Abstract

Architecting System Performance applies and elaborates the course Architectural Reasoning Using Conceptual Modeling to architect performance of systems. We teach an architecting method based on many views and fast iteration of the views. Visual models, functional models, and mathematical models in all views are the means to communicate about the system, to discuss specification and design choices, to reason about consequences, and to make decisions.

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.

June 21, 2020
status: preliminary draft
version: 0.4
Abstract

Course overview of the course Architecting System Performance.
<table>
<thead>
<tr>
<th>Course Overview</th>
<th>time-oriented performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Course introduction</td>
<td>15. Measuring Performance</td>
</tr>
<tr>
<td>3. Course didactics</td>
<td>17. Greedy and Lazy Pattern</td>
</tr>
<tr>
<td>4. Connecting breadth and depth</td>
<td>18. Scheduling</td>
</tr>
<tr>
<td>5. Performance Modeling</td>
<td>19. Robust Performance</td>
</tr>
<tr>
<td>7. Visualizing Dynamic Behavior</td>
<td></td>
</tr>
<tr>
<td>8. Emerging Behaviour</td>
<td></td>
</tr>
<tr>
<td>9. Budgeting</td>
<td></td>
</tr>
<tr>
<td>10. Modeling Paradigms</td>
<td></td>
</tr>
<tr>
<td>11. Applications and Variations</td>
<td></td>
</tr>
<tr>
<td>12. Model Analysis</td>
<td></td>
</tr>
<tr>
<td>13. Reasoning Approach</td>
<td></td>
</tr>
<tr>
<td>14. Defining Performance</td>
<td></td>
</tr>
</tbody>
</table>

Architecting System Performance; Course Overview
Gerrit Muller

version: 0.3
June 21, 2020
ASPCOnuggets
## Assignments in Face-to-Face Module

<table>
<thead>
<tr>
<th>0. elevator case</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>supersystem</strong></td>
</tr>
<tr>
<td><strong>system</strong></td>
</tr>
<tr>
<td><strong>subsystem</strong></td>
</tr>
</tbody>
</table>

1. **sketch the problem**
   - goal use case
   - key performance parameters
   - main concepts
   - critical technologies

2. **make conceptual model of the current situation**
   - model dynamic behavior
   - model 0-order kpp using functions (as simple as possible)
   - quantify contribution to kpp using observed data

3. **explore customer and business relevance**
   - develop story
   - model workflow and performance
   - model customer value as function of kpp

4. **make conceptual model of potential solutions**
   - model the foreseen solution
   - model & compare 2 alternative solutions

5. **list questions and uncertainties, reformulate problem and goal, and formulate gaps and options**

6. **develop an elevator pitch to report you findings and recommendations to management**
Abstract

Listing the course material for Architecting System Performance
The ASP™ course is partially derived from the EXARCH course developed at Philips CTT by Ton Kostelijk and Gerrit Muller.

Extensions and additional slides have been developed at ESI by Teun Hendriks, Roland Mathijssen and Gerrit Muller.
### core

Physical Models of an Elevator  
[http://www.gaudisite.nl/info/ElevatorPhysicalModel.info.html](http://www.gaudisite.nl/info/ElevatorPhysicalModel.info.html)

### optional

Teaching conceptual modeling at multiple system levels using multiple views  

Understanding the human factor by making understandable visualizations  
[http://www.gaudisite.nl/info/UnderstandingHumanFactorVisualizations.info.html](http://www.gaudisite.nl/info/UnderstandingHumanFactorVisualizations.info.html)
### Course Didactics

#### Core

Architecting System Performance; Course Didactics

http://www.gaudisite.nl/info/ASPcourseDidactics.info.html

#### Optional

**DSRP:** [https://en.wikipedia.org/wiki/DSRP](https://en.wikipedia.org/wiki/DSRP)

**Assumptions:** “Systems Engineering and Critical Reflection: The Application of Brookfield and Goffman to the Common Experiences of Systems Engineers” by Chucks Madhav; proceedings of INCOSE 2016, in Edinburgh, GB

**70/20/10:**

- [http://charles-jennings.blogspot.nl/](http://charles-jennings.blogspot.nl/)


**Assumptions and beliefs:**

- [https://pivotalthinking.wordpress.com/tag/ladder-of-inference/](https://pivotalthinking.wordpress.com/tag/ladder-of-inference/)
Greedy and Lazy Patterns

**core**

Architecting System Performance; Greedy and Lazy Patterns
http://gaudisite.nl/info/ASPgreedyAndLazy.info.html

**optional**

Fundamentals of Technology
### Measuring

<table>
<thead>
<tr>
<th><strong>core</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Architecting System Performance; Measuring</td>
</tr>
<tr>
<td><a href="http://www.gaudisite.nl/info/ASPmeasuring.info.html">http://www.gaudisite.nl/info/ASPmeasuring.info.html</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>optional</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Performance Method Fundamentals</td>
</tr>
<tr>
<td>Measurement issues; From gathering numbers to gathering knowledge by Ton Kostelijk</td>
</tr>
<tr>
<td><a href="http://www.gaudisite.nl/MeasurementExecArchSlides.pdf">http://www.gaudisite.nl/MeasurementExecArchSlides.pdf</a></td>
</tr>
<tr>
<td>Modeling and Analysis: Measuring</td>
</tr>
<tr>
<td>Exploring an existing code base: measurements and instrumentation</td>
</tr>
<tr>
<td><a href="http://www.gaudisite.nl/info/ExploringByMeasuringInstrumenting.info.html">http://www.gaudisite.nl/info/ExploringByMeasuringInstrumenting.info.html</a></td>
</tr>
</tbody>
</table>
Abstract

