duration}} = 4 \pm ? \mu\text{s}$$

systematic errors: add linear

stochastic errors: add quadratic

Figure 4.17: Accuracy 2: Be Aware of Error Propagation

4.2.9 Start measuring

At OS level a micro-benchmark was performed to determine the context switch time of a real-time executive on this hardware platform. The measurement results are shown in Figure 4.19. The measurements were done under different conditions. The most optimal time is obtained by simply triggering continuous context switches, without any other activity taking place. The effect is that the context switch runs entirely from cache, resulting in a $2\mu\text{s}$ context switch time. Unfortunately, this is a highly misleading number, because in most real-world applications many activities are running on a CPU. The interrupting context switch pollutes the cache, which slows down the context switch itself, but it also slows down the interrupted activity. This effect can be simulated by forcing a cache flush in the context switch. The performance of the context switch with cache flush degrades to $10\mu\text{s}$. For comparison the measurement is also repeated with a disabled cache, which decreases the context switch even more to $50\mu\text{s}$. These measurements show

Measurements have stochastic variations and systematic deviations resulting in a <i>range</i> rather than a <i>single value</i> .
The inputs of modeling, "facts", assumptions, and measurement results, also have stochastic variations and systematic deviations.
Stochastic variations and systematic deviations propagate (add, amplify or cancel) through the model resulting in an output range.

Figure 4.18: Intermezzo Modeling Accuracy

ARM9 200 MHz $t_{\text{context switch}}$
as function of cache use

cache setting	$t_{\text{context switch}}$
From cache	2 μs
After cache flush	10 μs
Cache disabled	50 μs

Figure 4.19: Actual ARM Figures

the importance of the cache for the CPU load. In cache unfriendly situations (a cache flushed context switch) the CPU performance is still a factor 5 better than in the situation with a disabled cache. One reason of this improvement is the locality of instructions. For 8 consecutive instructions "only" 38 cycles are needed to load these 8 words. In case of a disabled cache $8 * (22 + 2 * 1) = 192$ cycles are needed to load the same 8 words.

We did estimate $2\mu\text{s}$ for the context switch time, however already taking into account negative cache effects. The expectation is a factor 5 more optimistic than the measurement. In practice expectations from scratch often deviate a factor from reality, depending on the degree of optimism or conservatism of the estimator. The challenging question is: Do we trust the measurement? If we can provide a credible explanation of the difference, then the credibility of the measurement increases.

In Figure 4.20 some potential missing contributions in the original estimate are presented. The original estimate assumes single cycle instruction fetches, which is not true if the instruction code is not in the instruction cache. The Memory

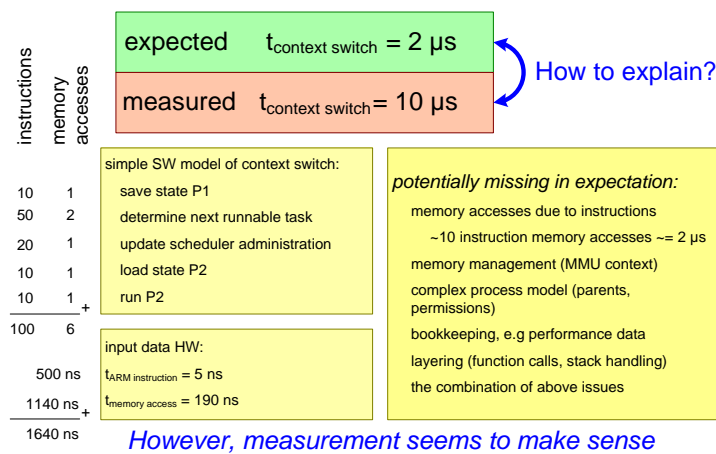


Figure 4.20: Expectation versus Measurement

Management Unit (MMU) might be part of the process context, causing more state information to be saved and restored. Often many small management activities take place in the kernel. For example, the process model might be more complex than assumed, with process hierarchy and permissions. Maybe hierarchy or permissions are accessed for some reasons, maybe some additional state information is saved and restored. Bookkeeping information, for example performance counters, can be maintained. If these activities are decomposed in layers and components, then additional function calls and related stack handling for parameter transfers takes place. Note that all these activities can be present as combination. This combination not only cumulates, but might also multiply.

$$t_{\text{overhead}} = n_{\text{context switch}} * t_{\text{context switch}}$$

$n_{\text{context switch}}$ (s^{-1})	$t_{\text{context switch}} = 10\mu\text{s}$		$t_{\text{context switch}} = 2\mu\text{s}$	
	t_{overhead}	CPU load overhead	t_{overhead}	CPU load overhead
500	5ms	0.5%	1ms	0.1%
5000	50ms	5%	10ms	1%
50000	500ms	50%	100ms	10%

Figure 4.21: Context Switch Overhead

Figure 4.21 integrates the amount of context switching time over time. This figure shows the impact of context switches on system performance for different context switch rates. Both parameters $t_{contextswitch}$ and $n_{contextswitch}$ can easily be measured and are quite indicative for system performance and overhead induced by design choices. The table shows that for the realistic number of $t_{contextswitch} = 10\mu s$ the number of context switches can be ignored with 500 context switches per second, it becomes significant for a rate of 5000 per second, while 50000 context switches per second consumes half of the available CPU power. A design based on the too optimistic $t_{contextswitch} = 2\mu s$ would assess 50000 context switches as significant, but not yet problematic.

4.2.10 Perform sanity check

In the previous subsection the actual measurement result of a single context switch including cache flush was $10\mu s$. Our expected result was in the order of magnitude of $2\mu s$. The difference is significant, but the order of magnitude is comparable. In general this means that we do not completely understand our system nor our measurement. The value is usable, but we should be alert on the fact that our measurement still introduces some additional systematic time. Or the operating system might do more than we are aware of.

One approach that can be taken is to do a completely different measurement and estimation. For instance by measuring the idle time, the remaining CPU time that is available after we have done the real work plus the overhead activities. If we also can measure the time needed for the real work, then we have a different way to estimate the overhead, but now averaged over a longer period.

4.2.11 Summary of measuring Context Switch time on ARM9

We have shown in this example that the goal of measurement of the ARM9 VxWorks combination was to provide guidance for concurrency design and task granularity. For that purpose we need an estimation of context switching overhead.

We provided examples of measurement, where we needed context switch overhead of about 10% accuracy. For this measurement the instrumentation used toggling of a HW pin in combination with small SW test program. We also provided simple models of HW and SW layers to be able to determine an expectation. Finally we found as measurement results for context switching on ARM9 a value of $10\mu s$.

4.3 Summary

Figure 4.22 summarizes the measurement approach and insights.

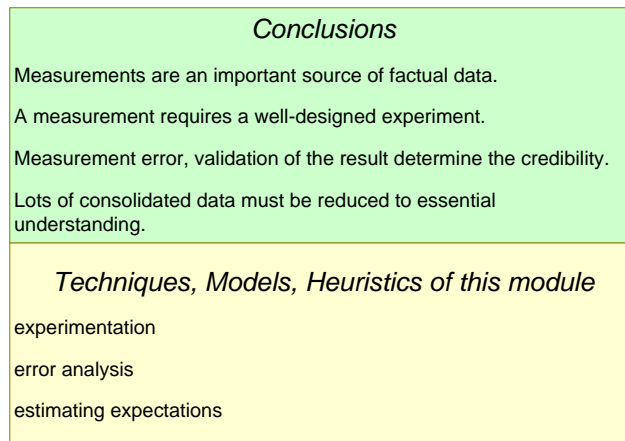


Figure 4.22: Summary Measuring Approach

4.4 Acknowledgements

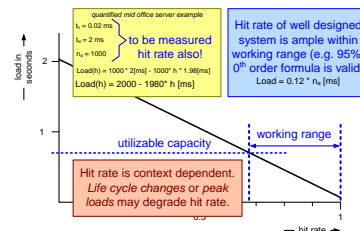
This work is derived from the EXARCH course at CTT developed by Ton Kostelijk (Philips) and Gerrit Muller. The Boderc project contributed to the measurement approach. Especially the work of Peter van den Bosch (Océ), Oana Florescu (TU/e), and Marcel Verhoef (Chess) has been valuable. Teun Hendriks provided feedback, based on teaching the Architecting System Performance course.

Part III

System Model

Chapter 5

Modeling and Analysis: System Model



5.1 Introduction

content
What to model of the system
Stepwise approach to system modeling
Non Functional requirements (NFR), System Properties and Critical Technologies
Examples of web shop case

Figure 5.1: Overview of the content of this paper

Figure 5.1 shows an overview of this paper. We will discuss what to model of the system of interest, see also Figure 5.2. We will provide a stepwise approach to system modeling, based on the relations between Non Functional Requirements (NFR), system design properties and critical technologies. Several examples will be shown using the web shop case.

In our modeling we will focus on the NFR's, such as performance, reliability,

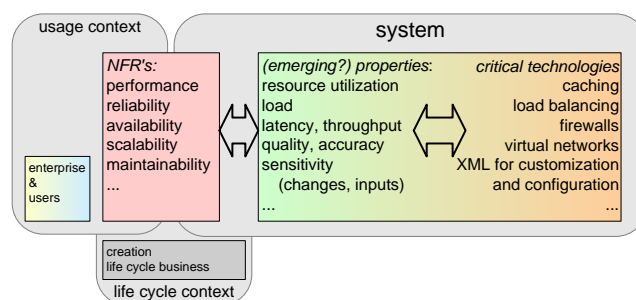


Figure 5.2: What to Model in System Context?

availability, scalability, or maintainability. We assume that the functional requirements and system decomposition are being created in the system architecting and design activity. In practice many NFR's emerge, due to lack of time and attention. We recommend to reduce the risks by modeling and analysis of relevant NFR's where some higher risk is perceived. Figure 5.2 shows that the external visible system characteristics depend on the design of system properties, such as resource utilization, load, latency, throughput, quality, accuracy, or sensitivity for changes or varying inputs. Note that these properties also often emerge, due to lack of time or attention. Not only the design of these properties determine the external visible system characteristics, but also the chosen technologies has a big impact. Therefore we also have to look at critical technologies, for example caching, load balancing, firewalls, virtual networks, or XML for customization and configuration.

5.2 Stepwise approach to system modeling

We recommend an approach where first the system is explored: what is relevant, what is critical? Then the most critical issues are modeled. Figure 5.3 shows a stepwise approach to model a system.

1. **Determine relevant Non Functional Requirements (NFR's)** where the relevance is often determined by the context: the usage context or the life cycle context.
2. **Determine relevant system design properties** by looking either at the NFR's or at the design itself: what are the biggest design concerns?
3. **Determine critical technologies** criticality can have many reasons, such as working close to the working range limit, new components, complex functions with unknown characteristics, sensitivity for changes or environmental conditions, et cetera.
4. **relate NFR's to properties to critical technologies** by making a graph of relations. Such a graph often has many-to-many relations.

1. determine relevant Non Functional Requirements (NFR's)
2. determine relevant system design properties
3. determine critical technologies
4. relate NFR's to properties to critical technologies
5. rank the relations in relevancy and criticality
6. model relations with a high score

Figure 5.3: Approach to System Modeling

5. Rank the relations in relevancy and criticality to find potential modeling candidates.

6. Model relations with a high ranking score a time-boxed activity to build up system understanding.

Note that this system modeling approach fits in the broader approach of modeling and analysis. The broader approach is discussed in Modeling and Analysis: Reasoning.

5.3 Example system modeling of web shop

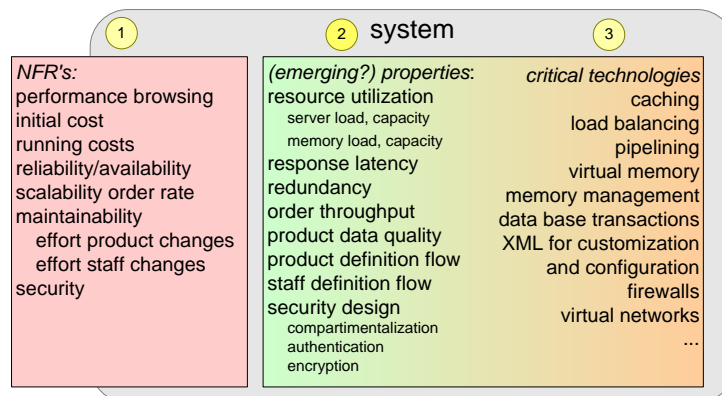


Figure 5.4: Web Shop: NFR's, Properties and Critical Technologies

Figure 5.4 shows the results of step 1, 2, and 3 of the approach.

1. Determine relevant Non Functional Requirements (NFR's) For the web shop the following requirements are crucial: performance browsing, initial cost,

running costs, reliability/availability, scalability order rate, maintainability (effort to enter product changes and effort to enter staff changes) and security.

2. Determine relevant system design properties based on experience and NFR's the following properties were identified as relevant: resource utilization (server load, server capacity, memory load, and memory capacity), response latency, redundancy, order throughput, product data quality, product definition flow, staff definition flow, security design (which can be refined further in compartmentalization, authentication, and encryption). Note that we mention here design issues such as *product definition flow* and *staff definition flow*, which have an direct equivalent in the usage context captured in some of the customer's processes.

3. Determine critical technologies Based on experience and risk assessment the following technologies pop up as potentially being critical: caching, load balancing, pipelining, virtual memory, memory management, data base transactions, XML for customization and configuration, firewalls, and virtual networks.

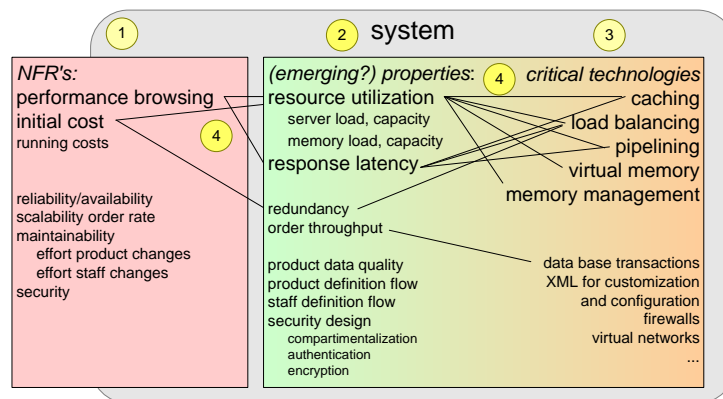


Figure 5.5: 4. Determine Relations

Figure 5.5 shows for a small subset of the identified requirements, properties and technologies the relations. The performance of browsing is related to the resource management design and the concurrency design to meet the response latency. The resource management design relates to several resource specific technologies from caching to memory management. The cost requirements also relate to the resource utilization and to the cost of redundancy measures. The dimensioning of the system depends also on the design of the order throughput. Crucial technology for the order throughput is the data base transaction mechanism and the related performance.

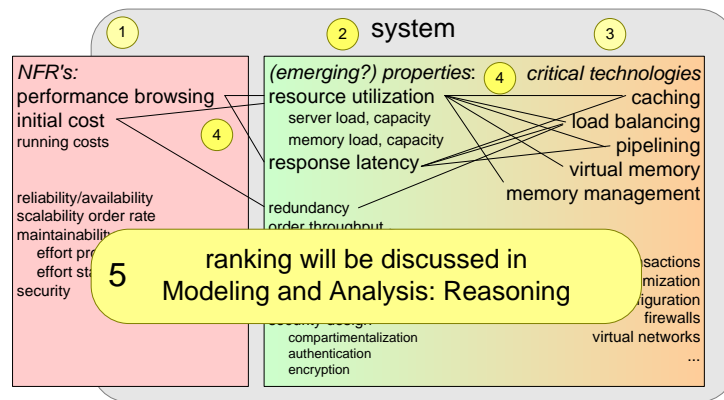


Figure 5.6: 5. Rank Relations

Ranking, Figure 5.6 will be discussed in the *Modeling and Analysis: Reasoning* paper. For this example we will mostly focus on the relations shown in Figure 5.5.

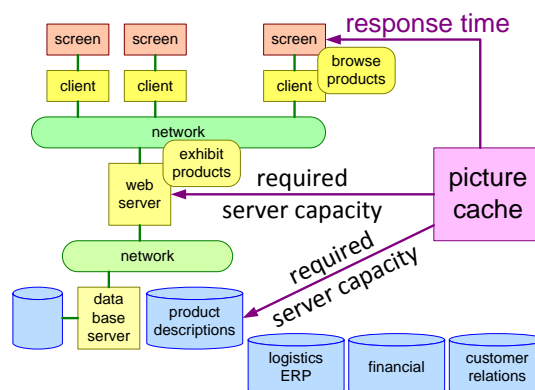


Figure 5.7: Purpose of Picture Cache Model in Web Shop Context

Figure 5.7 shows the picture cache as a specific example of the use of caching technology. The purpose of picture cache is to realize the required performance of product browsing at the client layer. At the web server layer and at the data base layer the picture cache should realize a limited server load of the product exhibition function.

The most simple model we can make for the server load as function of the number of requests is shown in Figure 5.8. This is a so called zero order model, where only the direct parameters are included in the model. It is based on the very simple assumption that the load is proportional with the number of requests.

