

Module Modeling and Analysis: Integration and Reasoning



Gerrit Muller

Embedded Systems Institute

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

`gerrit.muller@embeddedsystems.nl`

Abstract

This module addresses the integration of small or partial models into bigger models. We also discuss how multiple models are used and how to reason using multiple models.

The complete course MA 611TM is owned by Embedded Systems Institute. To teach this course a license from Embedded Systems Institute is required. This material is preliminary course material. The final material and course information can be found at: www.esi.nl/cursus.

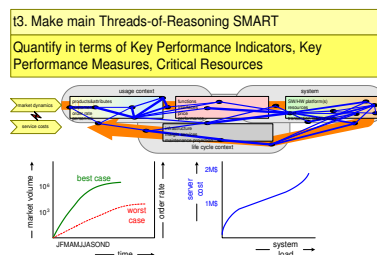
All Gaudí documents are available at:
<http://www.gaudisite.nl/>

Contents

1	Modeling and Analysis: Reasoning	1
1.1	Introduction	1
1.2	From Chaos to Order	3
1.3	Approach to Reason with Multiple Models	5
1.4	Balancing Chaos and Order	16
1.5	Life Cycle of Models	17
1.6	Summary	20
1.7	Acknowledgements	20

Chapter 1

Modeling and Analysis: Reasoning



1.1 Introduction

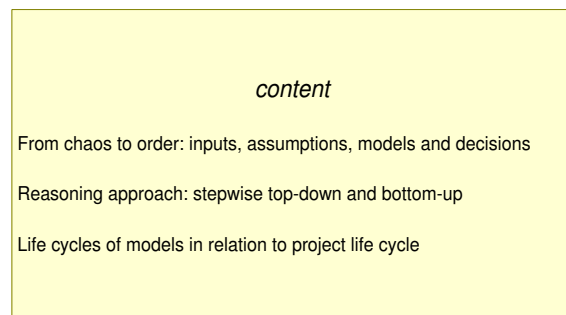


Figure 1.1: Overview of the content of this paper

Figure 1.1 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 1.2. Section 1.3 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 [4], to modeling and analysis.

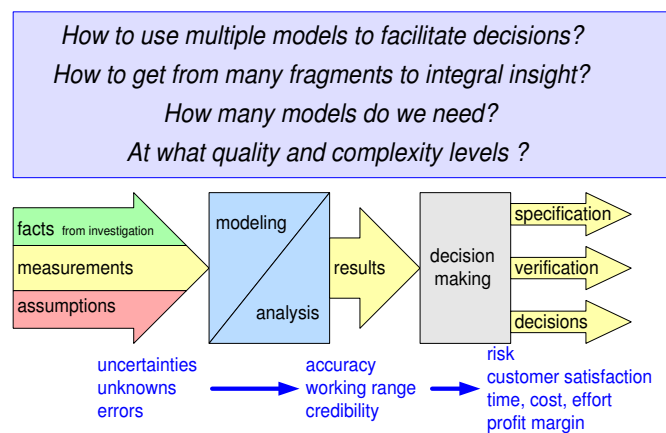


Figure 1.2: Purpose of Modeling

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 1.2, are *risk*, *customer satisfaction*, and also the rather tangible factors as time, effort, cost and profit margin.

Figure 1.2 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 [1].

1.2 From Chaos to Order

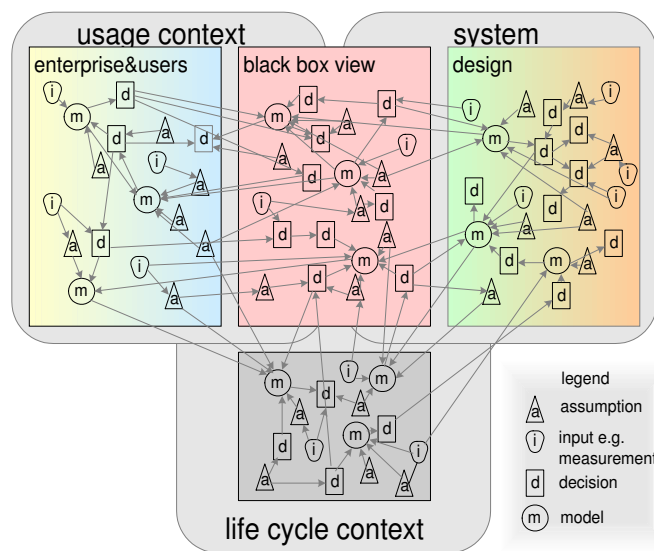


Figure 1.3: Graph of Decisions and Models

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 1.3. 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.