This presentation presents the ideas behind the course Architecting System Performance. A number of frameworks and mental models show the context of this course and the approach to performance advocated in this course.
Performance Playing Field

consumer experience
enterprise performance: enterprise productivity
enterprise throughput
enterprise response time

Performance attributes require means for analysis evaluation, and creation of structure (parts and interfaces) and dynamic behavior (functions) at all levels.

Hence, we need conceptual modeling at all levels.

competitiveness
service response time
service cost

system performance:
- system productivity
- system throughput
- system response time

technical concepts for:
- resource management
- internal logistics
- processing

Architecting System Performance; Managing System Performance

version: 0.2
June 21, 2020

Gerrit Muller
What and Why to Model

- How well is the customer served?
- How credible becomes the solution?
- How much are time and effort reduced?
- How much is the risk reduced?
- How much is the solution improved?

Purpose and type of model depend on project life cycle

- Type of model and views depend on purpose

Customer
- Key drivers
- Risks

Modeling
- Feasibility
- Communication
- Risk mitigation
- Exploration
- Validation

Business as usual
- (No modeling)
- Obvious
- Historic data
- Competitive data

Business
- Key drivers
- Risks

Decision factors:
- Accuracy of model
- Credibility of results
- Level of abstraction
- Working range
- Calibration of model
- Robustness of model
- Time to first results and feedback
- Effort
- Evolvability (adaptation to new questions)

How much effort is needed to create model(s)?
- How much effort is needed to use and maintain model(s)?
- How much time is needed to obtain useful result?
Modeling Evolves over Time

understanding | exploration | optimization | verification

project phase \(\rightarrow\) purpose of the model \(\rightarrow\) type of the model

determines \(\rightarrow\) determines \(\rightarrow\)

Architecting System Performance; Managing System Performance

version: 0.2
June 21, 2020
ASPCOwhyModeling
The Modeler’s Mindset Evolves too

- **Understanding**
- **Exploration**
- **Optimization**
- **Verification**

**Explorative**
- what is needed?
- what can be achieved?

**Defensive**
- what are the risks?
- will the system perform well?
- how to mitigate shortcomings?
The Architect Can Be "Out of Phase"

understanding → exploration → optimization → verification

Explorative
what is needed?
what can be achieved?

Defensive
what are the risks?
will the system perform well?
how to mitigate shortcomings?

"look ahead"

mindset of most stakeholders
mindset of architect

Architecting System Performance; Managing System Performance
version: 0.2
June 21, 2020
ASPC1OutOfPhase
10 Fundamental Recommendations

**principles**
- use feedback
- work incremental
- work evolutionary
- be explicit
- make issues tangible

**objectives**
- support communication
- facilitate reasoning
- support decision making
- create understanding
- maintain insight

**recommendations**
- Time-box
- Iterate
- Quantify early
- Measure and validate
- Multiple levels of abstraction
- (Simple) mathematical models
- Analysis of accuracy and credibility
- Multi-view
- System and its context
- Visualize

help to achieve

translate into

translate into

$
\text{Architecting System Performance; Managing System Performance}
$

version: 0.2

MAOrecommendations
Iterative Performance Management during Development

- measure
- evaluate
- analyse

- determine most important and critical requirements
- model
- analyse constraints and design options

- simulate
- build proto
Abstract

The didactics behind a course like Architecting System Performance is a challenge, because the learning goals relate mostly to attitude and ways of thinking. At the same time, the material covers methods, techniques, tools, and concepts, which may lure participants in mechanistic approaches. Core in the didactic approach is reflection. This presentation offers some ”thinking models” to assist reflection.
Competence Requires Various Learning Styles

- **what**
  - Knowledge
  - Skills
  - Ability
  - Attitude

- **how**
  - Lecturing
  - Exercises
  - Assignments
  - Practice
  - Coaching
  - Reflection

- **who**
  - Participant
  - Teacher/coach

Architecting System Performance; Course Didactics

version: 0.1
June 21, 2020
AACL.competenceProgram
Bloom’s Taxonomy and Higher Order Thinking Skills

Higher Order Thinking Skills more difficult to teach more valuable takes time to develop
must be mastered before, however when missing can be acquired fast

Lower Order Thinking Skills people can acquire them fast

remembering
understanding
applying
analyzing
evaluating
creating

Higher Order Thinking Skills

Version: 0.1
June 21, 2020
ASPCDbloomsTaxonomy
Course Assumption:

This course focuses on Higher Order Thinking Skills.