When we introduce a cache based design, then the server load depends on the

zero order web server load model

$$\text{Load} = n_a * t_a$$

n_a = total requests
 t_a = cost per request

Figure 5.8: Zero Order Load Model

first order web server load model

$$\text{Load} = n_{a,h} * t_h + n_{a,m} * t_m$$

$n_{a,h}$ = accesses with cache hit
 $n_{a,m}$ = accesses with cache miss
 t_h = cost of cache hit
 t_m = cost of cache miss

$$n_{a,h} = n_a * h$$

$$n_{a,m} = n_a * (1-h)$$

n_a = total accesses
 h = hit rate

$$\text{Load}(h) = n_a * h * t_h + n_a * (1-h) * t_m = n_a * t_m - n_a * h * (t_m - t_h)$$

Figure 5.9: First Order Load Model

effectiveness of the cache. Requests that can be served from the cache will have a much smaller server load than requests that have to be fetched from the data base. Figure 5.9 shows a simple first order formula, where the contributions of requests from cache are separated of the requests that need data base access. We introduce an additional parameter h , the hit-rate of the cache. This helps us to create a simple formula where the server load is expressed as a function of the hit-rate.

The simple mathematical formula starts to make sense when we instantiate the formula with actual values. An example of such an instantiation is given in Figure 5.10. In this example we use values for request handling of $t_h = 20\mu s$ and $t_m = 2ms$. For the number of requests we have used $n_a = 1000$, based on the assumption that we are serving the product exhibition function with millions of customers browsing our extensive catalogue. The figure shows the server load as function of the hit-rate. If the available server capacity is known, then we can deduce the minimal required hit-rate to stay within the server capacity. The allowed range of hit-rate values is called the working range.

In Figure 5.11 we zoom in on the hit-rate model. First of all we should realize that we have used assumed values for t_h , t_m and n_a . These assumptions were

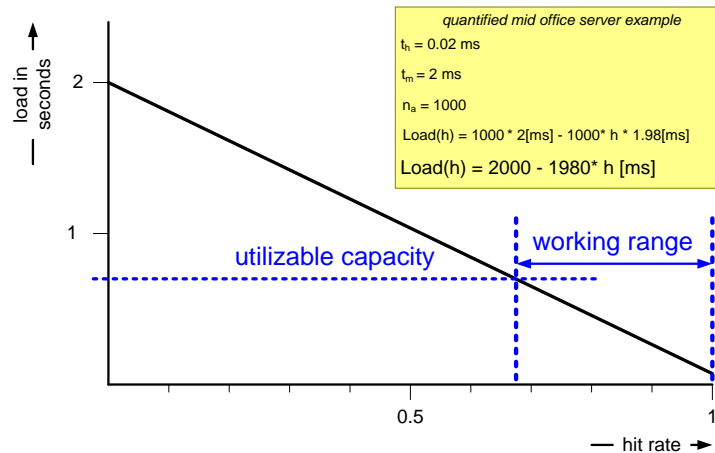


Figure 5.10: Quantification: From Formulas to Insight

based on experience, since we know as experienced designers that transactions cost approximately 1 ms, and that the cache roughly improves the request with a factor 100. However, the credibility of the model increases significantly by measuring these quantities. Common design practice is to design a system well within the working range, for example with a hit-rate of 95% or higher. If the system operates at these high hit-rates, then we can use the zero-order formula for the system load again. Using the same numbers for performance we should then use $t_a \approx 0.95 * t_h + 0.05 * t_m \approx 0.12ms$. Another assumption we have made is that the hit-rate is constant and independent of the circumstances. In practice this is not true. We will, for instance, have varying request rates, perhaps with some high peak values. If the peak values coincide with lower hit rates, then we might expect some nasty performance problems. The hit-rate might also be impacted by future life cycle changes. For example new and different browser functionality might decrease the hit-rate dramatically.

Another system property that was characterized as relevant was the response time design. Response time depends on the degree of concurrency, the synchronization design and the time required for individual operations. Figure 5.12 shows a timing model for the response time, visualizing the above mentioned aspects for the retrieval of a picture in case of a cache miss.

Yet another system design property is the use of resources, such as memory. Figure 5.13 shows the flow of pictures throughout the system, as a first step to address the question how much memory is needed for picture transfers.

In Figure 5.14 we zoom in on the web server to have a look at the memory used for picture transfers. This figure shows a number of alternative design options, ranging from a minimal set of copies in the memory to the more realistic situation

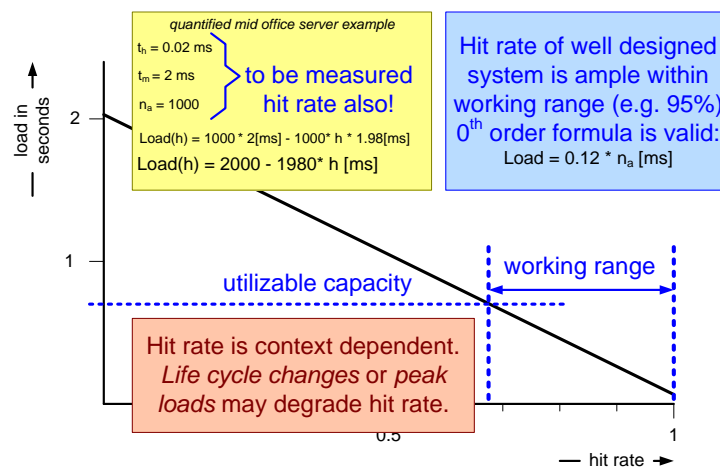


Figure 5.11: Hit Rate Considerations

where every thread contains multiple copies of every picture, while at the same time multiple threads serve concurrent customers.

These alternative design options are transformed in a simple mathematical model in Figure 5.15. This formula is parametrized for the different specification and design parameters:

n = number of data base access threads a design parameter dimensioning the amount of concurrency towards the data base layer.

m = number of picture cache threads a design parameter dimensioning the amount of concurrency of the picture cache itself.

k = number of web server threads a design parameter dimensioning the amount of concurrency of client access to the web server.

s = picture size in bytes an application and design dependent parameter, the average size in bytes of the pictures.

c = in memory cache capacity in number of pictures a design parameter determining the size of a picture cache in number of pictures per picture cache thread.

This formula is instantiated in a quantified table in Figure 5.16, for different values of the design parameters. Note that depending on the chosen design parameters the picture cache maps on completely different storage technologies, with the related different performance characteristics.

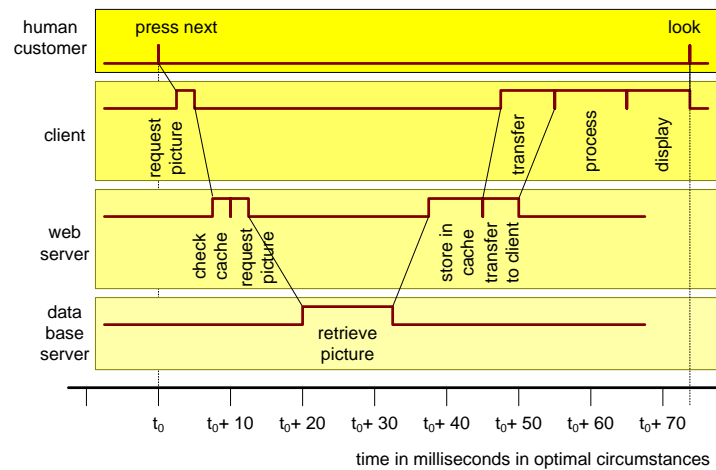


Figure 5.12: Response Time

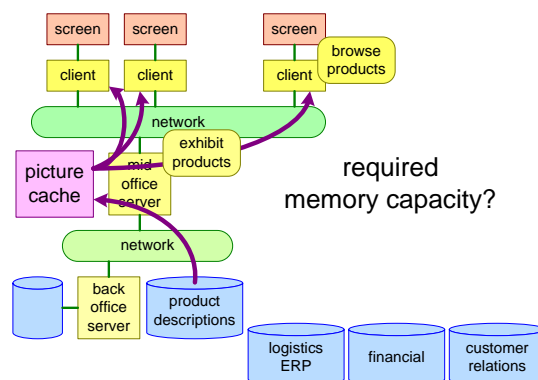


Figure 5.13: What Memory Capacity is Required for Picture Transfers?

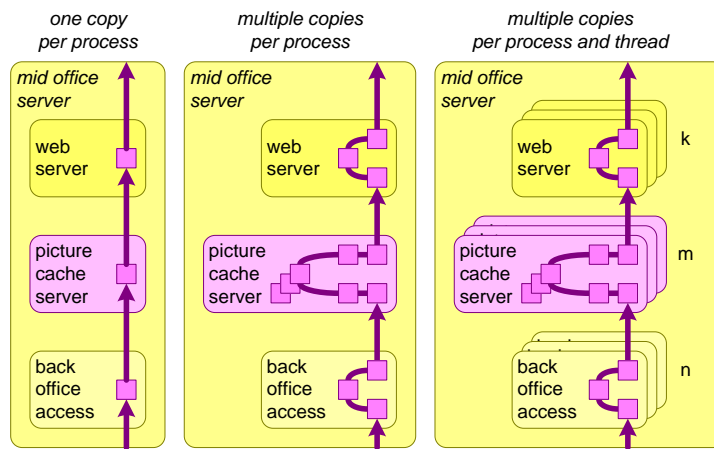


Figure 5.14: Process view of picture flow in web server

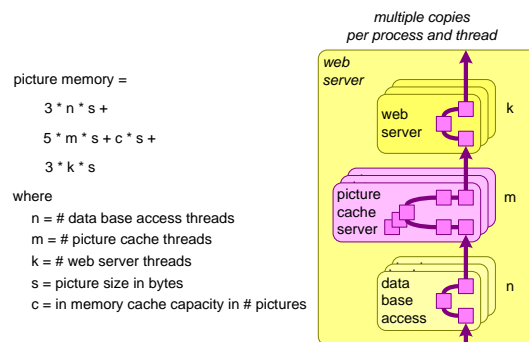


Figure 5.15: Formula memory Use Web Server

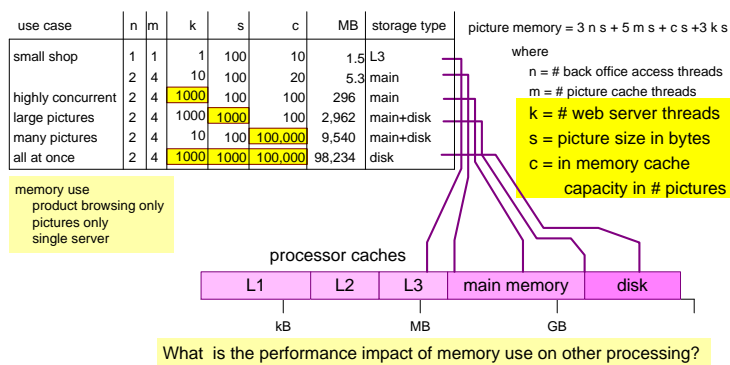


Figure 5.16: Web server memory capacity

5.4 Discussion

In the previous section we have modeled a few parts of the system. Figure 5.17 shows that we have covered so far a small part of the system space. We can describe the system space by several dimensions: functions, data and aspects. Our coverage so far has been limited to the browse and exhibit products function, looking at the pictures as data, looking at the aspects of server memory use, response time and server load.

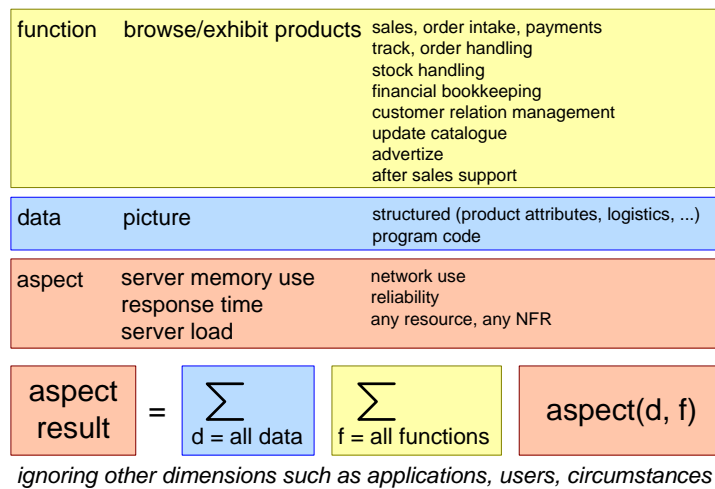


Figure 5.17: Only a small part of the system has been modeled so far

This figure shows many more functions, types of data and aspects present in systems. To answer one of the NFR like questions we have to combine the aspect results of functions and all data types. In practice the context also impacts the NFR's, we have still ignored applications, users, and circumstances.

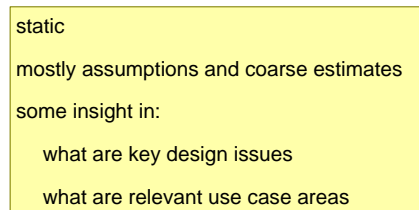


Figure 5.18: The modeling so far has resulted in understand some of the systems aspects

Figure 5.18 adds to this by reminding us that we so far have only made static

models, mostly based on assumptions and coarse estimates. Nevertheless we have obtained some insight in key design issues and relevant use cases.

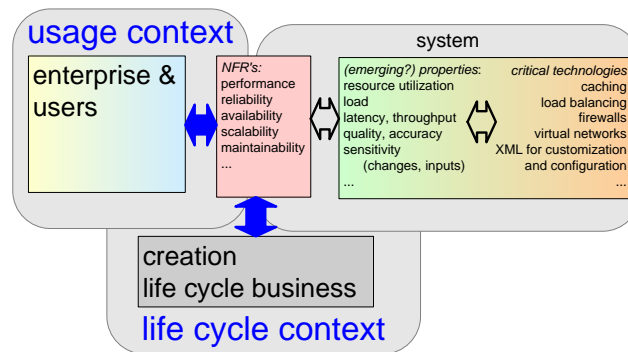


Figure 5.19: Refinement of the system models takes place after context modeling

We are far from finished with system modeling. However, with the results obtained so far it is important to take the next step of the broader iteration: modeling of the contexts, see Figure 5.19. Many of the models we have made of the system trigger questions about the system use and the life cycle. What is the expected amount of browsing, by how many customers, for what size of catalogue? What is the preferred picture quality? How relevant is the maintenance effort related to the product catalogue? et cetera.

5.5 Summary

<i>Conclusions</i>
Non Functional Requirements are the starting point for system modeling
Focus on highest ranking relations between NFR's and critical technologies
Make simple mathematical models
Evaluate quantified instantiations
<i>Techniques, Models, Heuristics of this module</i>
Non functional requirements
System properties
Critical technologies
Graph of relations

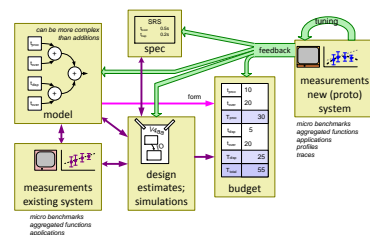
Figure 5.20: Summary of system modeling

Figure 5.20 shows a summary of this paper. We have shown that Non Functional Requirements are the starting point for system modeling. Our approach focuses on the highest ranking relations between NFR's and critical technologies. For these relations we make simple mathematical models that are evaluated by quantified instantiations of these models.

5.6 Acknowledgements

Roelof Hamberg caught several errors in the detailed models

Modeling and Analysis: Budgeting



6.1 Introduction

Budgets are well known from the financial world as a means to balance expenditures and income. The same mechanism can be used in the technical world to balance for instance resource use and system performance.

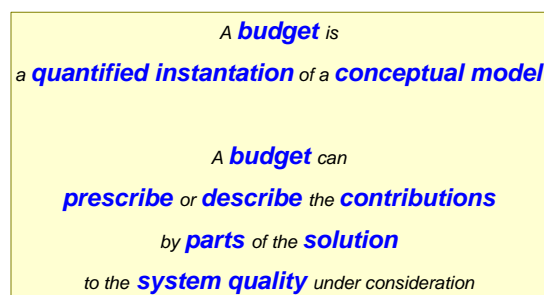


Figure 6.1: Definition of a budget in the technical world

Budgets are more than an arbitrary collection of numbers. The relationship

between the numbers is guided by an underlying model. Figure 6.1 shows *what* a budget is. Technical budgets can be used to provide guidance by prescribing allowed contributions per function or subsystem. Another use of budgets is as a means for understanding, where the budget describes these contributions.

We will provide and illustrate a budget method with the following attributes:

- a goal
- a decomposition in smaller steps
- possible orders of taking these steps
- visualization(s) or representation(s)
- guidelines

6.2 Budget-Based Design method

In this section we illustrate a *budget-based design* method applied at waferstepper, health care, and document handling systems, where it has been applied on different resources: overlay, memory, and power.

6.2.1 Goal of the method

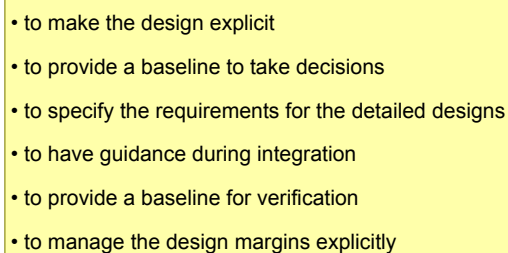
- 
- to make the design explicit
 - to provide a baseline to take decisions
 - to specify the requirements for the detailed designs
 - to have guidance during integration
 - to provide a baseline for verification
 - to manage the design margins explicitly

Figure 6.2: Goals of budget based design

The goal of the budget-based design method is to guide the implementation of a technical system in the use of the most important resource constraints, such as memory size, response time, or positioning accuracy. The budget serves multiple purposes, as shown in Figure 6.2.