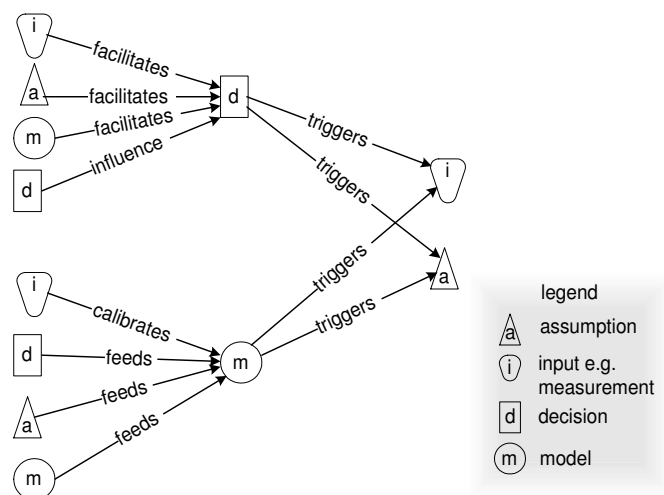


Figure 1.4: Relations: Decisions, Models, Inputs and Assumptions

These four types of information are related, as shown in Figure 1.4. 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 1.3.

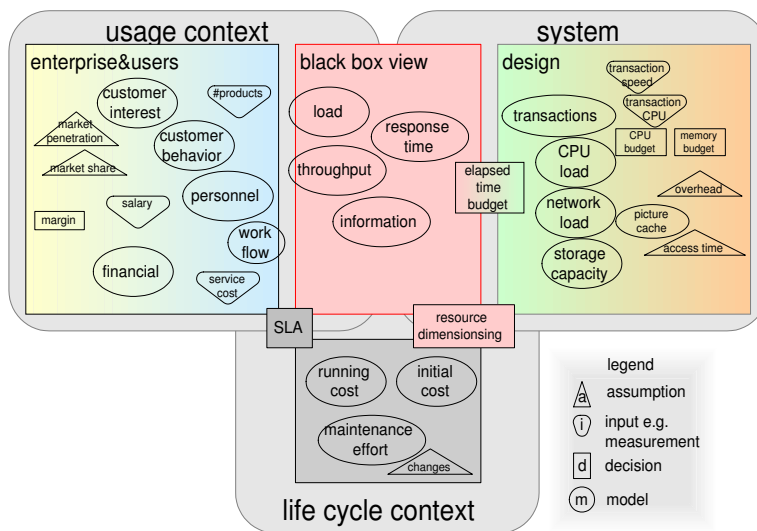


Figure 1.5: Example Graph for Web Shop

A somewhat less abstract graph is shown in Figure 1.5 for the web shop example. In this figure the relations have been left out, because the diagram is already overcrowded as is.

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.

1.3 Approach to Reason with Multiple Models

An overview of the stepwise approach is shown in Figure 1.6. 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.