We assume

that you have appropriate knowledge

and

that you are able to find and absorb

required specific knowledge fast.
Problem-Based Learning Using Reflection

source: Kolb's learning cycle
http://www.infed.org/biblio/b-explrn.htm
Role of Experience in Learning

70:20:10 learning model

70: Experience

20: Exposure

10: Education

DSRP Model

- Making **Distinctions**
  - A
  - not A
  - identity
  - Distinction
  - other

- Organizing **Systems**

- Recognizing **Relationships**

- Taking **Perspectives**
Separate Reflection Wall

mental switch from problem/system to “meta” how, what, why?

reflection wall

flips team 4

flips team 2

flips team 3

flips team 1

Architecting System Performance; Course Didactics
version: 0.1
June 21, 2020
Gerrit Muller
Scope and Topic of Reflection

operational or life cycle context
  system of interest
    component or function of interest

organization
  project
    team
      individual

principle
  process or method
    procedure or technique
      tool or notation

*technical*  *psychosocial*  *means*
The Role of Assumptions and Beliefs in Thinking

The “Ladder of Inference” originally proposed by Chris Argyris and developed by Peter Senge and his colleagues [The Fifth Discipline Fieldbook] illustrates how these biases can be built into our thinking. https://pivotalthinking.wordpress.com/tag/ladder-of-inference/

Architecting System Performance; Course Didactics

version: 0.1
June 21, 2020
ASPCDladderOfInference
Architecting System Performance; Connecting Breadth and Depth

by Gerrit Muller    TNO-ESI, University College of South East Norway

    e-mail: gaudisite@gmail.com
    www.gaudisite.nl

Abstract

System Performance plays a crucial role in the customer value proposition and the business proposition. Minor details deep down into the system may have a large impact on system performance, and hence on both value propositions. Challenge in architecting system performance is to connect both worlds, which are mentally far apart.
Organizational Problem: Disconnect

What does Customer need in Product and Why?

- Customer objectives
- Application
- Functional
- Conceptual
- Realisation

How can the product be realized

What are the critical decisions

system requirements
design decisions
parts connections
lines of code
and growing every year...

Architect: Connecting Problem and Technical Solution

**What** does Customer need in Product and **Why**?

- **Customer objectives**
- **Application**
- **Functional**
- **Conceptual**
- **Realisation**

**How** can the product be realized

**What** are the critical decisions

1. Number of details and growing every year
2. System requirements
3. Design decisions
4. Parts
5. Connections
6. Lines of code

Architecting System Performance; Connecting Breadth and Depth
Major Bottleneck: Mental Dynamic Range

Architecting System Performance; Connecting Breadth and Depth

version: 0
June 21, 2020

RATWmentalDynamicRange
Breadth

System of Interest
- supporting systems
- train plan maintain
- surrounding systems
  - supply receive manage
- stakeholders concerns needs interests
- regulations processes procedures

supporting systems
- train plan • • maintain
Devilish details in design space may have large impact on performance. Many detailed design decisions determine system performance.
Abstract

Principles and concepts of modeling performance.
Empirical versus First Principle Models

**Empirical** model: a model based on observations and measurements.

An empirical model *describes* the observations.

An empirical model provides **no understanding**.

**First principle** model: a model based on theoretical principles.

A first principle model *explains* the desired property from first principles from the laws of physics.

A first principle model requires **values** for incoming parameters to calculate results.

#### Empirical Model

- \( t_{\text{move}} = a \cdot n + b \)

#### First Principle Model

- Position in case of uniform acceleration:
  \[
  S_t = S_0 + v_0 t + \frac{1}{2} a_0 t^2
  \]
  \[
  t_{\text{top floor}} = t_a + t_v + t_a
  \]
  \[
  t_a = \frac{v_{\text{max}}}{a_{\text{max}}}
  \]
  \[
  S(t_a) = \frac{1}{2} \cdot a_{\text{max}} \cdot t_a^2
  \]
  \[
  S_{\text{linear}} = S_{\text{top floor}} - 2 \cdot S(t_a)
  \]
  \[
  t_v = \frac{S_{\text{linear}}}{v_{\text{max}}}
  \]
Conceptual model: a model explaining observations and measurements using a selection of first principles.

A conceptual model is a hybrid of empirical and first principle models; simple enough to understand and to reason, realistic enough to make sense.
<table>
<thead>
<tr>
<th>Order</th>
<th>Description</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>0&lt;sup&gt;th&lt;/sup&gt;</td>
<td>Main function</td>
<td>$t_{\text{top floor}} = \frac{S_{\text{top floor}}}{v_{\text{max}}}$</td>
</tr>
<tr>
<td>1&lt;sup&gt;st&lt;/sup&gt;</td>
<td>Add most significant secondary contributions</td>
<td>$t_{\text{top floor}} = \frac{S_{\text{top floor}}}{v_{\text{max}}} - \frac{a_{\text{max}}}{v_{\text{max}}} \cdot t_a^2 + \frac{2 \cdot v_{\text{max}}}{a_{\text{max}}} + \frac{2 \cdot a_{\text{max}}}{j_{\text{max}}}$</td>
</tr>
<tr>
<td>2&lt;sup&gt;nd&lt;/sup&gt;</td>
<td>Add next level of contributions</td>
<td>$t_{\text{top floor}} \sim \frac{S_{\text{top floor}}}{v_{\text{max}}} - \frac{a_{\text{max}}}{v_{\text{max}}} \cdot t_a^2 + \frac{2 \cdot v_{\text{max}}}{a_{\text{max}}} + \frac{2 \cdot a_{\text{max}}}{j_{\text{max}}}$</td>
</tr>
</tbody>
</table>
Abstract