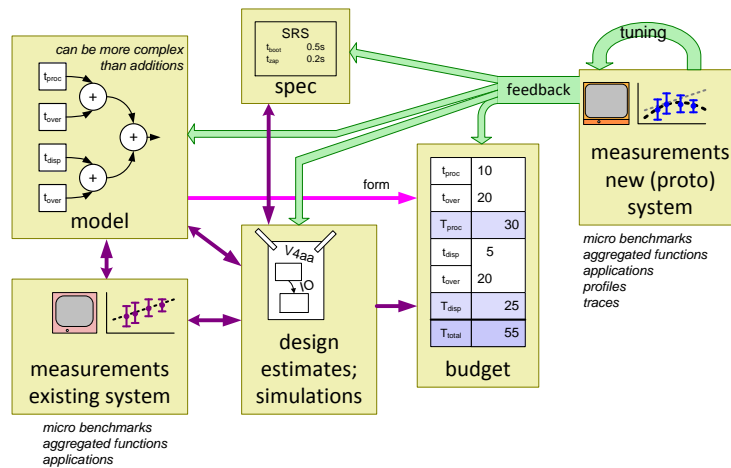


Figure 6.3: Visualization of Budget-Based Design Flow. This example shows a response time budget.

6.2.2 Decomposition into smaller steps

Figure 6.3 visualizes the budget-based design flow. This visualization makes it clear that although the budget plays a central role in this design flow, cooperation with other methods is essential. In this figure other cooperating methods are performance modeling, micro-benchmarking, measurement of aggregated functions, measurements at system level, design estimates, simulations, and requirements specification.

Measurements of all kinds are needed to provide substance to the budget. Micro-benchmarks are measurements of elementary component characteristics. The measured values of the micro-benchmarks can be used for a bottom-up budget. Measurements at functional level provide information at a higher aggregated level; many components have to cooperate actively to perform a function. The outcome of these function measurements can be used to verify a bottom-up budget or can be used as input for the system level budget. Measurements in the early phases of the system integration are required to obtain feedback once the budget has been made. This feedback will result in design changes and could even result in specification changes. The use of budgets can help to set up an integration plan. The measurement of budget contributions should be done as early as possible, because the measurements often trigger design changes.

6.2.3 Possible order of steps

Figure 6.4 shows a budget-based design flow (the *order* of the method). The starting point of a budget is a model of the system, from the conceptual view.

step	example
1A measure old systems	micro-benchmarks, aggregated functions, applications
1B model the performance starting with old systems	flow model and analytical model
1C determine requirements for new system	response time or throughput
2 make a design for the new system	explore design space, estimate and simulate
3 make a budget for the new system:	models provide the structure measurements and estimates provide initial numbers specification provides bottom line
4 measure prototypes and new system	micro-benchmarks, aggregated functions, applications profiles, traces
5 Iterate steps 1B to 4	

Figure 6.4: Budget-based design steps

An existing system is used to get a first guidance to fill the budget. In general the budget of a new system is equal to the budget of the old system, with a number of explicit improvements. The improvements must be substantiated with design estimates and simulations of the new design. Of course the new budget must fulfill the specification of the new system; sufficient improvements must be designed to achieve the required improvement.

6.2.4 Visualization

In the following three examples different actually used *visualizations* are shown. These three examples show that a multi-domain method does not have to provide a single solution, often several useful options exist. The method description should provide some guidance in choosing a visualization.

6.2.5 Guidelines

A *decomposition* is the foundation of a budget. No universal recipe exists for the decomposition direction. The construction decomposition and the functional decomposition are frequently used for this purpose. Budgets are often used as part of the design specification. From project management viewpoint a decomposition is preferred that maps easily on the organization.

The architect must ensure the *manageability* of the budgets. A good budget has tens of quantities described. The danger of having a more detailed budget is loss of overview.

The simplification of the design into budgets introduces design constraints. Simple budgets are entirely static. If such a simplification is too constraining or too

costly then a dynamic budget can be made. A dynamic budget uses situationally determined data to describe the budget in that situation. For instance, the amount of memory used in the system may vary widely depending on the function or the mode of the system. The budget in such a case can be made mode-dependent.

6.2.6 Example of overlay budget for wafersteppers

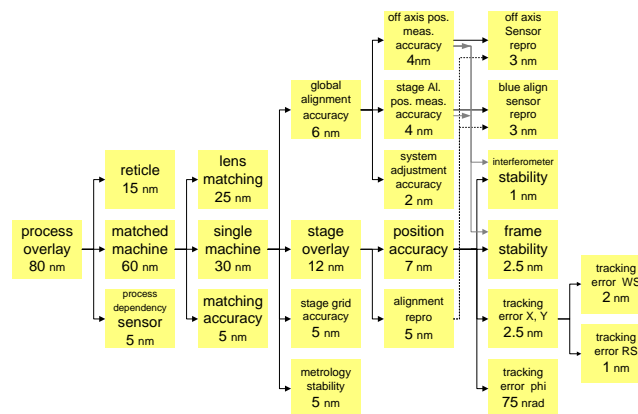


Figure 6.5: Example of a quantified understanding of overlay in a waferstepper

Figure 6.5 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.

The *goal* of the overlay budget is:

- to provide requirements for subsystems and components.
- to enable measurements of the actual contributions to the overlay during the design and integration process, on functional models or prototypes.
- to get early feedback of the overlay design by measurements.

The *steps* taken in the creation, use and validation of the budget follow the description of Figure 6.4. This budget is based on a model of the overlay functionality in the waferstepper (step 1B). The system engineers made an explicit model of the overlay. This explicit model captures the way in which the contributions accumulate: quadratic summation for purely stochastic, linear addition for systematic effects and some weighted addition for mixed effects. The waferstepper budget is created by measuring the contributions in an existing system (step 1A). At the same time a top-down budget is made, because the new generation of machines needs

a much better overlay specification than the old generation (step 1C). In discussions with the subsystem engineers, design alternatives are discussed to achieve the required improvements (step 2 and 3). The system engineers also strive for measurable contributions. The measurability of contributions influences the subsystem specifications. If needed the budget or the design is changed on the basis of this feedback (step 4).

Two *visualizations* were used for the overlay budget: tables and graphs, as shown in Figure 6.5.

The overlay budget plays a crucial role in the development of wafersteppers. The interaction between the system and the customer environment is taken into account in the budget. However, many open issues remain at this interface level, because the customer environment is outside the scope of control and a lot of customer information is highly confidential. The translation of this system level budget into mono-disciplinary design decisions is still a completely human activity with lots of interaction between system engineers and mono-disciplinary engineers.

6.2.7 Example of memory budget for Medical Imaging Workstation

The *goal* of the memory budget for the medical imaging workstation is to obtain predictable and acceptable system performance within the resource constraints dictated by the cost requirements. The *steps* taken to create the budget follow the order as described in Figure 6.4. The *visualization* was table based.

<i>memory budget in Mbytes</i>	code	obj data	bulk data	total
shared code	11.0			11.0
User Interface process	0.3	3.0	12.0	15.3
database server	0.3	3.2	3.0	6.5
print server	0.3	1.2	9.0	10.5
optical storage server	0.3	2.0	1.0	3.3
communication server	0.3	2.0	4.0	6.3
UNIX commands	0.3	0.2	0	0.5
compute server	0.3	0.5	6.0	6.8
system monitor	0.3	0.5	0	0.8
application SW total	13.4	12.6	35.0	61.0
UNIX Solaris 2.x				10.0
file cache				3.0
total				74.0

Figure 6.6: Example of a memory budget

The rationale behind the budget can be used to derive *guidelines* for the creation of memory budgets. Figure 6.6 shows an example of an actual memory budget for a medical imaging workstation from Philips Medical Systems. This budget decomposes the memory into three different types of memory use: code ("read only" memory with the program), object data (all small data allocations for control and

bookkeeping purposes) and bulk data (large data sets, such as images, which is explicitly managed to fit the allocated amount and to prevent memory fragmentation). The difference in behavior of these three memory types is an important reason to separate into different budget entries. The operating system and the system infrastructure, at the other hand, provide means to measure these three types at any moment, which helps for the initial definition, for the integration, and for the verification.

The second decomposition direction is the *process*. The number of processes is manageable, since processes are related to specific development teams. Also in this case the operating system and system infrastructure support measurement at process level.

The memory budget played a crucial role in the development of this workstation. The translation of this system level budget into mono-disciplinary design decisions was, as in the case of overlay in wafersteppers, a purely human activity. The software discipline likes to abstract away from physical constraints, such as memory consumption and time. A lot of room for improvement exists at this interface between system level design and mono-disciplinary design.

6.2.8 Example of power budget visualizations in document handling

Visualizations of a budget can help to share the design issues with a large multi-disciplinary team. The tables and graphs, as shown in the previous subsections, and as used in actual practice, contain all the information about the resource use. However the *hot spots* are not emphasized. The visualization does not help to see the contributions in perspective. Some mental activity by the reader of the table or figure is needed to identify the design issues.

Figure 6.7 shows a visualization where at the top the physical layout is shown and at the bottom the same layout is used, however the size of all units is scaled with the allocated power contribution. The bottom visualization shows the *power foot print* of the document handler units.

Figure 6.8 shows an alternative power visualization. In this visualization the energy transformation is shown: incoming electrical power is in different ways transformed into heat. The width of the arrows is proportional to the amount of energy. This visualization shows two aspects at the same time: required electrical power and required heat disposition capacity, two sides of the same coin.

6.2.9 Evolution of budget over time

Figure 6.9 shows a classification for budget types. It will be clear that already with four different attributes the amount of different types of budgets is large. Every type of budget might have its own peculiarities that have to be covered by the method. For instance, worst case budgets need some kind of over-kill prevention.

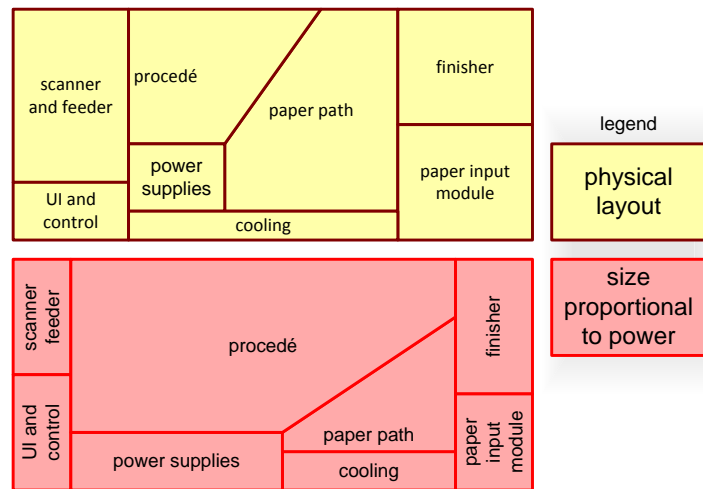


Figure 6.7: Power Budget Visualization for Document Handler

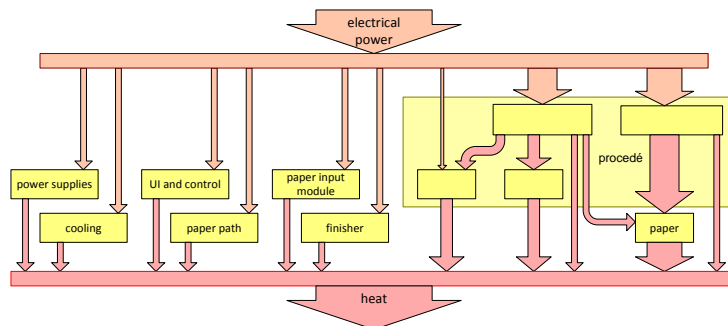


Figure 6.8: Alternative Power Visualization

Add to these different types the potential different purposes of the budget (design space exploration, design guidance, design verification, or quality assurance) and the amount of method variations explodes even more.

We recommend to start with a budget as simple as possible:

- coarse guesstimate values
- typical case
- static, steady state system conditions
- derived from existing systems

This is also shown in Figure 6.10. This figure adds the later evolutionary increments, such as increased accuracy, more attention for boundary conditions and

static	dynamic
typical case	worst case
global	detailed
approximate	accurate

is the budget based on
wish, empirical data, extrapolation,
educated guess, or expectation?

Figure 6.9: What kind of budget is required?

dynamic behavior.

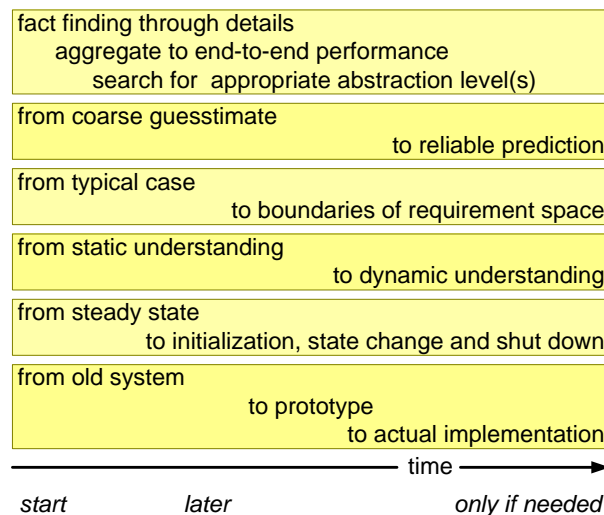


Figure 6.10: Evolution of Budget over Time

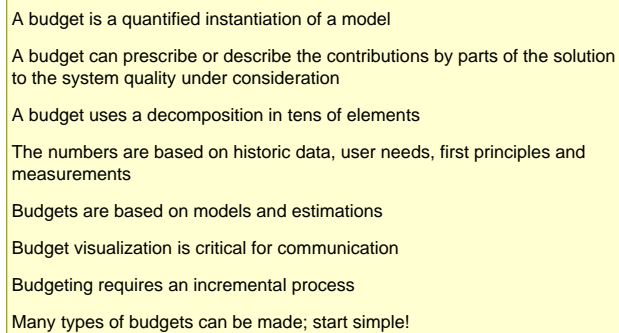
However, some fact finding has to take place before making the budget, where lots of details can not be avoided. Facts can be detailed technical data (memory access speed, context switch time) or at customer requirement level (response time for specific functions). The challenge is to mold these facts into information at the *appropriate* abstraction level. Too much detail causes lack of overview and understanding, too little detail may render the budget unusable.

6.2.10 Potential applications of budget method

For instance the following list shows potential applications, but this list can be extended much more. At the same time the question arises whether *budget-based design* is really the right submethod for these applications.

- resource use (CPU, memory, disk, bus, network)
- timing (response time, latency, start up, shutdown)
- productivity (throughput, reliability)
- image quality (contrast, signal to noise ratio, deformation, overlay, depth-of-focus)
- cost, space, time, effort (for instance expressed in lines of code)

6.3 Summary



A budget is a quantified instantiation of a model

A budget can prescribe or describe the contributions by parts of the solution to the system quality under consideration

A budget uses a decomposition in tens of elements

The numbers are based on historic data, user needs, first principles and measurements

Budgets are based on models and estimations

Budget visualization is critical for communication

Budgeting requires an incremental process

Many types of budgets can be made; start simple!

Figure 6.11: Summary of budget based design

6.4 Acknowledgements

The Boderc project contributed to the budget based design method. Figure 6.12 shows the main contributors.

The Boderc project contributed to Budget Based Design. Especially the work of *Hennie Freriks, Peter van den Bosch (Océ), Heico Sandee and Maurice Heemels (TU/e, ESI)* has been valuable.

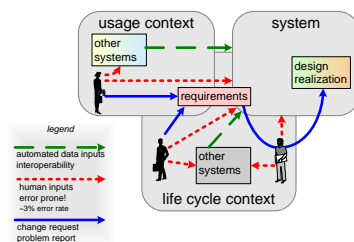
Figure 6.12: Colophon

Part IV

Application and Life Cycle Model

Chapter 7

Modeling and Analysis: Life Cycle Models



7.1 Introduction

Life cycle modeling is mostly modeling expected changes during the life cycle and the impact of these changes. We will provide an approach to make life cycle models. This approach is illustrated by means of a web shop example.

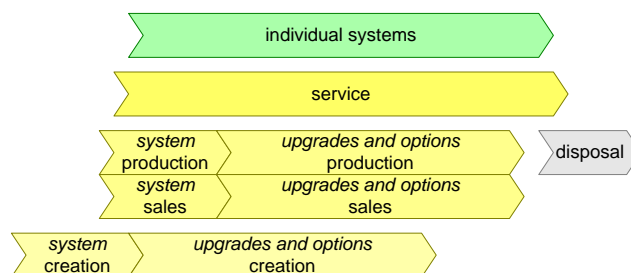


Figure 7.1: Product Related Life Cycles

Several life cycles are relevant. Figure 7.1 shows the related life cycles of

product, option and upgrade *creation*, *production* and *sales*, the systems themselves, and the disposition.

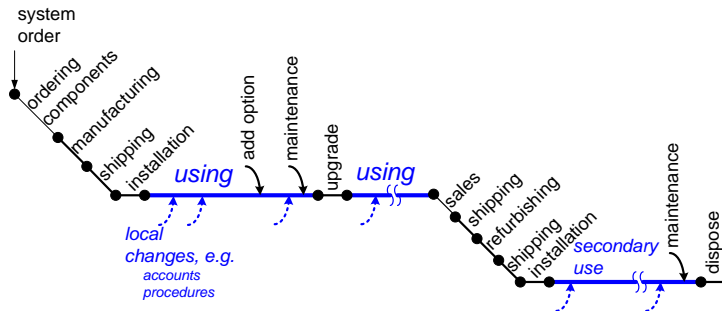


Figure 7.2: System Life Cycle

Figure 7.2 zooms in on the life cycle of individual system instances. Components are ordered and assembled into systems. The real use of the system starts after the system has been shipped and installed. During the use of the system many things happen to the system. The users themselves make small changes, such as adding or updating user accounts or procedures. Options can be added to the system and the system is maintained by service. Some systems are refurbished when they get older to be used again at some different location. Finally the system has to be disposed.

This paper belongs to the modeling and analysis series. It uses the same case example and overall approach.