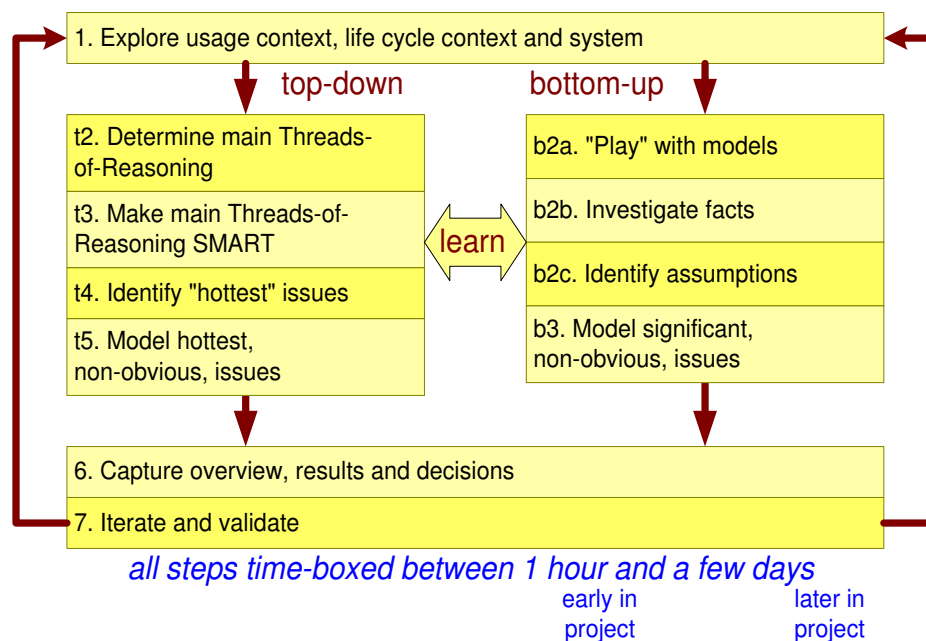


Figure 1.6: Reasoning Approach

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. Modeling is supportive to the processes of specification, verification and decision making. Step 1, shown in Figure 1.7 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 1.8. Threads-of-reasoning provide a means to relate business needs and concerns to specification to detailed realization decisions. This method is described in [4]. The most relevant individual chapters can be downloaded separately at: <http://www.gaudisite.nl/ThreadsOfReasoningPaper.pdf> and <http://www.gaudisite.nl/MIthreadsOfReasoningPaper.pdf>.

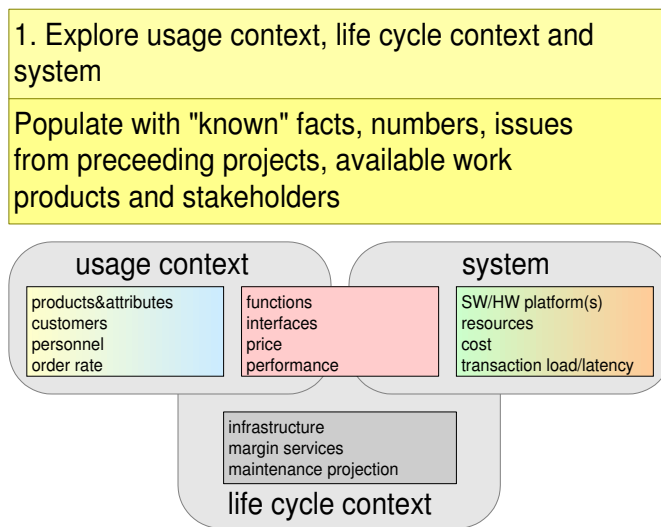


Figure 1.7: 1. Explore

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 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 1.9, 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 1.10 explains a frequently used acronym: SMART [2]. This acronym is used amongst others as checklist for the formulation of requirements. There

t2. Determine main Threads-of-Reasoning
 Architecting and System Design
 e.g. <http://www.gaudisite.nl/ModuleTORSides.pdf>

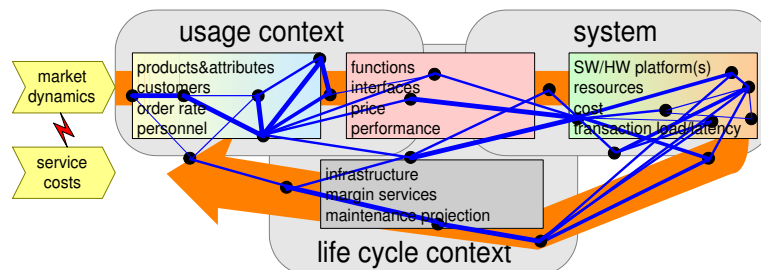


Figure 1.8: t2. Thread-of-Reasoning

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.