A recurring question in modeling and performance analysis is when to stop digging. What level of detail is needed to achieve acceptable performance? What level of abstraction result in credible and sufficiently accurate results? How to cope with many levels of abstraction?
Level of Abstraction Single System

- **system requirements**
- **multidisciplinary design**
- **static system definition**
  - **monodisciplinary**

The level of abstraction is represented by the number of details, which increases exponentially from 10^0 to 10^7.

Architecting System Performance; Level of Abstraction

version: 0
June 21, 2020
RAPpyramid

Gerrit Muller
From system to Product Family or Portfolio

Architecting System Performance; Level of Abstraction

version: 0
June 21, 2020

DRALpyramidGrowth

system → portfolio

increase

system
multidisciplinary
monodisciplinary

systems
multidisciplinary
monodisciplinary

number of details

10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9

10^0
10^1
10^2
10^3
10^4
10^5
10^6
10^7
10^8
10^9
Product Family in Context

- enterprise context
- enterprise
- stakeholders
- systems
- multidisciplinary design
- parts, connections, lines of code

Number of details:
- $10^9$
- $10^6$
- $10^3$
- $10^0$
- $10^3$
- $10^6$
- $10^9$
The seemingly random exploration path

thinking path of an architect during a few minutes up to 1 day
Coverage of problem and solution space

covered or touched by architects

subjects

covered by engineers and experts
Many Levels of Abstraction

Architecting System Performance; Level of Abstraction

version: 0
June 21, 2020
ASPLAlevels

key performance
performance definition
elaborated use cases
performance models
budgets and measurements
component designs
killing details

simplification, abstraction

system designs
Fidelity Properties

- Low fidelity
  - Low effort
  - Fast
  - What fidelity is needed for: planning, training, validation, design exploration?
  - What configurations do we need?
  - What can we afford?

- High fidelity
  - Large effort
  - Slow

Architecting System Performance; Level of Abstraction

version: 0
June 21, 2020
ASPLAproperties
Visualizing Dynamic Behavior

by Gerrit Muller       TNO-ESI, University College of South East Norway
                        e-mail: gaudisite@gmail.com
                        www.gaudisite.nl

Abstract

Dynamic behavior manifests itself in many ways. Architects need multiple complementary visualizations to capture dynamic behavior effectively. Examples are capturing information, material, or energy flow, state, time, interaction, or communication.
Example Functional Model of Information Flow

1. **Get sensor data**
   - Transform into image
   - Fuse sensor images
   - Detect objects
   - Classify objects
   - Update world model

2. **Get GPS data**
   - Calculate GPS location
   - Estimate location
   - Update location

3. **Get goal trajectory**
   - Analyze situation
   - Determine next step

4. **Get external data**
   - Visualizing Dynamic Behavior

5. Version: 0
   - June 21, 2020

6. BSEARfunctionalModel
Workflow as Functional Model

1. assembly, functional test
2. run EDP/LRP
3. run risers
4. hook up SFT and TF
5. move above well
6. ROV assisted connect
7. hook up coil tubing and wireline BOP
8. system function and connection seal test
9. run coil tubing and wireline
10. perform workover operations
11. retrieve coil tubing and wireline BOP
12. unhook coil tubing and wireline BOP

Visualizing Dynamic Behavior

version: 0
June 21, 2020
SSMESH2019TypicalWorkoverOperation
Workflow as Timeline

assumptions:
running and retrieving risers: 50m/hr
running and retrieving coiled tubing/wireline: 100m/hr
depth: 300m
Example Signal Waveforms

imaging = repeating similar pattern many times

typical TE: 5..50 ms

Gy=0 Gy=127

Gz
Gx
Gy
RF
TE
TR

transmit
receive
Example Time Line with Functional Model

Functional flow:
- Call family doctor
- Visit family doctor
- Call neurology department
- Visit neurologist
- Call radiology department
- Examination itself
- Diagnosis by radiologist
- Report from radiologist to neurologist
- Visit neurologist

Days:
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10
- 11
- 12
- 13
- 14
- 15
- 16
- 17
- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25
Flow of Light (Physics)

- Laser
- Illuminator
- Sensor
- Pulse-freq, bw, wavelength, ...
- Uniformity
- Lens
- Wafer
- Reticle
- Aerial image
- NA
- Abberations
- Transmission

Visualizing Dynamic Behavior
Gerrit Muller

version: 0
June 21, 2020
TSAIT physicsView
Dynamic Behavior is Multi-Dimensional

How does the system work and operate?

Functions describe *what* rather than *how*.

Functions are *verbs*.

Input-Process-Output paradigm.