7.2 Life Cycle Modeling Approach

Identify potential life cycle changes and sources	
Characterize time aspect of changes	how often how fast
Determine required effort	amount type
Determine impact of change on system and context	performance reliability
Analyse risks	business

see reasoning

Figure 7.3: Approach to Life Cycle Modeling

Figure 7.3 shows a step-wise approach to make life-cycle models. The following steps are performed:

Identify potential life cycle changes and sources

Characterize time aspect of changes How often do these changes occur, how fast must be responded to these changes?

Determine required effort , the amount and the type of effort.

Determine impact of change on system and context for instance by using qualities, such as performance, reliability, security, cost, et cetera.

Analyse risks for the business. For instance, what is the cost of being several hours or days too late?

The impact and risks analysis is not elaborated in this paper, see *reasoning* and *analysis* papers.

business volume	www.homes4sale.com
product mix	www.apple.com/itunes/
product portfolio	www.amazon.com
product attributes (e.g. price)	www.ebay.com
customers	www.shell.com
personnel	www.stevens.edu
suppliers	www.nokia.com
application, business processes	stock market
et cetera	insurance company
	local Dutch cheese shop

Figure 7.4: What May Change During the Life Cycle?

During the life cycle, many elements may change, for example business volume, product mix, product portfolio, see Figure 7.4 for more examples. The amount of changes depends strongly on the type of business. For example a real estate portal is selling unique products with a lot attribute data per product. A music web shop, such as iTunes, at the other hand is selling the same product many many times. Figure 7.4 shows more variations of web sites.

The source of a data change influences the impact of such a change. A fundamental difference is data input from automated sources, such as data bases of content providers, versus data input by humans. Human inputs are very error prone. About 3 out of 100 human actions are erroneous. Luckily humans are also very flexible, so many errors are repaired immediately. Nevertheless, many errors in the human inputs slip through and enter the system. The amount of errors

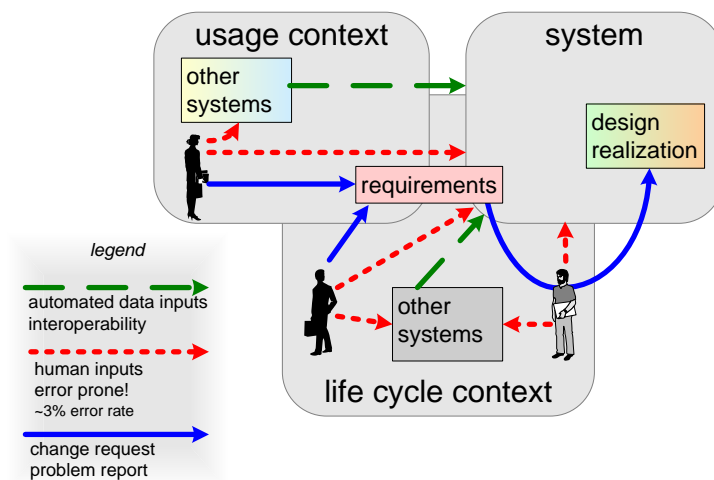


Figure 7.5: Simple Model of Data Sources of Changes

in automated inputs depends on the data quality of the source and on the degree of interoperability (“level of mutual understanding”) between the providing and receiving systems.

Figure 7.5 shows possible sources of changes from the usage context and life cycle context. Note that several human stakeholders can also generate problem reports or change requests, resulting ultimately in changes in of the design and realization of the system. Typically the response on problem reports must be fast (days or weeks), while the change request response is normally much slower (months or year). These response times are also a function of the business. For example in the world of the entertainment industry, where television shows use cell phone for interactive inputs, may suddenly require change request response times of days or weeks, rather than months.

Figure 7.6 zooms in one step further on changes that impact the web server of the web shop example. The changes in content are prepared outside of the production system. Most content changes will be provided by different content providers. For example publishers will provide most new books and related attributes for book shops. Human interaction will be limited to selection and the addition of sales information. Nevertheless we should be aware that even the automated input has its quality limits and originates somewhere from humans. Two other sources of changes are configuration related:

the shop configuration , for example roles, accountabilities and responsibilities of the staff

the system configuration , for example what servers are used, how are functions and resources allocated.

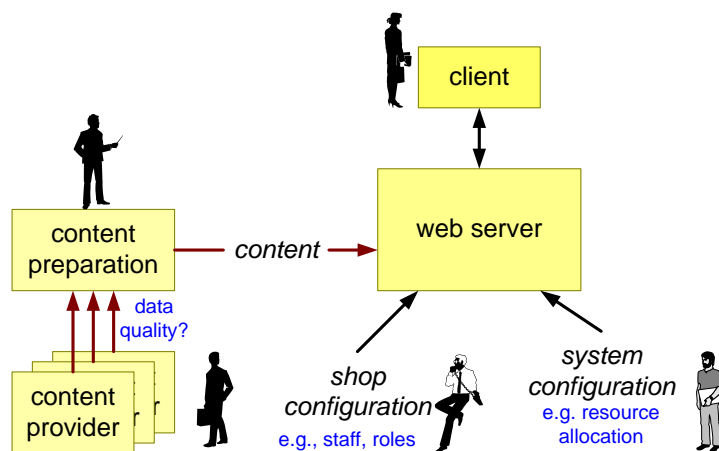


Figure 7.6: Data Sources of Web Server

We have observed that configuration changes are a frequent source of reliability and availability problems. In terms of the popular press this called a computer or software problem. The last source of change in this figure is the behavior of the customers. A sudden hype or fashion may cause a very specific load on the system.

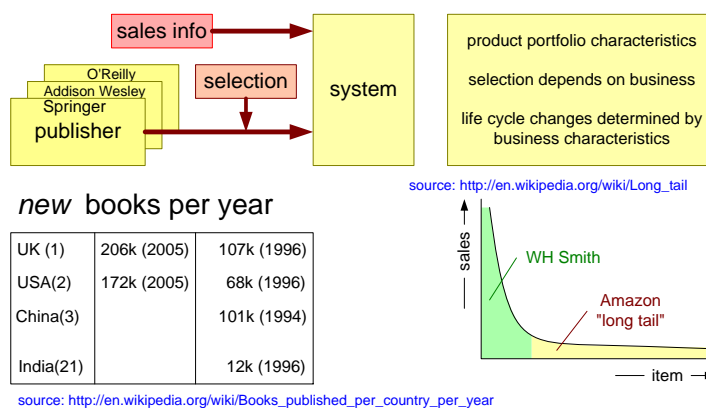


Figure 7.7: Example Product Portfolio Change Books

The modeling becomes much more concrete if we are able to quantify the number of changes. Figure 7.7 shows as an example the quantification of the number of books that is published per year. This data, from wikipedia, shows that UK and USA both publish more than 100k new books per year, together these two countries publish more than 1000 new books per day! The same data source provides data for many different countries. This illustrates the geographic impact

on the quantification. India is still a small producer of books, but with more inhabitants than the UK and USA together, it can be expected that this will increase significantly. Note that this short discussion about India is a discussion of a second order effect: the change in the rate of change.

Wikipedia also provides data on the sales frequency of books. The interesting notion of the *long tail* is explained. In the case of book sales the total volume of very popular books is smaller than the long tail of many books with small individual sales. The characteristics of a book shop of popular books is entirely different from a book shop selling a wide variety of less popular books.

internet: broadband penetration

	Q1 '04	Q2 '04	growth in Q2 '04
Asia Pacific total	48M	54M	12.8%
China	15M	19M	26.1%
India	87k	189k	116.8%

http://www.apira.org/download/world_broadband_statistics_q2_2004.pdf

What is the expected growth of # customers?
 What is the impact on system and infrastructure?
 What is the impact on CRM (Customer Relation Management)?
 What is the impact on customer, sales support staff?

Figure 7.8: Example Customer Change

Figure 7.8 provides numbers related to the potential change in the customer base. This figure shows the number of broadband connections in China and India. If people connected to broadband are most probable customers of a web shop, then these numbers provide an indication of a potential shift in the customer base. Note that the amount of broadband connections in China increases with 25% per quarter, while this increase is more than 100% in India. The current situation is that very few Indian people are potential web shop customers, but this number doubles every quarter! Note that the sales volume of the web shop is not only determined by the customer potential. Also market share growth and extension of the product portfolio will increase sales volume and customer base. A web shop in India might start very small and low cost, but it might have to scale up very rapidly!

The growth in the number of customers will trigger other changes:

What is the impact on system and infrastructure? The dimensions of the system have to be adapted to the changed load. In scaling thresholds occur where

a more fundamental change is triggered. For example from a single multi-purpose server to several dedicated servers.

What is the impact on CRM (Customer Relation Management)? This might be a trivial function for a few thousand customers, but with tens or hundreds of thousands of customers more support might be necessary.

What is the impact on customer, sales support staff? More customers often has as a consequence that more staff is required: more customer support and more sales managers. An increase in staffing may also trigger changes in the system itself.

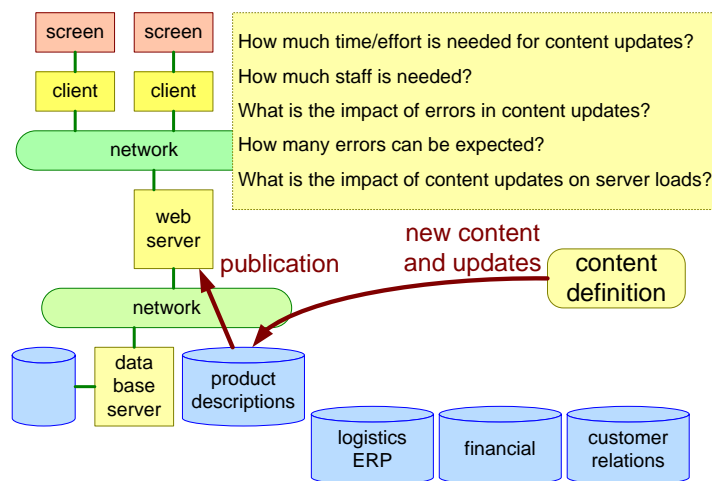


Figure 7.9: Web Shop Content Update

Once changes are identified we can analyze the propagation of these changes, as shown for the customer base. Changes trigger new changes. Figure 7.9 formulates a number of questions to look at this ripple through effect:

How much time/effort is needed for content updates? see below for elaboration.

How much staff is needed? And how many staff and role changes are to be expected?

What is the impact of errors in content updates? So what is the impact on system quality and reliability? What is the process to prevent and cope with errors?

How many errors can be expected? Make the problem more tangible.

What is the impact of content updates on server loads? Do we have to scale the server configuration, due to changes in the content updates?

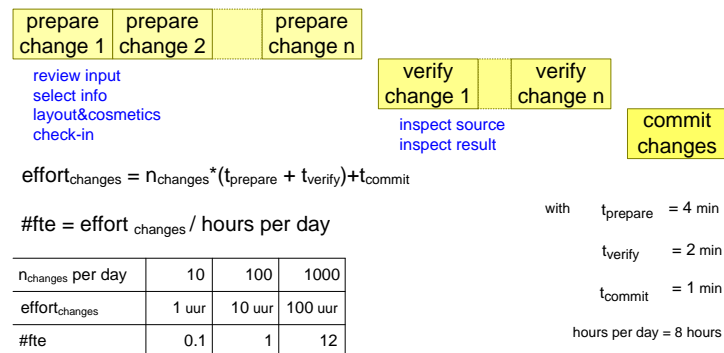


Figure 7.10: Web Shop Content Change Effort

We need a simple model of the update process to estimate the amount of effort to change the content. Figure 7.10 provides such a simple model:

- every change is a sequence of 4 steps:
 - review input
 - select information to be used
 - design layout and apply cosmetics or “styling”
 - check in of the change, an administrative step

Automated checks will take place concurrently with these steps, ensuring syntactically correct input.

- every change is verified by inspection: the implementation and the result are inspected.
- the complete set of changes is committed.

This simple process model can be used to make an effort model. If we substitute numbers in the formula derived in Figure 7.10, then we can explore the impact of the number of changes on effort and staff size.

The business context, the application, the product and its components have all their own specific life-cycles. In Figure 7.11 several different life-cycles are shown. The application and business context in the customer world are shown at the top of the figure, and at the bottom the technology life-cycles are shown. Note that the time-axis is exponential; the life-cycles range from one month to more than ten years! Note also the tension between commodity software and hardware life-cycles and software release life-cycles: How to cope with fast changing commodities? And how to cope with long living products, such as MR scanners, that use commodity technologies?

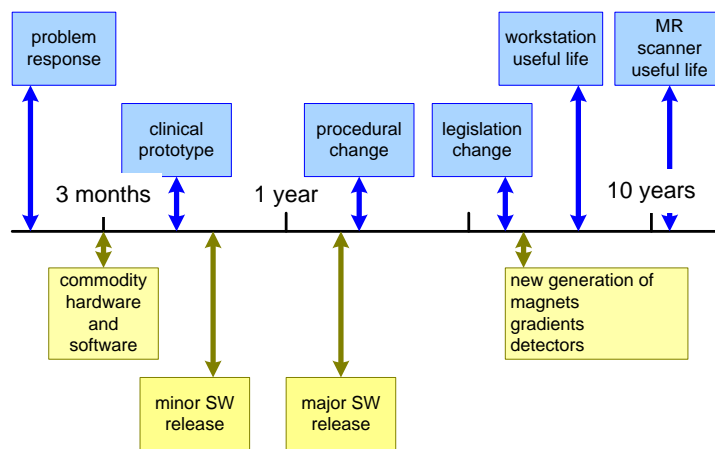


Figure 7.11: Life-cycle Differences for health care equipment

Note that the web shop content life cycle may be shorter than one month in the health care equipment example. Content life cycles may be one day or even shorter.

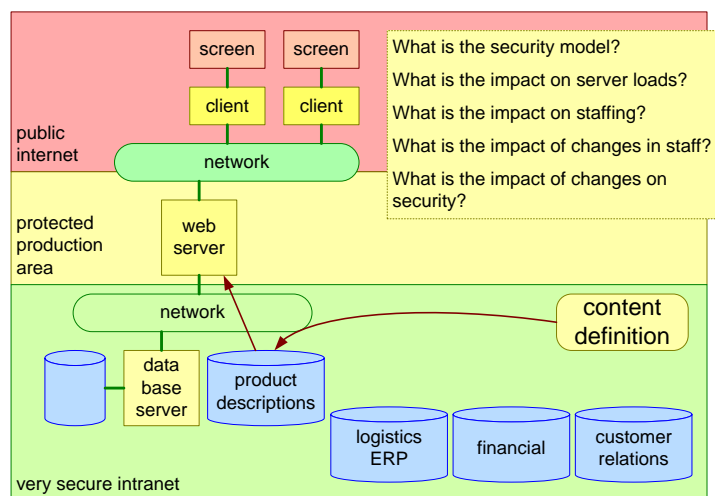


Figure 7.12: Web Shop Security and Changes

One way of modeling and analyzing the consequences of changes is by following the qualities. As an example, Figure 7.12 zooms in on the security aspect of the web shop example. The following questions can be analyzed:

What is the security model? In the diagram it is shown that different security domains are used:

- public internet where the clients are connected
- the production domain with enhanced access control through gateways. The external world has limited access to interact with the production environment of the web shop.
- a very secure intranet environment, where web shop content preparation and web shop management takes place.

What is the impact on server loads? The layers of security and the security based allocation of functions and data may impact the load of the servers.

What is the impact on staffing? The processes to ensure security will have impact on the way-of-working of the staff and the amount of work.

What is the impact of changes in staff? The staff itself has a significant impact on overall security. Changes of the staff itself will trigger second order effects, such as screening and authorization work and blocking moved staff.

What is the impact of changes on security? Any change somewhere in the system might have a side effect on overall security. Security concerns will create a continuous overhead for systems and staff.

new faults = average fault density * #changes

$$\#errors = \sum_{\text{faults}} f(\text{severity, hit probability, detection probability})$$

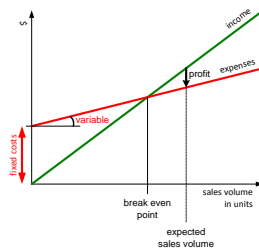
	severity	hit probability	detection probability
<i>Jansen iso</i>	low	high	low
<i>Janssen</i>			
<i>operator iso</i>	high	high	medium
<i>sales repr</i>			

Figure 7.13: Web Shop Reliability and Changes

Figure 7.13 shows an example of reliability modeling. *this needs to be elaborated GM.*

Chapter 8

Simplistic Financial Computations for System Architects.



8.1 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 8.2 a very basic cost and margin model is described. Section 8.3 refines the model at the cost side and the income side. In Section 8.4 the time dimension is added to the model. Section 8.5 provides a number of criteria for making financial decisions.

8.2 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 8.1 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.

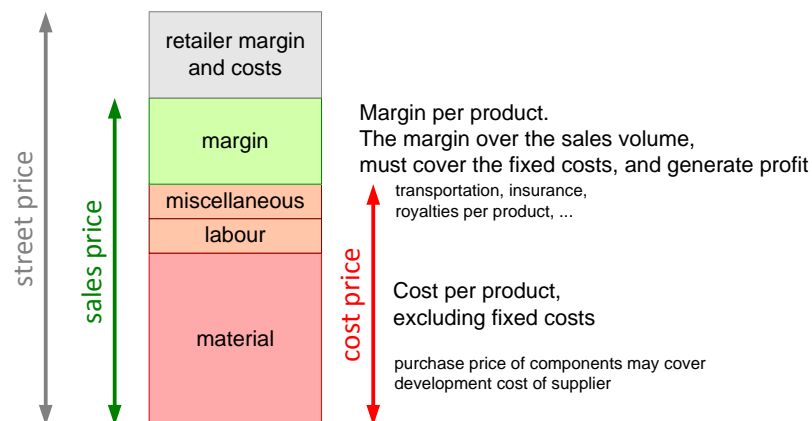


Figure 8.1: The relation between sales price, cost price and margin per product

The translation of margin into profit can be done by plotting income and expenses in one figure, as shown in Figure 8.2, 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.