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 1.10 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

t3. Make main Threads-of-Reasoning SMART
 Quantify in terms of Key Performance Indicators, Key Performance Measures, Critical Resources

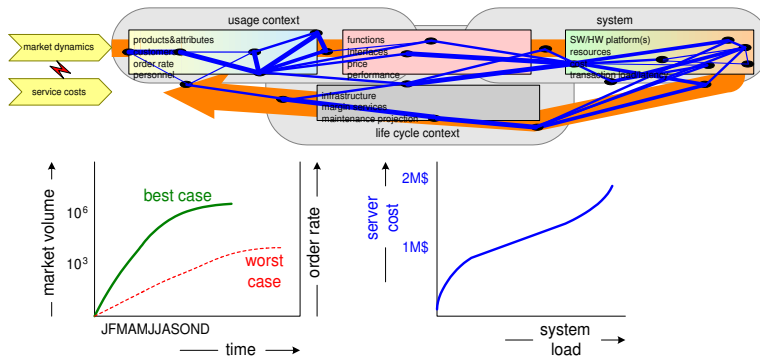


Figure 1.9: t3. SMART'en Thread-of-Reasoning

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 5 = hot. For example, for risks a small risk = 1, a large risk = 5. The table at the bottom of Figure 1.11 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 perspec-

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

Figure 1.10: 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 1.11: t4: Identify Hottest

tives. 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 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 1.12 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

t5. Model hottest, non-obvious, issues

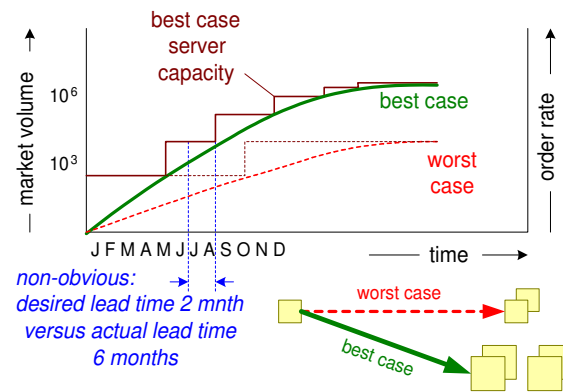


Figure 1.12: t5: Model Hottest Issues

one month. These simple models allow for further modeling of income, cost and margin.

b2a. "Play" with models	b2b. Investigate facts	b2c. Identify assumptions
vary inputs vary model structure to understand <i>model applicability</i> , <i>design quality</i> and <i>specification feasibility</i>	market research measurements preceding systems micro benchmarks literature, supplier info	What is the basis for model structure, design decision, specification, quantification et cetera? <i>Most assumptions are implicit and hidden!</i>

$$n_{CPU} = t_{required\ total} / t_{1\ CPU}$$

$$t_{required\ total} = n_{transactions} * t_{1\ transaction} + t_{other}$$

$$t_{1\ transaction} = 1\ ms\ (on\ 1\ CPU)$$

http://www.tpc.org/tpcc/results/tpcc_perf_results.asp

IBM System p5 595
 TPC-C Throughput 4,033,378
 Total # of Processors: 32
 Total # of Cores: 64
 $1/t_{1\ transaction} = 4 * 10^6 / 60 / 64$
min to sec / # cores
 $t_{1\ transaction} \approx 1\ ms$

server load dominated by transactions
 transaction load scales linear
 TPC-C is representative
 what is the effect of other TPC-C workload?

Figure 1.13: b2abc: Bottom-up

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 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 1.13 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 1.13 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 1.13 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.

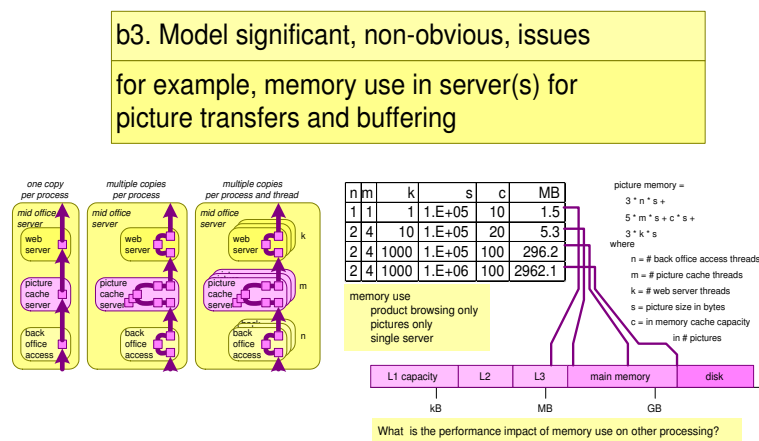


Figure 1.14: b3: Model Significant Issues

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 directly evident what the effect of change is, should be modeled. Figure 1.14