Multiple kinds of flows:

- physical (e.g. hydrocarbons, goods, energy)
- information (e.g. measurements, signals)
- control

Time, events, cause and effect

Concurrency, synchronization, communication

multi-dimensional information and dynamic behavior
Abstract

The essence of a system is that the parts together can do more than the separate parts. The interaction of the parts results in behavior and properties that cannot be seen as belonging to individual parts. We call this type of behavior “emerging behavior”.

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.
Emergence is Normal and Everywhere

emergent behavior and properties =

function of

**dynamic interaction** between

**parts** in the system and

**context** of the system

examples

- flying and stalling of an airplane
- Tacoma bridge resonance
Emergence, Desire, and Foreseeing

- **Foreseen**
  - **Mitigated**
    - Goal of design

- **Foreseen, but underestimated**
  - **Risk**
    - Side-effect

- **Unforeseen**
  - **Risk**
    - Undesired
  - **Side-effect**
    - Desired
Abstract

This presentation addresses the fundamentals of budgeting: What is a budget, how to create and use a budget, what types of budgets are there. What is the relation with modeling and measuring.
content of this presentation

What and why of a budget

How to create a budget (decomposition, granularity, inputs)

How to use a budget
What is a Budget?

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*
Why Budgets?

- 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
Visualization of Budget Based Design Flow

- **Modeling and Analysis: Budgeting**

  - **Version:** 1.0
  - **Date:** June 21, 2020

  - **Microbenchmarks:**
    - Aggregated functions
    - Applications

  - **Existing System:**
    - Measurements

  - **Design Estimates:**
    - Simulations

  - **Budget:**
    - Specifications
    - Feedback

  - **Tuning:**
    - New (proto) system

  - **Can be more complex than additions:**
    - Model
    - Feedback

  - **Modeling and Analysis:**
    - Budgeting

  - **Version:** 1.0
  - **Date:** June 21, 2020

  - **Microbenchmarks:**
    - Aggregated functions
    - Applications

  - **Existing System:**
    - Measurements

  - **Design Estimates:**
    - Simulations

  - **Budget:**
    - Specifications
    - Feedback

  - **Tuning:**
    - New (proto) system

  - **Can be more complex than additions:**
    - Model
    - Feedback

  - **Modeling and Analysis:**
    - Budgeting

  - **Version:** 1.0
  - **Date:** June 21, 2020

  - **Microbenchmarks:**
    - Aggregated functions
    - Applications

  - **Existing System:**
    - Measurements

  - **Design Estimates:**
    - Simulations

  - **Budget:**
    - Specifications
    - Feedback

  - **Tuning:**
    - New (proto) system

  - **Can be more complex than additions:**
    - Model
    - Feedback
## Stepwise Budget Based Design Flow

<table>
<thead>
<tr>
<th>Step</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>1A</td>
<td>Measure old systems</td>
<td>micro-benchmarks, aggregated functions, applications</td>
</tr>
<tr>
<td>1B</td>
<td>Model the performance starting with old systems</td>
<td>flow model and analytical model</td>
</tr>
<tr>
<td>1C</td>
<td>Determine requirements for new system</td>
<td>response time or throughput</td>
</tr>
<tr>
<td>2</td>
<td>Make a design for the new system</td>
<td>explore design space, estimate and simulate</td>
</tr>
<tr>
<td>3</td>
<td>Make a budget for the new system:</td>
<td>models provide the structure, measurements and estimates provide initial numbers, specification provides bottom line</td>
</tr>
<tr>
<td>4</td>
<td>Measure prototypes and new system</td>
<td>micro-benchmarks, aggregated functions, applications profiles, traces</td>
</tr>
<tr>
<td>5</td>
<td>Iterate steps 1B to 4</td>
<td></td>
</tr>
</tbody>
</table>
Budgets Applied on Waferstepper Overlay

- **process overlay 80 nm**
- **matched machine 60 nm**
- **single machine 30 nm**
- **stage overlay 12 nm**
- **position accuracy 7 nm**
- **frame stability 2.5 nm**
- **metrology stability 5 nm**
- **system adjustment accuracy 2 nm**
- **alignment repro 5 nm**
- **stage grid accuracy 5 nm**
- **local alignment accuracy 6 nm**
- **global alignment accuracy 6 nm**
- **off axis pos. meas. accuracy 4 nm**
- **stage Al. pos. meas. accuracy 4 nm**
- **blue align sensor repro 3 nm**
- **interferometer stability 1 nm**
- **tracking error WS 2 nm**
- **tracking error X, Y 2.5 nm**
- **tracking error RS 1 nm**
- **tracking error phi 75 nrad**

**Notes:**
- **process dependency sensor 5 nm**
- **lens matching 25 nm**
- **matching accuracy 5 nm**
- **off axis pos. Sensor repro 3 nm**
- **metrology stability 5 nm**
- **frame stability 2.5 nm**
## Budgets Applied on Medical Workstation Memory Use