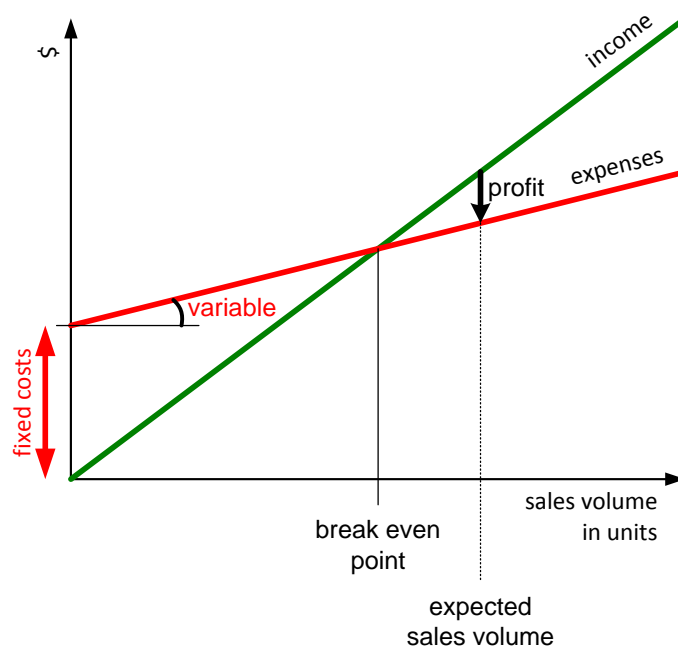


Figure 8.2: Profit as function of sales volume

8.3 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 8.3 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:

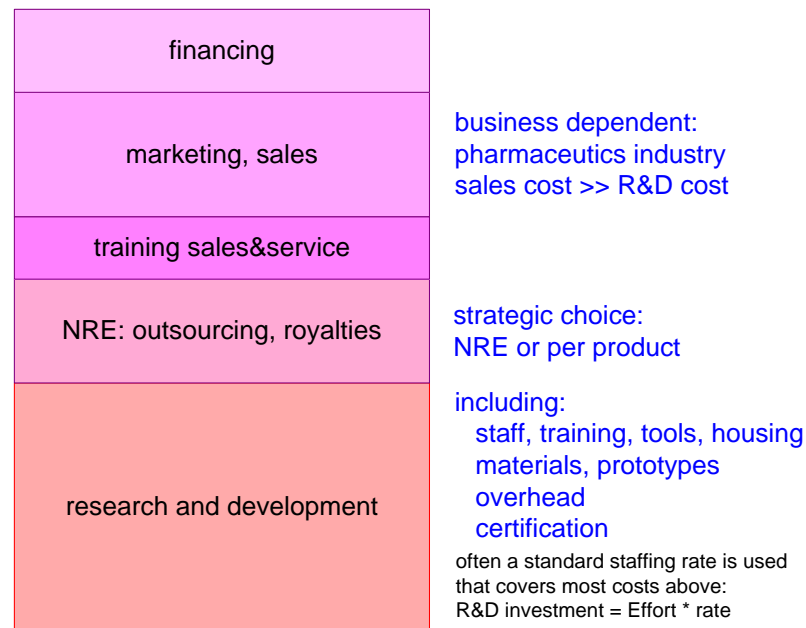


Figure 8.3: Investments, more than R&D

- the supplier takes a risk by making the investments, but also benefits from 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 8.4. The options and accessories are sold as separate entities, generating a significant revenue for many products. For many products the base products are

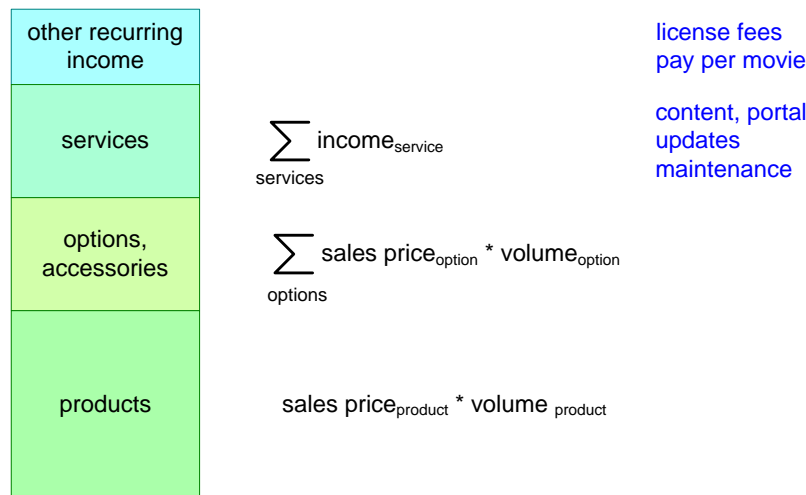


Figure 8.4: Income, more than product sales only

sold with a loss. This loss is later compensated by the profit on options and accessories.

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 prinjet 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.

8.4 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 8.5. 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:

$$\text{variable costs} = \text{sales volume} * \text{cost price}$$

$$\text{income} = \text{sales volume} * \text{sales price}$$

	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 8.5: The Time Dimension

$$profit = income - (investments + variable costs)$$

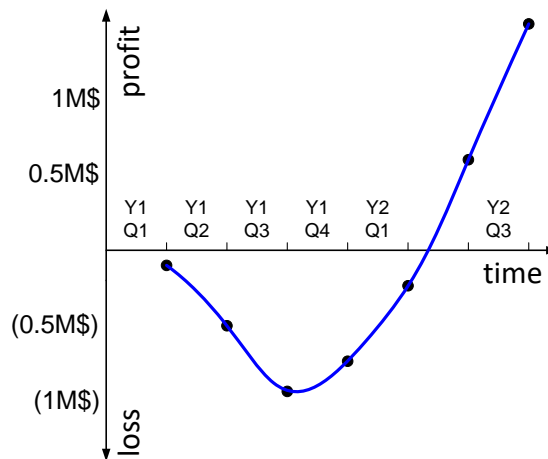


Figure 8.6: The “Hockey” Stick

Figure 8.6 shows the cumulative profit from Figure 8.5 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?

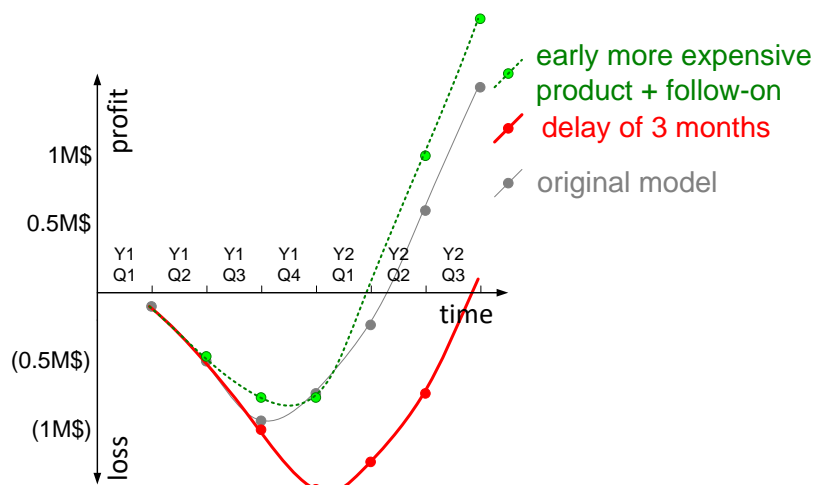


Figure 8.7: What if ...?

These questions can also be refined by performing a simple sensitivity analysis. Figure 8.7 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 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.

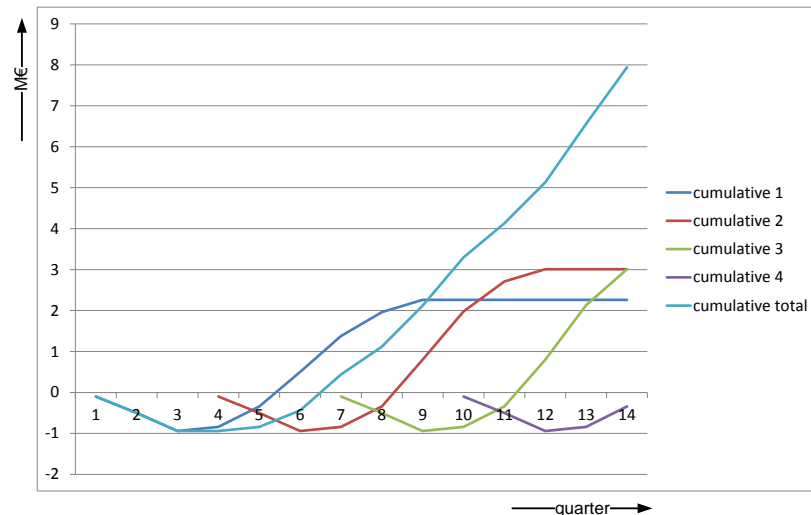


Figure 8.8: Stacking Multiple Developments

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 repeated for the next products. The sales volume for the later products is increasing gradually.

8.5 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 8.9 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.

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 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 8.9: Fashionable financial yardsticks

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 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

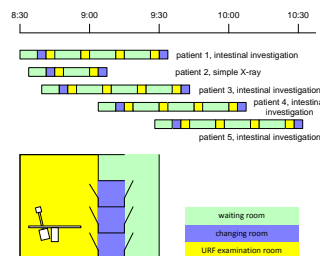
negative cash-flow. The crisis of Philips in 1992 was caused by this effect: years of profit combined with a negative cash-flow.

8.6 Acknowledgements

William van der Sterren provided feedback and references. Hans Barella, former CEO of Philips medical Systems, always stressed the importance of Figure 8.2, and especially the importance of a robust profit. Ad van den Langenberg pointed out a number of spelling errors.

Chapter 9

The application view



9.1 Introduction

The application view is used to understand how the customer is achieving his objectives. The methods and models used in the application view should discuss the customer's world. Figure 9.1 shows an overview of the methods discussed here.

The customer is a gross generalization, which can be made more specific by identifying the customer stakeholders and their concerns, see section 9.2.

The customer is operating in a wider world, which he only partially controls. A context diagram shows the context of the customer, see section 9.3. Note that part of this context may interface actively with the product, while most of this context simply exists as neighboring entities. The fact that no interface exists is no reason not to take these entities into account, for instance to prevent unwanted duplication of functionality.

The customer domain can be modelled in static and dynamic models. Entity relationship models (section 9.4) show a static view on the domain, which can be complemented by dynamic models (section 9.5).

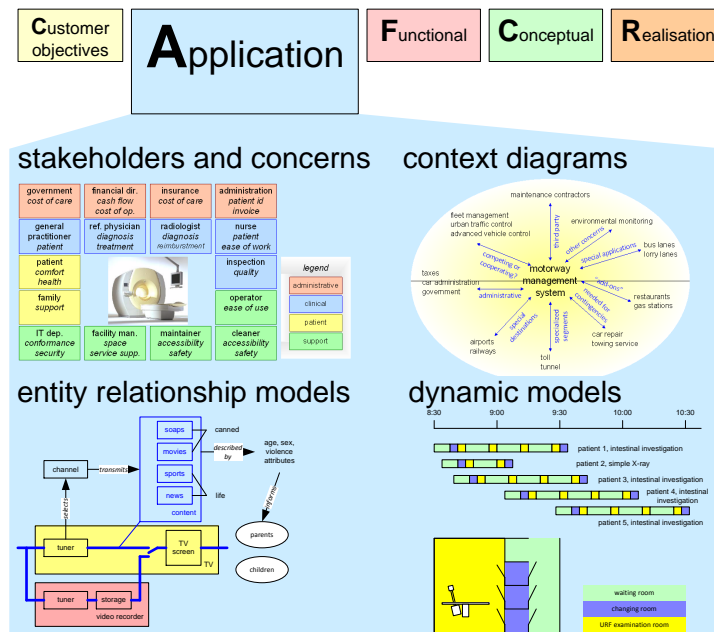


Figure 9.1: Overview of methods and models that can be used in the application view

9.2 Customer stakeholders and concerns

In the daily use of the system many human and organizational entities are involved, all of them with their own interests. Of course many of these stakeholders will also appear in the static entity relationship models. However human and organizations are very complex entities, with psychological, social and cultural characteristics, all of them influencing the way the customer is working. These stakeholders have multiple concerns, which determine their needs and behavior. Figure 9.2 shows stakeholders and concerns for an MRI scanner.

The IEEE 1471 standard about architectural descriptions uses stakeholders and concerns as the starting point for an architectural description.

Identification and articulation of the stakeholders and concerns is a first step in understanding the application domain. The next step can be to gain insight in the *informal* relationships. In many cases the formal relationships, such as organization charts and process descriptions are solely used for this view, which is a horrible mistake. Many organizations function thanks to the unwritten information flows of the social system. Insight in the informal side is required to prevent a solution which does only work in theory.

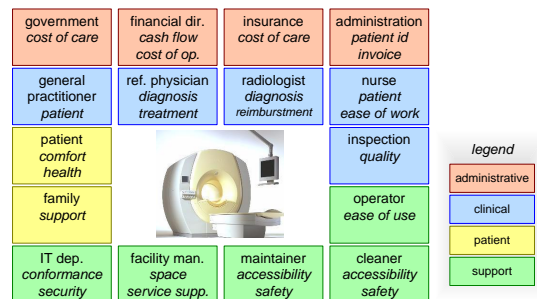


Figure 9.2: Stakeholders and concerns of an MRI scanner

9.3 Context diagram

The system is operating in the customer domain in the context of the customer. In the customer context many systems have some relationship with the system, quite often without having a direct interface.

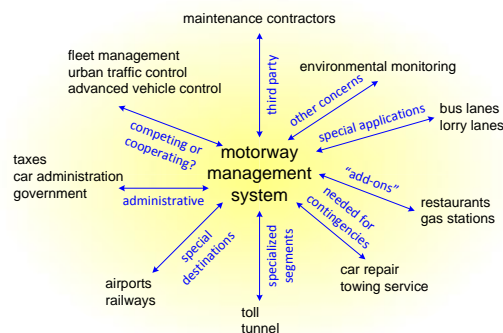


Figure 9.3: Systems in the context of a motorway management system

Figure 9.3 shows a simple context diagram of a motorway management system. Tunnels and toll stations often have their own local management systems, although they are part of the same motorway. The motorway is connecting destinations, such as urban areas. Urban areas have many traffic systems, such as traffic management (traffic lights) and parking systems. For every system in the context questions can be asked, such as:

- is there a need to interface directly (e.g. show parking information to people still on the highway)
- is duplication of functionality required (measuring traffic density and sending it to a central traffic control center)

9.4 Entity relationship model

The OO (Object Oriented software) world is quite used to entity relationship diagrams. These diagrams model the outside world in such a way that the system can interact with the outside world. These models belong in the "CAFCR" thinking in the conceptual view. The entity relationship models advocated here model the customers world in terms of entities in this world and relations between them. Additionally also the activities performed on the entities can be modelled. The main purpose of this modelling is to gain insight in how the customer is achieving his objectives.

One of the major problems of understanding the customers world is its infinite size and complexity. The art of making an useful entity relationship model is to very carefully select what to include in the model and therefore also what **not** to include. Models in the application view, especially this entity relationship model, are by definition far from complete.

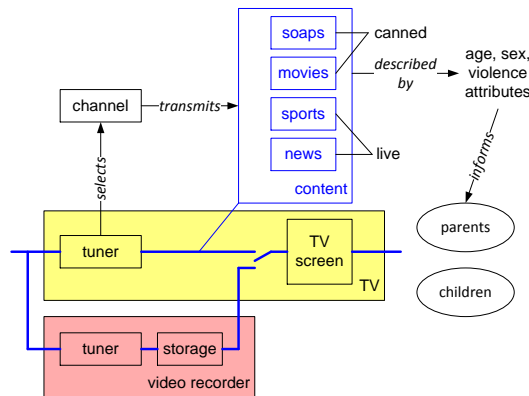


Figure 9.4: Diagram with entities and relationship for a simple TV appliance

Figure 9.4 shows an example of an entity relationship model for a simple TV. Part of the model shows the well recognizable flow of video content (the bottom part of the diagram), while the top part shows a few essential facts about the contents. The layout and semantics of the blocks are not strict, these form-factors are secondary to expressing the essence of the application.

9.5 Dynamic models

Many models, such as entity relationship models, make the static relationships explicit, but don't address the dynamics of the system. Many different models can be used to model the dynamics, or in other words to model the behavior in time. Examples of dynamic models are shown in figure 9.5

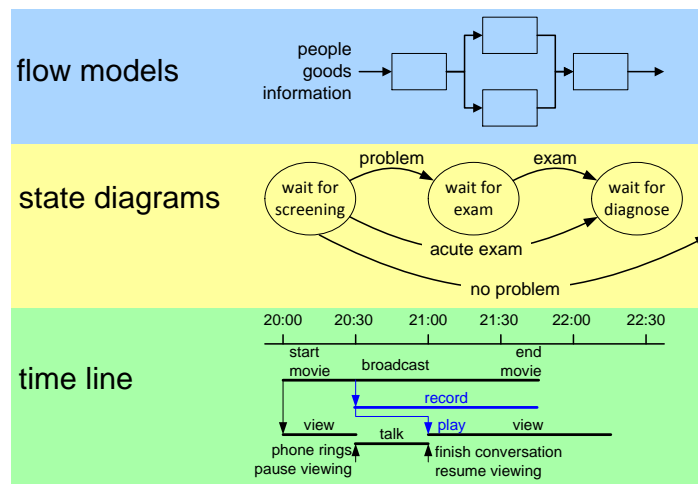


Figure 9.5: Examples of dynamic models

Productivity and Cost of ownership models are internally based on dynamic models, although the result is often a more simplified parameterized model, see figure 9.6.