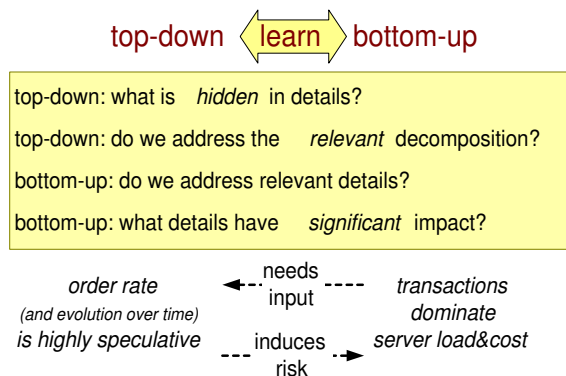


Figure 1.15: Learning Concurrent Bottom-up and Top-down

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 1.15 shows as mutually related questions the top-down uncertainty about the sales volume and the bottom-up impact of transactions on server load.

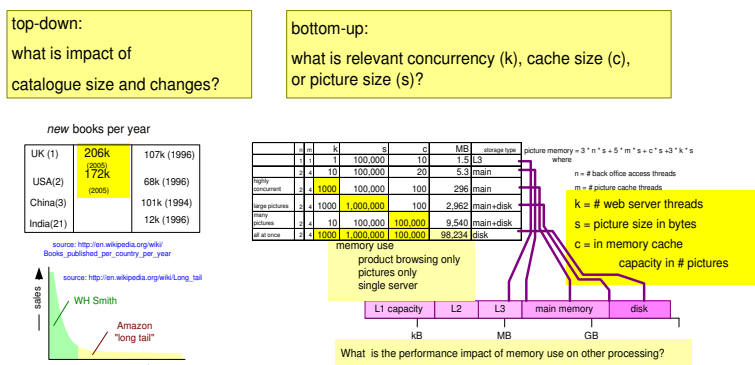


Figure 1.16: Example top-down and bottom-up

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.

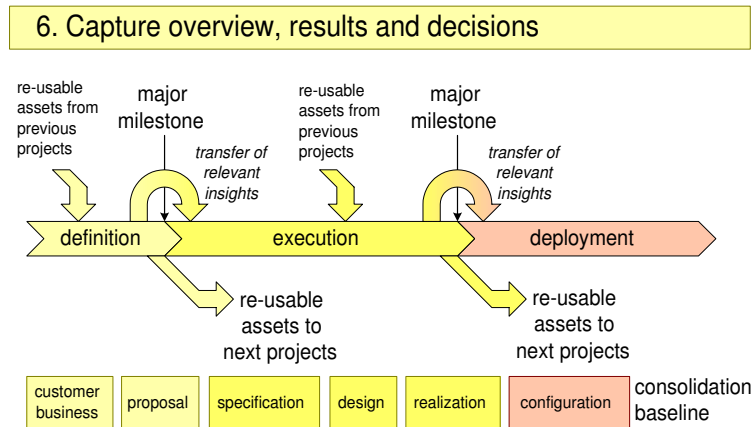


Figure 1.17: 6. Capture overview, results and decisions

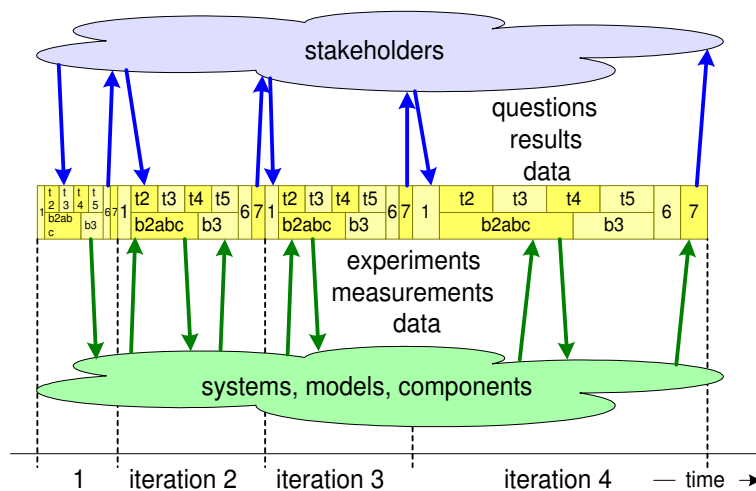


Figure 1.18: 7. Iterate and Validate

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

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 and the interaction with systems and components is shown in Figure 1.18

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.