<table>
<thead>
<tr>
<th>memory budget in Mbytes</th>
<th>code</th>
<th>obj data</th>
<th>bulk data</th>
<th>total</th>
</tr>
</thead>
<tbody>
<tr>
<td>shared code</td>
<td>11.0</td>
<td></td>
<td></td>
<td>11.0</td>
</tr>
<tr>
<td>User Interface process</td>
<td>0.3</td>
<td>3.0</td>
<td>12.0</td>
<td>15.3</td>
</tr>
<tr>
<td>database server</td>
<td>0.3</td>
<td>3.2</td>
<td>3.0</td>
<td>6.5</td>
</tr>
<tr>
<td>print server</td>
<td>0.3</td>
<td>1.2</td>
<td>9.0</td>
<td>10.5</td>
</tr>
<tr>
<td>optical storage server</td>
<td>0.3</td>
<td>2.0</td>
<td>1.0</td>
<td>3.3</td>
</tr>
<tr>
<td>communication server</td>
<td>0.3</td>
<td>2.0</td>
<td>4.0</td>
<td>6.3</td>
</tr>
<tr>
<td>UNIX commands</td>
<td>0.3</td>
<td>0.2</td>
<td>0</td>
<td>0.5</td>
</tr>
<tr>
<td>compute server</td>
<td>0.3</td>
<td>0.5</td>
<td>6.0</td>
<td>6.8</td>
</tr>
<tr>
<td>system monitor</td>
<td>0.3</td>
<td>0.5</td>
<td>0</td>
<td>0.8</td>
</tr>
</tbody>
</table>

| application SW total            | 13.4 | 12.6     | 35.0      | 61.0  |

| UNIX Solaris 2.x                |      |          |           | 10.0  |
| file cache                      |      |          |           | 3.0   |

| total                           |      |          |           | 74.0  |
Power Budget Visualization for Document Handler

scanning and feeder
procedé
paper path
power supplies
cooling
finisher
paper input module

UI and control

size proportional to power

Legend

physical layout
Evolution of Budget over Time

- fact finding through details
- aggregate to end-to-end performance
- search for appropriate abstraction level(s)

- from coarse guesstimate to reliable prediction
- from typical case to boundaries of requirement space
- from static understanding to dynamic understanding
- from steady state to initialization, state change and shut down
- from old system to prototype to actual implementation

---

**Time**

*start*  
*later*  
*only if needed*
Potential Applications of Budget based design

- resource use (CPU, memory, disk, bus, network)
- timing (response, latency, start up, shutdown)
- productivity (throughput, reliability)
- Image Quality parameters (contrast, SNR, deformation, overlay, DOF)
- cost, space, time
What kind of budget is required?

<table>
<thead>
<tr>
<th>Static</th>
<th>Dynamic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Typical Case</td>
<td>Worst Case</td>
</tr>
<tr>
<td>Global</td>
<td>Detailed</td>
</tr>
<tr>
<td>Approximate</td>
<td>Accurate</td>
</tr>
</tbody>
</table>

Is the budget based on wish, empirical data, extrapolation, educated guess, or expectation?
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!
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.
Abstract

The word modeling is used for a wide variety of modeling approaches. These approaches differ in purpose, level of detail, effort, stakeholders, degree of formaility, and tool support.
Human Thinking and Tools

- Number of details:
  - $10^9$
  - $10^6$
  - $10^3$
  - $10^0$
  - $10^3$
  - $10^6$
  - $10^9$

- Systems:
  - Multi-disciplinary design
  - Parts, connections, lines of code

- Stakeholders:
  - Enterprise

- Enterprise context

- Human overview

Tools to manage large amounts of information:
- E.g. Doors Core

Modeling and Analysis; Modeling Paradigms
82  Gerrit Muller
Formality Levels in Pyramids

- **mono-disciplinary**
  - less formal, communication-oriented
  - heterogeneous uncertainties, unknowns, variable backgrounds, concerns

- **multi-disciplinary**
  - well defined, repeatable, reusable, machine readable

- **generated/instantiated**
  - more formal, more rigorous

Number of details:
- $10^0$
- $10^1$
- $10^2$
- $10^3$
- $10^4$
- $10^5$
- $10^6$
- $10^7$
- $10^8$
- $10^9$
## Modeling Paradigms

<table>
<thead>
<tr>
<th><strong>paradigm</strong></th>
<th><strong>purpose</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptual system modeling</td>
<td>architecting understanding, evaluating, creating reasoning, communicating, decision making</td>
</tr>
<tr>
<td>SysML</td>
<td>formal capture of structure and behavior, integrating other tools, simulating</td>
</tr>
<tr>
<td>Design for 6 sigma</td>
<td>quality improvement in repeatable environments, black box oriented</td>
</tr>
<tr>
<td>Conceptual information modeling</td>
<td>understanding and formalizing information</td>
</tr>
<tr>
<td>Design Framework</td>
<td>capturing and tracing architecture decisions</td>
</tr>
<tr>
<td>Matlab</td>
<td>modeling and analyzing designs and algorithms, simulation and code generation</td>
</tr>
<tr>
<td>CAD</td>
<td>mechanical and electrical design, interoperates with dedicated analysis, e.g. thermal, structural</td>
</tr>
<tr>
<td>Formal specification and design (model checkers)</td>
<td>verification</td>
</tr>
</tbody>
</table>
Modeling and Analysis: Applications and Variations

by Gerrit Muller  TNO-ESI, University College of South East Norway

e-mail: gaudisite@gmail.com

www.gaudisite.nl

Abstract

Models are used for a wide variation of purposes. Stakeholders can get confused between “reality” and the virtual counterparts. In practice, many hybrids between ”real” and virtual systems exist. For example, planning and training systems using real algorithms and data, and physical systems using a world model for situation awareness.
Model Applications and Variations