Figure 9.7 shows an example of a time-line model for an URF examination room. The involved rooms play an important role in this model, therefore an example geographical layout is shown to explain the essence of the time-line model.

The patient must have been fasting for an intestine investigation. In the beginning of the examination the patient gets a barium meal, which slowly moves through the intestines. About every quarter of an hour a few X-ray images-images are made of the intestines filled with barium. This type of examination is interleaving multiple patients to efficiently use the expensive equipment and clinical personnel operating it.

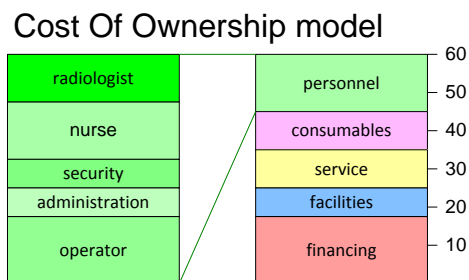
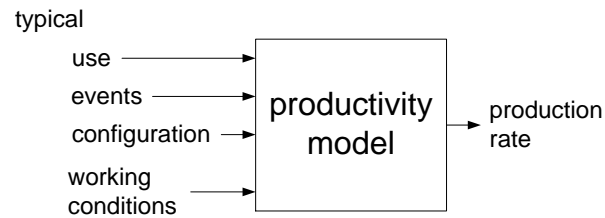


Figure 9.6: Productivity and cost models

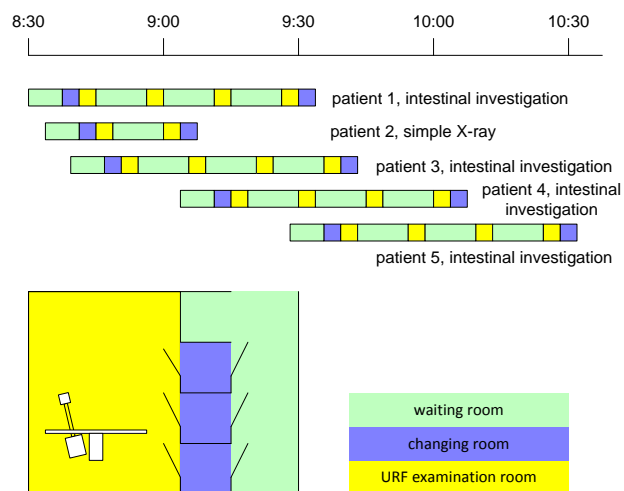


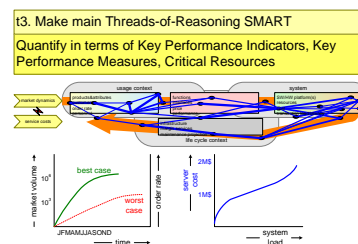
Figure 9.7: Dynamics of an URF examination room

Part V

Integration and Reasoning

Chapter 10

Modeling and Analysis: Reasoning



10.1 Introduction

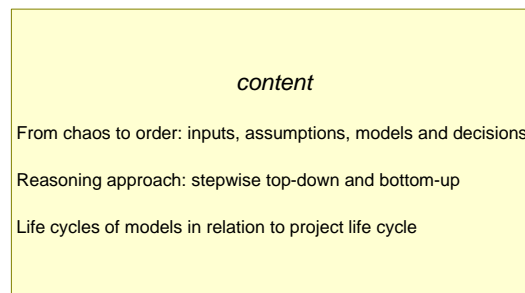


Figure 10.1: Overview of the content of this paper

Figure 10.1 shows an overview of the content of this paper. We will discuss how to get from a chaotic amount of information (inputs, assumptions, models and decisions) to a more ordered situation. We will introduce a *reasoning approach*

that is stepwise concurrently working top-down and bottom-up. We finish with a discussion of life cycles of models in relation to the project life cycle.

Analysis of context and system characteristics during the creation project is based on discussions, experience and inputs from many stakeholders. Some characteristics cannot be predicted in an obvious way. Important, valuable or critical characteristics can be modeled. The size of today's systems and the complexity of the context results in a modular modeling approach: the *grand universal model* is impossible to build, instead we create many small, simple, but related models.

Early in projects lots of fragmented information is available, ranging from hard facts from investigations or stakeholders, to measurement data. This phase appears to be rather chaotic. We discuss the perceived chaos and the need for overview in section 10.2. Section 10.3 provides a stepwise approach to relate all these fragments and to decide on models to be made or integrated.

This approach extends the notion of *threads-of-reasoning*, as described in [5], to modeling and analysis.

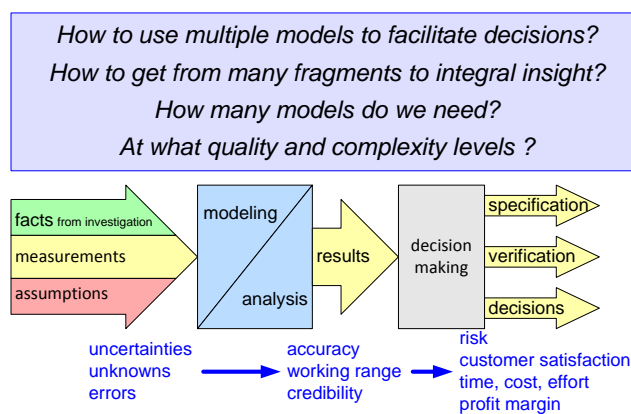


Figure 10.2: Purpose of Modeling

The final purpose of modeling is to facilitate the decision making process of projects. Decisions may range from project management type decisions about time or effort, to decisions about system specification or design. Crucial factors in this decision making process, shown underneath the decision making box in Figure 10.2, are *risk*, *customer satisfaction*, and also the rather tangible factors as time, effort, cost and profit margin.

Figure 10.2 also shows that the results of modeling and analysis are used as input for the decision making process. Modeling and analysis transforms a set of facts from investigations, measurement data and assumptions into information about system performance and behavior in the context. The input data are unfortunately not ideal, uncertainties, unknowns and errors are present. As a consequence models have a limited accuracy and credibility. Models also have a limited working

range due to the chosen (and necessary) abstractions.

The reasoning approach should help to find answers for the following questions:

- How to use multiple models to facilitate decisions?
- How to get from many fragments to integral insight?
- How many models do we need?
- At what quality and complexity levels?

A nice example of relating technical architecture considerations to business architecture considerations is the description of the Google Cluster Architecture by Barroso et al [2].

10.2 From Chaos to Order

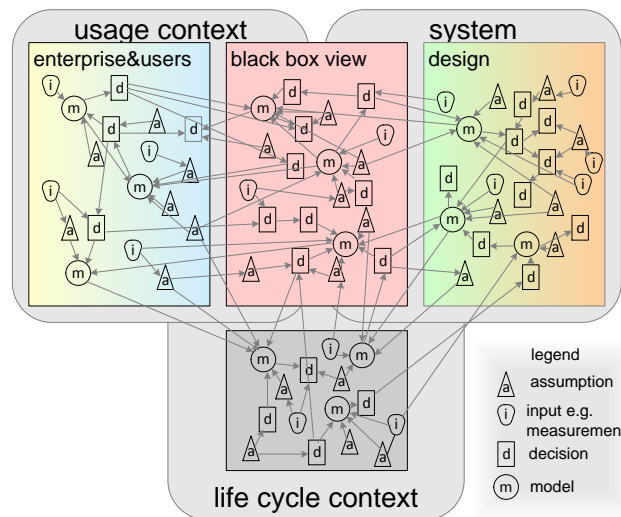


Figure 10.3: Graph of Decisions and Models

The description, as described in the introduction, suggests a rather orderly world and process from data to decision. However, most of today's projects are complex due to problem and project size. This more chaotic view on modeling is shown in Figure 10.3. The basis for this diagram is the standard diagram of the usage context, the life cycle context and the system itself. This space is populated with information to create a landscape. Four types of information are shown:

Decision is a consciously taken decision by one of the stakeholders. For example, the company strives for 40% margin, the system must have a throughput of 10 requests per second, or the infrastructure will run without any operator.

Input is information from some investigation, for example from suppliers, or obtained by measurements.

Assumption are made by project members or other stakeholders, whenever hard information is missing. Quite often assumptions are made unconsciously. For example, the programmer assumes that CPU load and memory consumption of a function is small and may be ignored. No programmer has the time to measure all functions in all possible circumstances. We make assumptions continuously, based on experience and craftsmanship.

Model is created to transform information into usable results for the decision making process. For example, a throughput model or a memory consumption model.

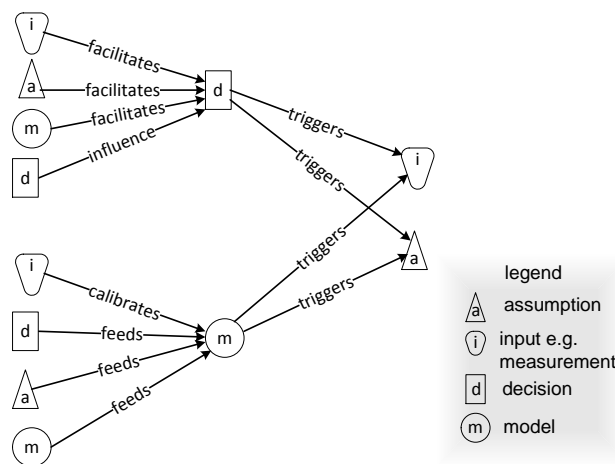


Figure 10.4: Relations: Decisions, Models, Inputs and Assumptions

These four types of information are related, as shown in Figure 10.4. Decisions are facilitated by inputs, models and assumptions. Most decisions are based on previous decisions. During the discussion preceding a decision missing information is detected. This triggers the search for this information or forces new assumptions. Models are fed by other models, decisions and assumptions. Inputs feeds models, but is also used to calibrate models. While we create models, many open issues are discovered, triggering new inputs and assumptions.

The combination of all this information and their relations creates a huge graph, which represented in this way, is chaotic, see Figure 10.3.

A somewhat less abstract graph is shown in Figure 10.5 for the web shop example. In this figure the relations have been left out, because the diagram is already overcrowded as is.

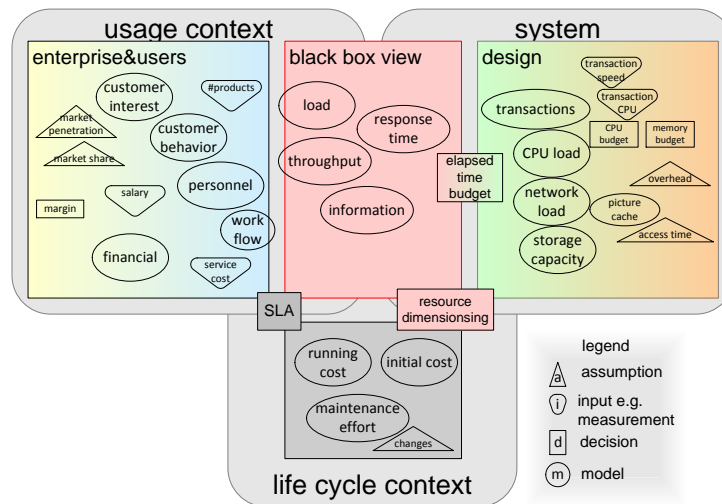


Figure 10.5: Example Graph for Web Shop

The challenge we are going to address is to bring order in this chaos. However, we as modelers must understand that the ordering is a representation of an inherently complex and intertwined reality.

10.3 Approach to Reason with Multiple Models

An overview of the stepwise approach is shown in Figure 10.6. After a quick scan of the system, the usage context and the life cycle context, the approach propagates two concurrent tracks: *top-down* and *bottom-up*. The results of these 2 concurrent tracks are consolidated, capturing both decisions as well as the overview. The side effect of the concurrent activities is that the involved human participants have increased insight in problem space and solution space.

The entire set of steps is reiterated many times. Every iteration the focus is shifting more to relevant (significant, critical) issues, reducing project risks, and increasing project feasibility. The purpose of this rapid iteration is to obtain short feedback cycles on modeling efforts, analysis results and the specification and project decisions based on the results.

One of the frequently occurring pitfalls of modeling is that too much is modeled. A lot of time and effort is wasted with little or no return on investment. This waste can be avoided by getting feedback from the actual project needs, and aborting modeling efforts that are not fruitful.

The first step is the exploration of the *landscape*: the system itself, the usage context and the life cycle context. Specification and design decisions ought to be justified by the value provided to the usage context or the life cycle context.

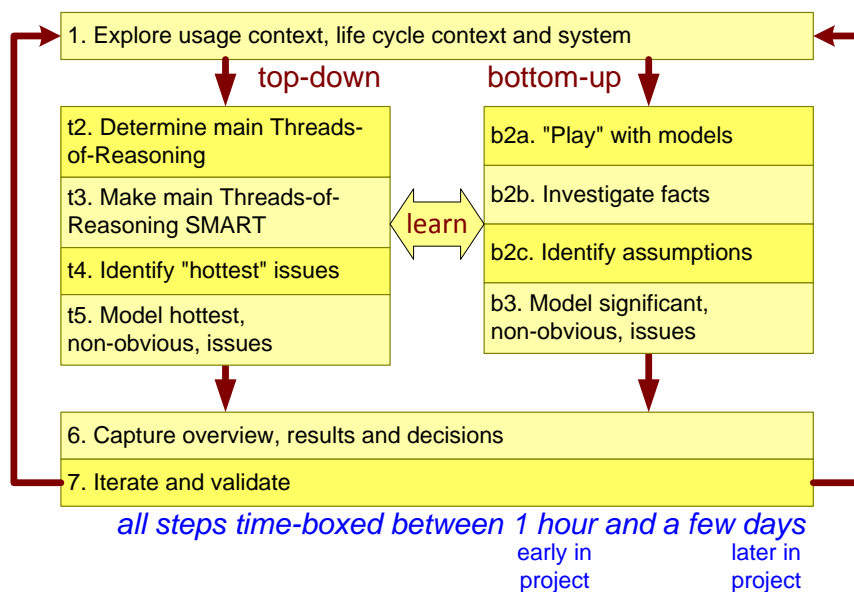


Figure 10.6: Reasoning Approach

Modeling is supportive to the processes of specification, verification and decision making. Step 1, shown in Figure 10.7 is a scan through the system and the contexts, high lighting potential problems and risks, and identifying valuable, important needs, concerns and requirements and identifying critical sensitive or difficult design and implementation issues.

After step 1 two concurrent tracks are used: top-down (t) and bottom-up (b). In the top-down track we work from customer and life cycle objectives towards models to reduce project risks and to increase project feasibility. The bottom-up track scans through many details to get insight in significance of details or when details may be neglected. The bottom-up track ensures that models don't drift too far from reality.

The first top-down step is t2: creation of thread(s)-of-reasoning, shown in Figure 10.8. Threads-of-reasoning provide a means to relate business needs and concerns to specification to detailed realization decisions. This method is described in [5]. The most relevant individual chapters can be downloaded separately at: <http://www.gaudisite.nl/ThreadsOfReasoningPaper.pdf> and <http://www.gaudisite.nl/MIthreadsOfReasoningPaper.pdf>.

A thread-of-reasoning is constructed by creating a graph of related concerns and needs via business process decisions towards system requirements and then connected to design concepts and realization decisions. Such a graph is often quite extensive and messy. Relationships can either be supportive or reenforcing, or

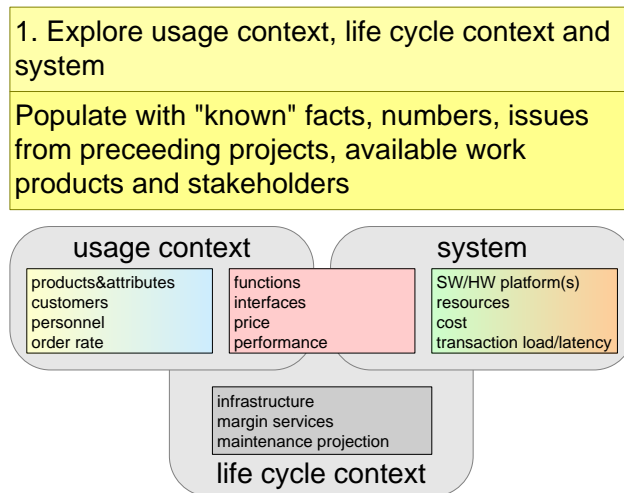


Figure 10.7: 1. Explore

based on tension. The actual thread-of-reasoning reduces this graph to essential relationships of both kinds. The essence of most projects can be expressed in one or a few of these threads.

For instance in the case of a web shop, the main tensions might be between the need to be flexible in terms of sales volume and the need for affordable cost of the supporting IT services. If the web shop starts in a completely new market, then the order rate prediction might be highly uncertain. Nevertheless, the web shop owner needs to be able to serve millions of customers if the market development is positive. At the same time the web shop owner can not afford a grossly over-dimensioned server-park. These two concerns ripple through the landscape, touching on many aspects, such as amount of required web shop personnel, amount of maintenance work, price of the IT hardware itself, resource consumption of web services et cetera.

The result of step t2 is a qualitative graph of relations. In step t3, shown in Figure 10.9, this graph is annotated with quantitative information and relations. For example the uncertainty of the sales volume can be quantified by making a best case and a worst case scenario for the sales growth over time. At system level the expected relation between system load and server cost can be quantified.