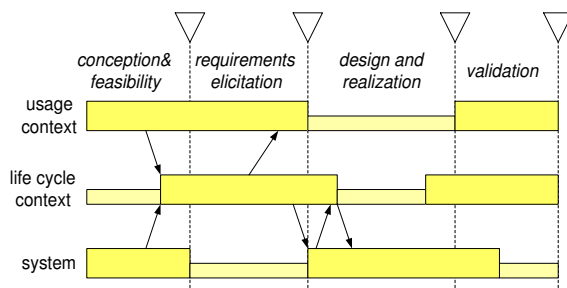


Figure 1.19: The focus is shifting during the project

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 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).

1.4 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 1.20 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).

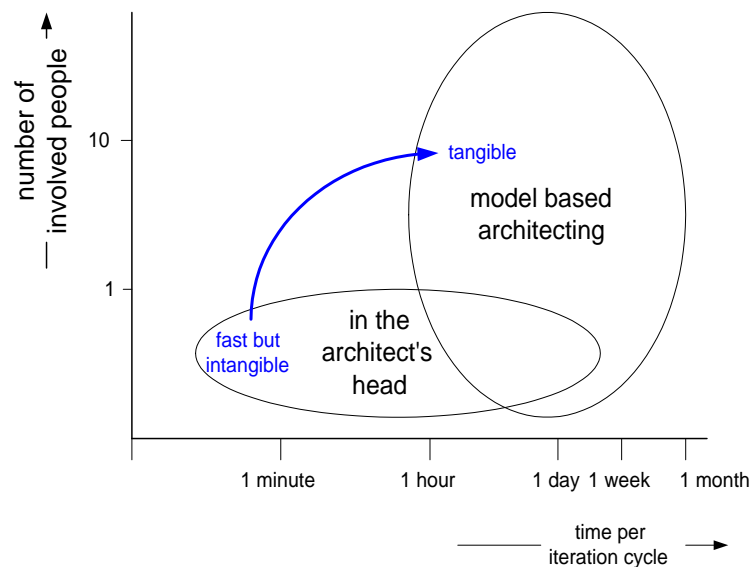


Figure 1.20: Models Support Communication

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 1.21 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

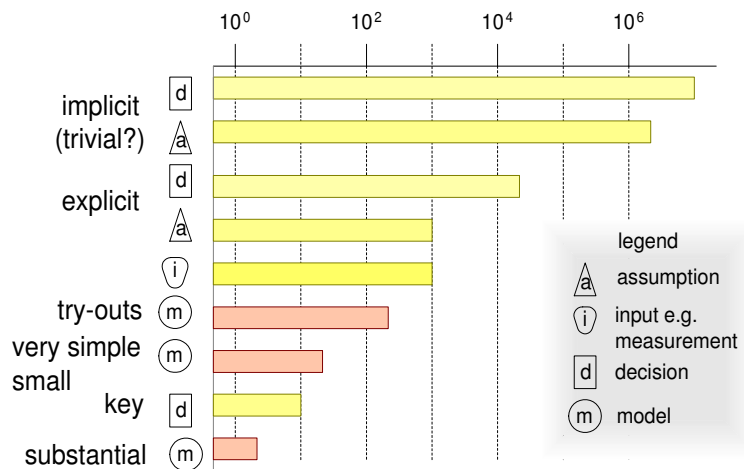


Figure 1.21: Frequency of Assumptions, Decisions and Modeling

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.

1.5 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 1.22.