- sales
- acquisition
- capability analysis

- development
- verification
- validation
- operation

- understanding
- exploration
- optimization
- test data
- comparison
- trouble shooting
- mission planning
- training
- health monitoring

- Systems of Systems
  apply all asynchronously

- evolvability
  all phases repeat
  with same needs

- in system
  situation awareness
  planning
  training
  health monitoring
Spectrum from Real to Virtual Systems

"real" world

- system of interest
- stakeholders
- mutually interacting
- subsystem
  - hardware component
  - software component

virtual world

- virtual world of interest
- stakeholders
- mutually interacting
- subsystem
  - hardware component
  - software component

virtual world: HIL

- virtual world of interest
- stakeholders
- mutually interacting
- subsystem
  - hardware component
  - software component

virtual world: SIL

- virtual world of interest
- stakeholders
- mutually interacting
- subsystem
  - hardware component
  - software component

"real" world; testing

- system of interest
- stakeholders
- mutually interacting
- subsystem
  - hardware component
  - software component

simulation in context

- environment
  - data
- stakeholders
- mutually interacting
- subsystem
  - hardware component
  - software component
Architecting for Variations

**variation dimensions**
- fidelity
- product/system performance
- functionality
- application
- model purpose
- exhaustiveness

**properties**
- time-performance
- accuracy
- build & update effort
- build & update time
- testing effort and time
- credibility
- applicability
- usability

**Impact**

**System Architecture**
- modularity
- variation design

**Model Architecture**
- modularity
- variation design

**Feed**

**Iterate**
Abstract

Models only get value when they are actively used. We will focus in this presentation on analysis aspects: accuracy, credibility, sensitivity, efficiency, robustness, reliability and scalability.
What Comes out of a Model

- varying inputs
- varying circumstances
- varying design options
- varying realizations
- specification changes
- and ripple through

- use cases
- worst case exceptions
- varying inputs
- varying circumstances
- varying design options
- varying realizations

- design
- understanding
- exploration
- optimization
- verification

- life cycle
- specification changes and ripple through
- change cases

- model(s)
  - accuracy
  - credibility
  - working range
  - working range
  - worst case behavior
  - exceptional behavior
  - sensitivity
  - robustness
  - efficiency
  - performance
  - reliability
  - scalability
  - other system qualities

- model
  - applicability

- design
  - quality

- specification
  - feasibility

- use cases
- worst case
- exceptions

- specification

- feasibility

- use cases
- worst case
- exceptions
- change cases
Applicability of the Model

+ $\varepsilon_1$
- $\varepsilon_2$

input
accuracy
credibility

measurements
assumptions
facts

abstraction

model(s)

accuracy
credibility
working range

model realization
credibility
propagation

usage context
specifications
designs
realizations
## How to Determine Applicability

**try out models**
be aware of accuracy, credibility and working range

### simple and small models

1. **Estimate accuracy of results**
   - based on most significant inaccuracies of inputs and assumed model propagation behavior

2. **Identify top 3 credibility risks**
   - identify biggest uncertainties in inputs, abstractions and realization

3. **Identify relevant working range risks**
   - identify required (critical) working ranges and compare with model working range

### substantial models

systematic analysis and documentation of accuracy, credibility and working range
Common Pitfalls

- Discrete events in continuous world
  - Discretization artefacts
    - E.g. Stepwise simulations

- (Too) systematic input data
  - Random data show different behavior
    - E.g. Memory fragmentation

- Fragile model
  - Small model change results in large shift in results

- Self fulfilling prophecy
  - Price erosions + Cost increase (inflation) -> Bankruptcy
Worst Case Questions

Which design assumptions have a big impact on system performance?

What are the worst cases for these assumptions?

How does the system behave in the worst case?

a. poor performance within spec

b. poor performance not within spec

c. failure -> reliability issue
### FMEA-like Analysis Techniques

<table>
<thead>
<tr>
<th>Safety</th>
<th>Hazard Analysis</th>
<th>Potential Hazards</th>
<th>Damage</th>
<th>Measures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliability</td>
<td>FMEA</td>
<td>Failure Modes</td>
<td>Effects</td>
<td>Measures</td>
</tr>
<tr>
<td>Security</td>
<td>Vulnerability Risks</td>
<td>Consequences</td>
<td>Impact, Effort, Time</td>
<td>Measures</td>
</tr>
<tr>
<td>Maintainability</td>
<td>Change Cases</td>
<td>Impact, Effort, Time</td>
<td>System Behavior</td>
<td>Decisions</td>
</tr>
<tr>
<td>Performance</td>
<td>Worst Cases</td>
<td>System Behavior</td>
<td>Decisions</td>
<td></td>
</tr>
</tbody>
</table>
Brainstorming Phases

wave 1: the obvious

wave 2: more of the same

wave 3: the exotic, but potentially important

don't stop too early with brainstorming!
Different Viewpoints for Analysis

usage context
- new product
e.g. WoW extension
- merger
- automated access

system
- new functions
- new interfaces
- new media
- new standards
- cache/memory trashing
- garbage collection
- critical sections
- local peak loads
- intermittent HW failure

life cycle context
- power failure
- network failure
- new SW release
- roll back to old SW release
Abstract