Figure 10.10 explains a frequently used acronym: SMART [3]. This acronym is used amongst others as checklist for the formulation of requirements. There appears to be consensus about the meaning of the first two letters. The letter *S* stands for *Specific*: is the definition of the subject well-defined and sharp. The letter *M* is used for *Measurable*: is the subject measurable. Measurability often requires quantification. Measurability is needed for verification.

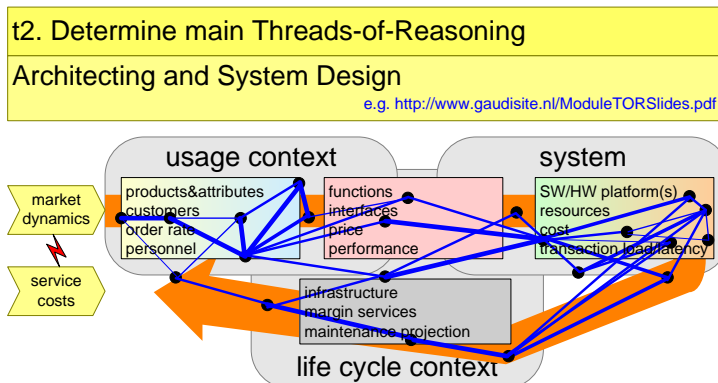


Figure 10.8: t2. Thread-of-Reasoning

The main questions in step t3 are related to *Specific* and *Measurable*. The letters *A*, *R* and *T*, may provide useful insight also, see Figure 10.10 for their varied meanings.

The next top-down step is to determine the “hottest” issues. The hottest issues are those issues that require most attention from the architect and the project team. There are several reasons that an issue can be important:

highest (perceived) risk as result, for instance, of a project risk inventory. The subtle addition of the word *perceived* indicates that an issue is hot when it is perceived as having a high risk. Later modeling may show that the actual risk is lower.

most important/valuable from usage or life cycle perspective. Core processes, key functions and key performance parameters require attention by definition.

most discussed within the project team or by outside stakeholders. Lots of discussions are often a sign of uncertainty, perceived risk, or ill communicated decisions. Attention is required to transform fruitless discussions into well defined decisions and actions. In some cases models provide the means for communication, filtering out organizational noise and providing focus on real hot issues.

historic evidence of experienced project members. The past often provides us with a wealth of information and potential insights. If we know from past experience that we systematically underestimate the number of transaction with one order of magnitude, for example, then we can analyze the cause of past failures and create more valid estimates this time.

The issues present in the thread-of-reasoning can be assessed from these different perspectives. We propose to use a scale from 1 to 5, with the meaning 1 = cold and

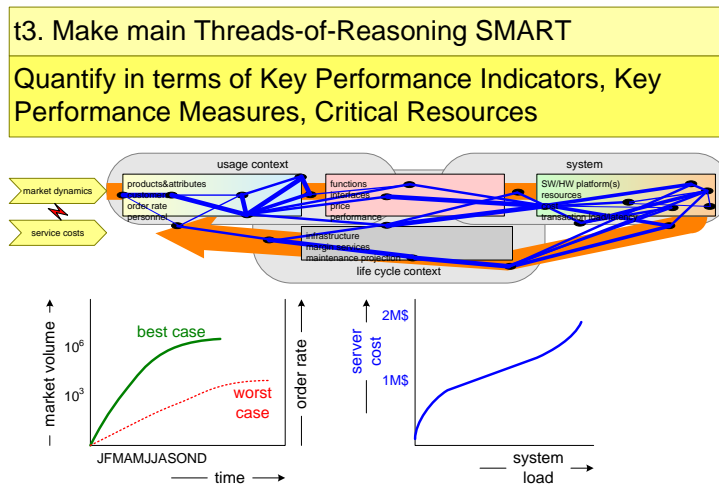


Figure 10.9: t3. SMART'en Thread-of-Reasoning

5 = hot. For example, for risks a small risk = 1, a large risk = 5. The table at the bottom of Figure 10.11 shows a number of issues and the assessment of these issues for the four perspectives. For visual support the values are color coded with the heat scale: 5 = red, 4 = orange, 3 = yellow. Issues where scores are present of 5 or 4 deserve attention anyway.

The issues are ranked after assessing individual issues against these four perspectives. The hot (5) and very warm (4) issues get more weight than the colder issues. The *order rate* is apparently the hottest issue, with a lot of value attached to it (not being able to serve quickly increasing sales would be disastrous), with a lot of discussion caused by the uncertainty, and a high risk due to the unavailable

• Specific	quantified	
• Measurable	verifiable	acronym consensus
• Assignable (Achievable, Attainable, Action oriented, Acceptable, Agreed-upon, Accountable)		
• Realistic (Relevant, Result-Oriented)		
• Time-related (Timely, Time-bound, Tangible, Traceable)		variation of meaning

Figure 10.10: Intermezzo: the acronym SMART

t4. Identify "hottest" issues						
assess explored landscape:						
highest (perceived) risk	1..5 scale,					
most important/valuable	1 = low risk					
most discussed	5 = high risk					
historic evidence	et cetera					
urgency						
rank issues according to aggregated assessment						
	risk	value	discussion	history	urgency	ranking
server cost	2	3	2	1	3	
order rate	4	5	5	3	5	1
transactions	3	3	3	4	2	3
response time	3	5	1	4	2	2
availability	2	5	1	3	3	4
network bandwidth	1	1	3	1	3	
storage capacity	1	1	1	2	3	

Figure 10.11: t4: Identify Hottest

facts. The *response time* scores second, based on value (customer dissatisfaction endangering sales if the system is slow responding) and past experience (many new applications perform far below expectation in the first year of operation). Based on historic evidence (number of transactions is always severely underestimated) and the medium level of all other perspectives, turns the *number of transactions* into the third issue. Finally the *availability* requires attention, because of the very high customer value. Craftsmanship of the project team indicates that *availability* ought not to be a problem.

The uncertainty in the order intake can be modeled by taking best and worst case scenarios. In Figure 10.12 it is shown that the best case (from business point of view) is an exponential increase of sales to a saturation level of about ten million products. The worst case is an exponential growth to a saturation level of only 10 thousand products. In the same graph server capacities are modeled. For the server capacities it is assumed that the business starts with a rather small server. When needed an additional small server is added. In the best case scenario the next increment is a much more powerful server, to prepare for the rapid increase in sales. Following increments are again powerful servers. Note that these IT infrastructure scenarios require increments of powerful servers with a lead-time in the order of one month. These simple models allow for further modeling of income, cost and margin.

The bottom-up track in itself has three concurrent activities:

b2a. "Play" with models to create insight in relevant details. Simple models are created and the input and the structure of the model is varied to get insight

t5. Model hottest, non-obvious, issues

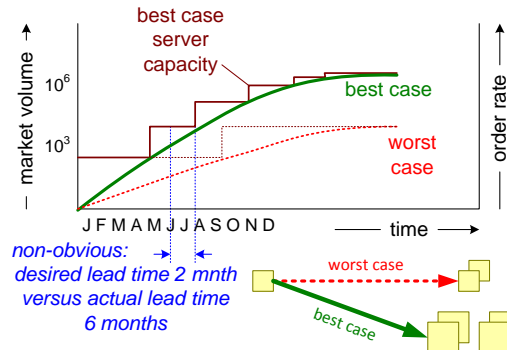


Figure 10.12: t5: Model Hottest Issues

in issues that have significant impact higher system levels. Note that these models may address issues in any of the contexts: usage, life cycle or the system itself. In Figure 10.13 some simple models are shown relating transactions to number of CPU's or cores.

b2b. Investigate facts from any relevant source: market research, supplier documentation, internet, et cetera. Figure 10.13 shows as an example the TPC-C benchmark results found on internet and transforms these numbers in $t_{1transaction}$, as used in step b2a. Note that the load of other work-flows in TPC-C is ignored in this simple estimation.

b2c. Identify assumptions made by any stakeholder, including the architect self. The example in Figure 10.13 shows some assumptions made in steps b2a and b2b:

- the server load is dominated by transactions
- the transaction load scales linear
- the TPC-C benchmark is representative for our application. We ignored the other TPC-C workload, what is the effect of other TPC-C workload on the transaction load?

Note that most assumptions are implicit and hidden. Most stakeholders are unaware of the assumptions they made implicitly.

Only few detailed issues are modeled somewhat more extensively. Criteria to continue modeling is the significance of these details at system and context level, and the transparency of the subject. Only non-obvious subjects, where it is not

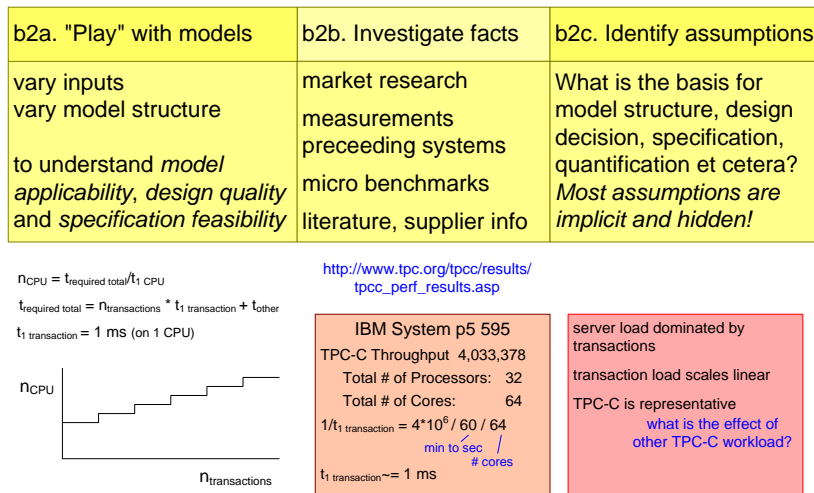


Figure 10.13: b2abc: Bottom-up

directly evident what the effect of change is, should be modeled. Figure 10.14 shows a somewhat more detailed model of memory used for picture transfers and buffering in the servers.

The concurrent tracks for top-down and bottom-up mutually influence each other. The top-down track may learn that significant issues are hidden somewhere in the details. As a consequence the question in top-down is: “Do we address the relevant decomposition”? The bottom-up track gets input about the relevancy of exploring specific details. In the bottom-up track the question is: “What details have significant impact”? Figure 10.15 shows as mutually related questions the top-down uncertainty about the sales volume and the bottom-up impact of transactions on server load.

During the top-down modeling we may have discovered the potential size of the product catalogue and the number of changes on this product catalogue. Where does this product catalogue size impact the design? Bottom-up we have found that 3 parameters of the memory usage model have significant impact on system performance: concurrency (in terms of the number of concurrent server threads), cache size (in number of cached pictures) and picture size (average number of bytes per cached picture). Combined we see that catalogue size will relate somehow to cache size. In other words we can refine the memory usage model with more focus by taking the catalogue size into account.

Many problems and questions that are addressed by modeling appear to be local, but are in practice related to other issues in the usage context, life cycle context or the system itself. The threads-of-reasoning are used to make the most important relations explicit. When we make small models in a step-by-step fashion

b3. Model significant, non-obvious, issues

for example, memory use in server(s) for picture transfers and buffering

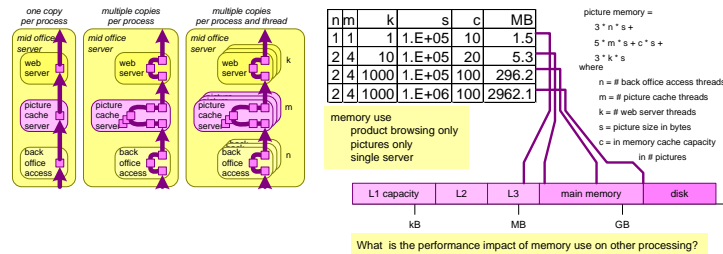


Figure 10.14: b3: Model Significant Issues

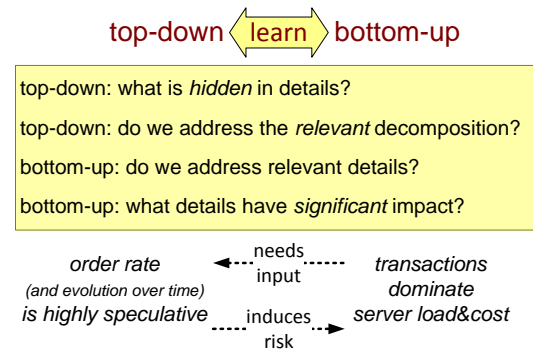


Figure 10.15: Learning Concurrent Bottom-up and Top-down

as described above, we have to take care that the models or their results are reconnected again. The reconnection is achieved by going through the same process many times:

- using results from the previous iteration to improve the thread-of-reasoning
- using the insights from the previous iteration to dig deeper into significant issues

During the iterations questions are asked to stakeholders to obtain input, data is provided to stakeholders for validation, and stakeholders may bring in new information spontaneously. Modeling is not at all an isolated activity, it is one of the communication means with stakeholders! Also a lot of experiments and measurements are done during those iterations at component level or at system level, in the real world or in the modeled world. The iteration with the stakeholder interaction

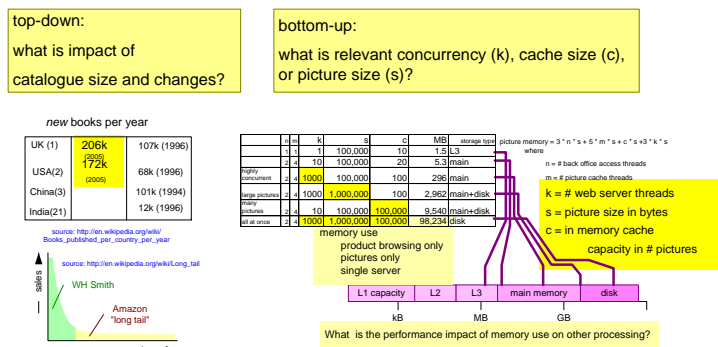


Figure 10.16: Example top-down and bottom-up

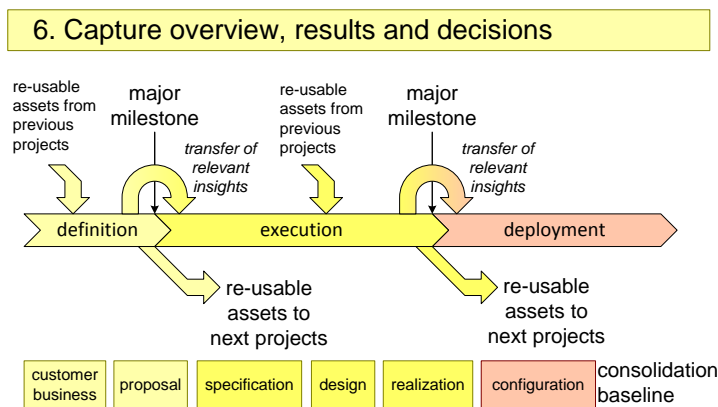


Figure 10.17: 6. Capture overview, results and decisions

and the interaction with systems and components is shown in Figure 10.18

The iterations and the steps within the iteration should be time-boxed. very early in the project one iteration can be done in one hour to establish a shared baseline in the architecting team. The time-boxes will increase to hours and ultimately to a few days at the end of the project. The maximum of a few days is to prevent modeling activities that are not problem oriented. Modeling is already an indirect activity, when lots of time is spent without feedback, then the risk is large that this time and effort is wasted.

The focus of the modeling activities is shifting over time. Early during the project life cycle, the focus will be on the usage context and the feasibility of the system. When the solution crystallizes, then the life cycle issues become more visible and tangible. After conception and feasibility more attention is paid to the life cycle context. During the design and realization of the system most of the

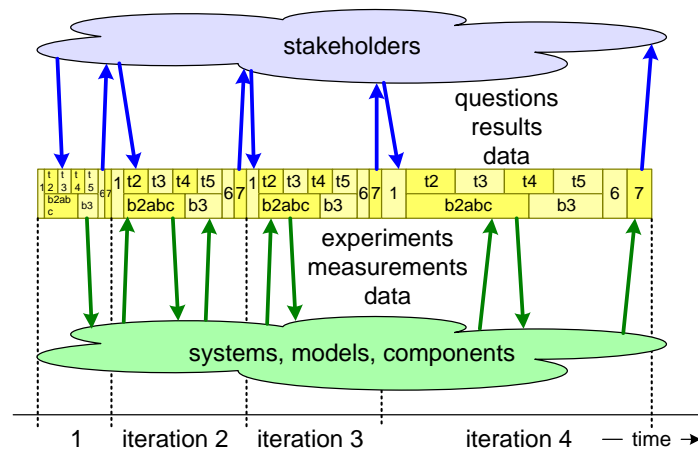


Figure 10.18: 7. Iterate and Validate

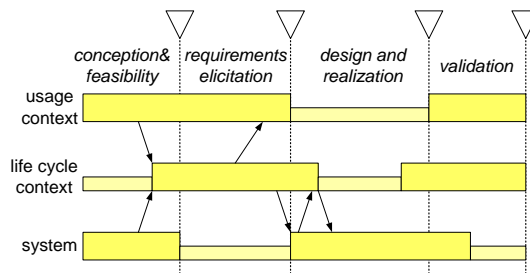


Figure 10.19: The focus is shifting during the project

focus will be on the system itself. At the end of design and realization the life cycle context will increase again, due to the eminent deployment of the actual system. Finally during validation the emphasis will shift from the system to the validation of the system in the context(s).