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*

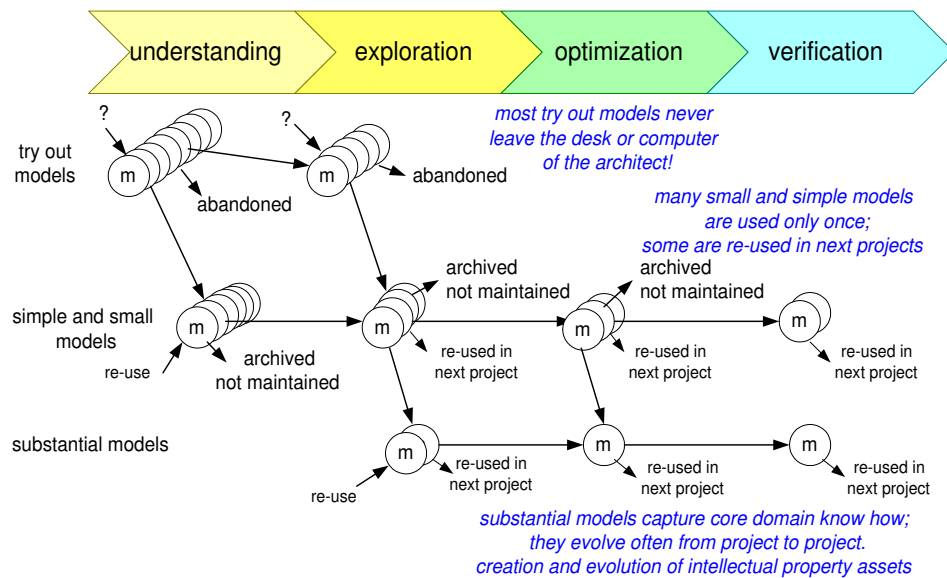


Figure 1.22: Life Cycle of Models

models in Figure 1.22, 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, 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.

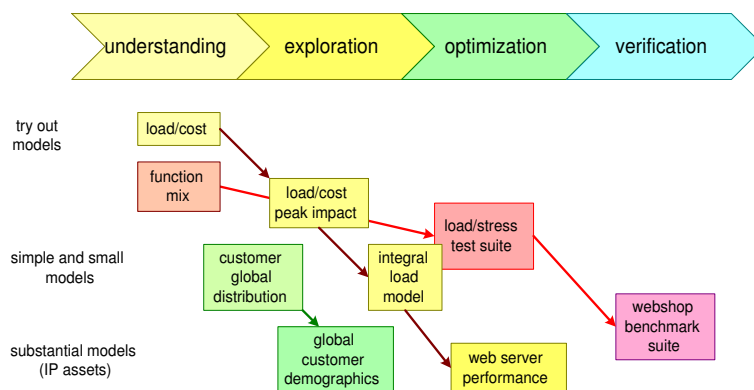


Figure 1.23: Examples of Life Cycle of Models

Figure 1.23 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*.

1.6 Summary

<i>Conclusions</i>
Top-down and bottom-up provide complementary insights
Key words for selection: hottest, non-obvious, significant, relevant
Multiple small models are used in combination
Some models evolve from very simple to more substantial

<i>Techniques, Models, Heuristics of this module</i>
Threads-of-reasoning
SMART
Key Performance Indicators, Key Performance Measures, Critical Resources
Ranking matrices

Figure 1.24: summary of this paper

Figure 1.24 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.

We have used several techniques:

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

1.7 Acknowledgements

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

Bibliography

- [1] Luiz André Barroso, Jeffrey Dean, and Urs Holzle. 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.
- [2] 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.
- [3] Gerrit Muller. The system architecture homepage. <http://www.gaudisite.nl/index.html>, 1999.
- [4] Gerrit Muller. CAFCR: A multi-view method for embedded systems architecting; balancing genericity and specificity. <http://www.gaudisite.nl/ThesisBook.pdf>, 2004.

History

Version: 0.3, date: 6 March, 2007 changed by: Gerrit Muller

- added position slide

Version: 0.2, date: 6 February, 2007 changed by: Gerrit Muller

- created reader

Version: 0.1, date: 17 January, 2007 changed by: Gerrit Muller

- added presentation Integration and Modularity

Version: 0, date: 16 January, 2007 changed by: Gerrit Muller

- created module