10.4 Balancing Chaos and Order

Architects and designers are unconsciously juggling with lots of inputs, assumptions, and decisions in their head. They iterate over the steps in this reasoning approach with very short cycle times. This enables them to understand relations and identify issues quickly. Unfortunately, this information and insight is very intangible and not easily shared with other stakeholders. Figure 10.20 positions the process in the head of the architect relative to the iteration cycle time (horizontal axis) and the communication scope in number of people involved (vertical axis).

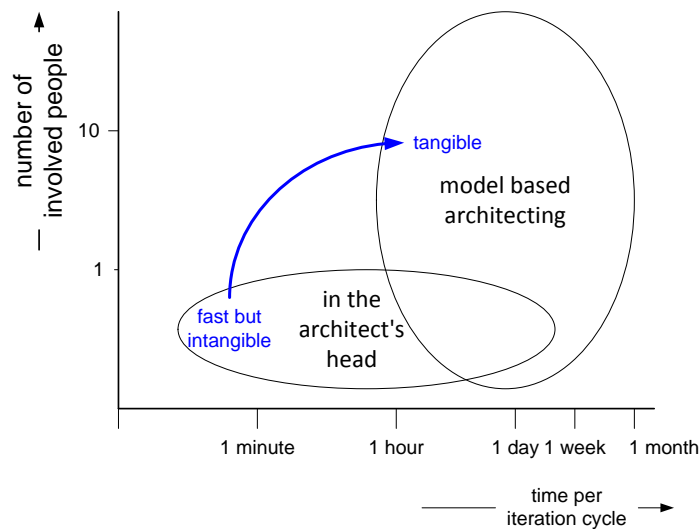


Figure 10.20: Models Support Communication

Models facilitate communication between multiple people, because knowledge and insights have been made more tangible. Note that a model itself is not yet insight. Insight is obtained by interaction between persons and interaction between person and model. To create *relevant* models taking into account *significant* details a team of people has to iterate as described in the reasoning approach.

Figure 10.21 quantifies the chaos discussed above. It shows that in a single project millions of implicit decisions are taken based on millions of assumptions. For example, a programmer coding a loop like:

```
for image in images:
    process(image)
```

has decided to use a *for*-loop, based on the assumption that the overhead is small. Implicit decisions and assumptions are mostly about obvious aspects, where experience and craftsmanship provide a shortcut that is not made explicit. This implicit process is very important, because we would create a tremendous overhead if we have to make all obvious aspects explicit. Tens of thousands decisions and thousands of assumptions and inputs are made explicit, for instance in detailed design specifications. For such a project hundreds of try-out models are made, tens of of these models get documented as simple and small models. About 10 key decisions, such as key performance parameters, are used to control the project. Few *substantial* models are used or made.

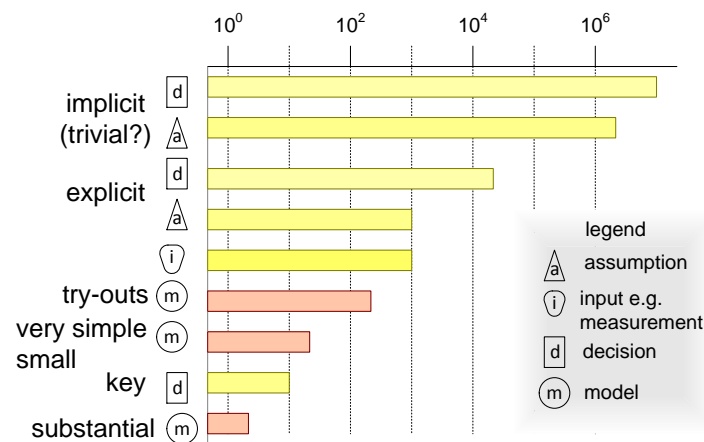


Figure 10.21: Frequency of Assumptions, Decisions and Modeling

10.5 Life Cycle of Models

Models have their own life cycle. The purpose of models is shifting during the project. This shift of purpose is shown at the top of Figure 10.22.

understanding problem and potential solutions and getting insight is the early purpose of creating and using models.

exploration of problem and solution space to find an acceptable solution.

optimization and fine tuning of the chosen solution to fit as good as possible in the usage context and life cycle context.

verification of the realization, where the model is used as test reference. Differences between realization and model must be explainable.

Project team members have to start somewhere early in the project to get started in understanding the system and its context. Making small models, called *try-out models* in Figure 10.22, such a start is made. Many *try-out* models provide some insight and are not useful afterwards. For example, the performance of the system is independent of the amount of memory used, as long as less than 80% of the available memory is used. The 80% working range is a useful input, the model itself can be abandoned. Some *try-out* models keep useful somewhat longer for its creator, but often these *try-out* models lose their value or, if they prove to be very valuable, they get slightly more formalized. This next level of formalization is called *small and simple* models in this figure.

Simple and small models have some more long term value. The model and the modeling results are documented as part of the project archive. However,

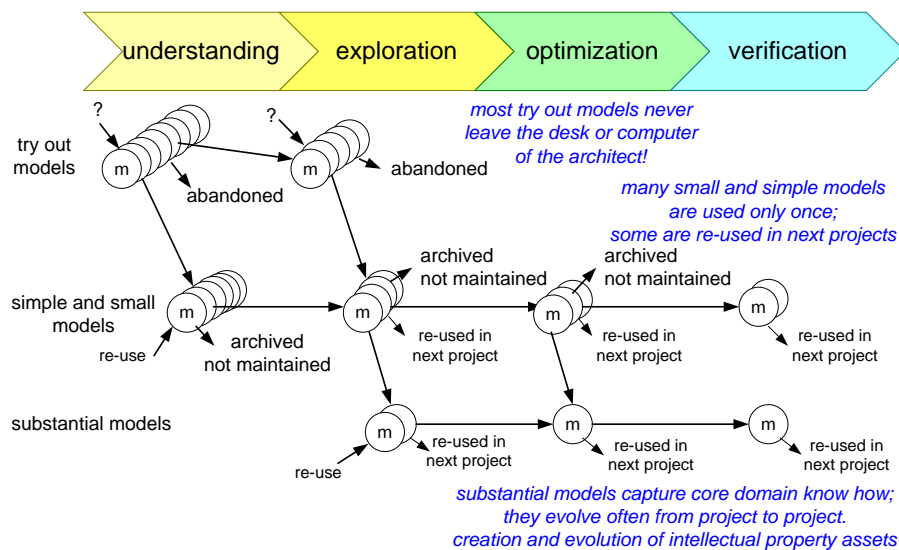


Figure 10.22: Life Cycle of Models

many of those models are valuable once. The documentation is in that case only archived, but not maintained. This makes results traceable, without creating a large maintenance burden. Some of the *small and simple* models are re-used by next projects. When models tend to grow and provide consistent value, then they become *substantial* models.

Substantial models tend to capture a lot of core domain know how. These models evolve from project to project. Eventually these models might become a product in their own right. For example a *load balancing simulation for web services* simulation tool might be used by many web services based systems.

Most modeling is used for understanding and exploration. Only a fraction of the models is also used for optimization purposes or for verification.

Note that a model used for exploration is in itself not directly useful for optimization. For optimization we have to add input generation and result evaluation. Also the performance of the model itself may become much more important, in order to run and evaluate the model for many different potential configurations. Going from exploration to optimization can be a significant investment. Such an investment is only in a few cases justifiable.

For verification models may have to be adapted as well. Simple verification of implementation versus model as sanity check can be done without many changes. However, if we want to use models as part of the integration process, then we have to invest in interfaces and integration of models with actual implementations. Also for comparison of results between models and implementation we need to automate evaluation and archiving. We have to consider the return on investment

before actually investing in verification modeling and support.

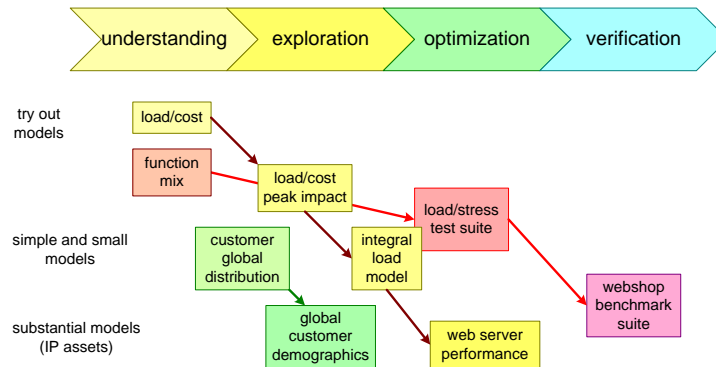


Figure 10.23: Examples of Life Cycle of Models

Figure 10.23 shows a number of examples of the evolution of models throughout their life cycle. A small model is made to estimate the load of the servers and the related cost. A separate small model is used for the mix of functions running on the servers. The load model evolves to take peak loads into account. The peak load depends on the usage characteristics. For global systems the distribution of customers over time zones is highly relevant, because peaks in this distribution cause peak loads. The global distribution of customers proves to be rather generic and evolves into a re-usable asset, a *global customer demographics* model. The load model may evolve further into an *integral load model*, where loads and solutions can be studied and compared to find an appropriate solution. A full fledged simulation model to study load and server dimensioning could be a re-usable asset, a *web server performance* model. The very simple function mix model may evolve into a load and stress test suite. A more generic variant of such a test suite is a *web shop benchmark*.

10.6 Summary

Figure 10.24 provides a summary of this paper.

The reasoning approach emphasize the complementary value of working *top-down* and *bottom-up*. To reduce the chaos we have to reduce the amount of information we are working on. Key words for selection are *hottest*, *non-obvious*, *significant*, and *relevant*. We recommend the use of multiple small models that are used in combination, rather than making large and much more complicated models addressing multiple issues at once. Models itself have a life cycle. Some models evolve from very simple to more substantial, while other models stop to involve much more early.

<p style="text-align: center;"><i>Conclusions</i></p> <p>Top-down and bottom-up provide complementary insights</p> <p>Key words for selection: hottest, non-obvious, significant, relevant</p> <p>Multiple small models are used in combination</p> <p>Some models evolve from very simple to more substantial</p>
<p style="text-align: center;"><i>Techniques, Models, Heuristics of this module</i></p> <p>Threads-of-reasoning</p> <p>SMART</p> <p>Key Performance Indicators, Key Performance Measures, Critical Resources</p> <p>Ranking matrices</p>

Figure 10.24: summary of this paper

We have used several techniques:

- Threads-of-reasoning
- SMART
- Key Performance Indicators, Key Performance Measures, Critical Resources
- Ranking matrices

10.7 Acknowledgements

Russ Taylor asked for a stepwise how-to and provided feedback.

Part VI

Analysis

List of Figures

1.1	An architecting method supports the architect in his process to go from a vague notion of the problem and a vague notion of the potential solutions to a well articulated and structured architecture description. Modeling and Analysis supports the architecting effort.	3
1.2	Modeling and Analysis supports:	3
1.3	Purpose of Modeling	4
1.4	What to Model?	4
1.5	Program of Modeling and Analysis Course	5
1.6	Overview of Approach	6
1.7	Iteration over viewpoints	7
2.1	Image Retrieval Performance	10
2.2	Straight Forward Read and Display	10
2.3	More Process Communication	11
2.4	Meta Information Realization Overhead	11
2.5	I/O overhead	12
2.6	Non Functional Requirements Require System View	12
2.7	Function in System Context	13
2.8	Challenge	13
2.9	Summary of Problem Introduction	14
3.1	Overview Content <i>Fundamentals of Technology</i>	15
3.2	What do We Need to Analyze?	16
3.3	Typical Block Diagram and Typical Resources	17
3.4	Hierarchy of Storage Technology <i>Figures of Merit</i>	17
3.5	Performance as Function of Data Set Size	18
3.6	Communication Technology Figures of Merit	19
3.7	Multiple Layers of Caching	19
3.8	Why Caching?	20
3.9	Example Web Shop	21
3.10	Impact of Picture Cache	22
3.11	Risks of Caching	22

3.12 Summary	24
4.1 Presentation Content	26
4.2 Measuring Approach: What and How	27
4.3 What do We Need? Example Context Switching	28
4.4 Define Quantity by Initial Model	29
4.5 Define Required Accuracy	30
4.6 How to Measure CPU Time?	30
4.7 Define the Measurement Set-up	31
4.8 Case: ARM9 Hardware Block Diagram	32
4.9 Key Hardware Performance Aspect	32
4.10 OS Process Scheduling Concepts	33
4.11 Determine Expectation	33
4.12 Determine Expectation Quantified	34
4.13 Code to Measure Context Switch	34
4.14 Measuring Context Switch Time	35
4.15 Understanding: Impact of Context Switch	36
4.16 Accuracy: Measurement Error	37
4.17 Accuracy 2: Be Aware of Error Propagation	37
4.18 Intermezzo Modeling Accuracy	38
4.19 Actual ARM Figures	38
4.20 Expectation versus Measurement	39
4.21 Context Switch Overhead	39
4.22 Summary Measuring Approach	41
5.1 Overview of the content of this paper	44
5.2 What to Model in System Context?	45
5.3 Approach to System Modeling	46
5.4 Web Shop: NFR's, Properties and Critical Technologies	46
5.5 4. Determine Relations	47
5.6 5. Rank Relations	48
5.7 Purpose of Picture Cache Model in Web Shop Context	48
5.8 Zero Order Load Model	49
5.9 First Order Load Model	49
5.10 Quantification: From Formulas to Insight	50
5.11 Hit Rate Considerations	51
5.12 Response Time	52
5.13 What Memory Capacity is Required for Picture Transfers?	52
5.14 Process view of picture flow in web server	53
5.15 Formula memory Use Web Server	53
5.16 Web server memory capacity	53
5.17 Only a small part of the system has been modeled so far	54

5.18	The modeling so far has resulted in understand some of the systems aspects	54
5.19	Refinement of the system models takes place after context modeling	55
5.20	Summary of system modeling	55
6.1	Definition of a budget in the technical world	57
6.2	Goals of budget based design	58
6.3	Visualization of Budget-Based Design Flow. This example shows a response time budget.	59
6.4	Budget-based design steps	60
6.5	Example of a quantified understanding of overlay in a waferstepper	61
6.6	Example of a memory budget	62
6.7	Power Budget Visualization for Document Handler	64
6.8	Alternative Power Visualization	64
6.9	What kind of budget is required?	65
6.10	Evolution of Budget over Time	65
6.11	Summary of budget based design	66
6.12	Colophon	67
7.1	Product Related Life Cycles	69
7.2	System Life Cycle	70
7.3	Approach to Life Cycle Modeling	70
7.4	What May Change During the Life Cycle?	71
7.5	Simple Model of Data Sources of Changes	72
7.6	Data Sources of Web Server	73
7.7	Example Product Portfolio Change Books	73
7.8	Example Customer Change	74
7.9	Web Shop Content Update	75
7.10	Web Shop Content Change Effort	76
7.11	Life-cycle Differences for health care equipment	77
7.12	Web Shop Security and Changes	77
7.13	Web Shop Reliability and Changes	78
8.1	The relation between sales price, cost price and margin per product	80
8.2	Profit as function of sales volume	81
8.3	Investments, more than R&D	82
8.4	Income, more than product sales only	83
8.5	The Time Dimension	84
8.6	The “Hockey” Stick	84
8.7	What if ...?	85
8.8	Stacking Multiple Developments	86
8.9	Fashionable financial yardsticks	87

9.1	Overview of methods and models that can be used in the application view	90
9.2	Stakeholders and concerns of an MRI scanner	91
9.3	Systems in the context of a motorway management system	91
9.4	Diagram with entities and relationship for a simple TV appliance	92
9.5	Examples of dynamic models	93
9.6	Productivity and cost models	94
9.7	Dynamics of an URF examination room	94
10.1	Overview of the content of this paper	96
10.2	Purpose of Modeling	97
10.3	Graph of Decisions and Models	98
10.4	Relations: Decisions, Models, Inputs and Assumptions	99
10.5	Example Graph for Web Shop	100
10.6	Reasoning Approach	101
10.7	1. Explore	102
10.8	2. Thread-of-Reasoning	103
10.9	3. SMART'en Thread-of-Reasoning	104
10.10	Intermezzo: the acronym SMART	104
10.11	4. Identify Hottest	105
10.12	5. Model Hottest Issues	106
10.13	2abc: Bottom-up	107
10.14	b3: Model Significant Issues	108
10.15	Learning Concurrent Bottom-up and Top-down	108
10.16	Example top-down and bottom-up	109
10.17	6. Capture overview, results and decisions	109
10.18	7. Iterate and Validate	110
10.19	The focus is shifting during the project	110
10.20	Models Support Communication	111
10.21	Frequency of Assumptions, Decisions and Modeling	112
10.22	Life Cycle of Models	113
10.23	Examples of Life Cycle of Models	114
10.24	summary of this paper	115

List of Tables

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] Luiz André Barroso, Jeffrey Dean, and Urs Hölzle. Web search for a planet: The google cluster architecture. <http://labs.google.com/papers/googlecluster-ieee.pdf>, March-April 2003. IEEE Computer Society, pages 22-28.
- [3] George T. Doran. There's a S.M.A.R.T. way to write management's goals and objectives. *Management Review (AMA Forum)*, pages 35–36, November 1981.
- [4] Gerrit Muller. The system architecture homepage. <http://www.gaudisite.nl/index.html>, 1999.
- [5] Gerrit Muller. CAFCR: A multi-view method for embedded systems architecting: Balancing genericity and specificity. <http://www.gaudisite.nl/ThesisBook.pdf>, 2004.

History

Version: 0.1, date: 23 February, 2007 changed by: Gerrit Muller

- Added chapter fundamentals of technology
- added chapter system model
- moved list of tables and list of figures
- reduced table of contents depth

Version: 0, date: 6 February, 2007 changed by: Gerrit Muller

- Created very preliminary bookstructure, no changelog yet