We make models to facilitate decision making. These decisions range from business decisions, such as Service Level Agreements, to requirements, and to detailed design decisions. The space of decisions is huge and heterogeneous. The proposed modeling approach is to use multiple small and simple models. In this paper we discuss how to reason by means of multiple models.
Purpose of Modeling

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?
Graph of Decisions and Models

usage context

enterprise & users

black box view

system

design

life cycle context

legend

assumption

input e.g. measurement

decision

model
Example Graph for Web Shop

usage context

enterprise\&users

- customer interest
- customer behavior
- personnel
- financial
- salary
- market penetration
- market share
- margin

life cycle context

- #products

system

black box view

- load
- response time
- throughput
- information
- elapsed time budget

design

- transactions
- CPU load
- network load
- storage capacity
- CPU budget
- memory budget
- overhead
- picture cache
- access time

resource dimensionsing

- running cost
- initial cost
- maintenance effort
- changes

SLA

legend

a - assumption
i - input e.g. measurement
d - decision
m - model

Modeling and Analysis: Reasoning Approach

version: 1.0

June 21, 2020

Gerrit Muller
Relations: Decisions, Models, Inputs and Assumptions

Modeling and Analysis: Reasoning Approach
102 Gerrit Muller

version: 1.0
June 21, 2020
MARErelations
1. Explore usage context, life cycle context and system

2. Determine main Threads-of-Reasoning

3. Make main Threads-of-Reasoning SMART

4. Identify "hottest" issues

5. Model hottest, non-obvious, issues

6. Capture overview, results and decisions

7. Iterate and validate

_all steps time-boxed between 1 hour and a few days_

early in project

later in project
Frequency of Assumptions, Decisions and Modeling

- Implicit (trivial?)
  - Assumption
    - Very simple
    - Small
  - Decision
    - Key
    - Substantial
  - Model
    - Try-outs

- Explicit
  - Assumption
  - Input e.g.
  - Measurement
  - Decision
  - Model

Legend:
- Assumption (a)
- Input e.g. (i)
- Decision (d)
- Model (m)
Life Cycle of Models

- **Understanding**
  - Try out models
  - Simple and small models: archived, not maintained
  - Substantial models: re-used in next project

- **Exploration**
  - Abandoned
  - Simple and small models: archived, not maintained
  - Substantial models: re-used in next project

- **Optimization**
  - Archived, not maintained
  - Simple and small models: archived, not maintained
  - Substantial models: re-used in next project

- **Verification**
  - Most try out models never leave the desk or computer of the architect!
  - Many small and simple models are used only once; some are re-used in next projects
  - Substantial models capture core domain knowledge; they evolve often from project to project.

Creation and evolution of intellectual property assets

---

Modeling and Analysis: Reasoning Approach
105   Gerrit Muller

version: 1.0
June 21, 2020
MAREmodelLifeCycle
Examples of Life Cycle of Models

understanding exploration optimization verification

try out models

load/cost

function mix

load/cost peak impact

load/stress test suite

integral load model

web server performance

webshop benchmark suite

customer global distribution

global customer demographics

simple and small models

substantial models (IP assets)
Abstract

Performance is a broad term. Each domain has its own key performance parameters. Performance can be used to indicate time-oriented performance, such as response time, throughput, or productivity. However, more broadly, it may be used for aspects like image quality, spatial performance (f.i. positioning accuracy), energy or power properties, sensitivity and specificity of algorithms, or reliability and availability.
Performance Attributes

**time-oriented**
- response time
- latency
- throughput
- productivity

**spatial**
- positioning accuracy
- working envelope
- range
- turning cycle

**reliability**
- MTBF
- MTTR
- uptime
- unscheduled breaks

**energy/power**
- energy consumption
- range
- standby time
- maximum power
- heat release
- cooling capacity

**algorithmic**
- sensitivity
- specificity
- accuracy
- coverage

**image quality**
- sharpness
- contrast
- color consistency
- color rendition
- streakiness
- uniformity
Defining Performance

performance is a function of:
- context
- perception depends on individual human characteristics
- circumstances scenario
- operation of interest use case¹
- system of interest
- specification generic, valid for the class of systems normal and special cases (worst case, degraded, exceptions, …)
- design
- configuration
- version instance specific
- history

¹a use case in this context is rich (includes quantifications) and broad (covers the operation of interest, not a single function)
Example EV Range Definition

Electric Vehicle Driving Range

Range = f(
v(t),
Circumstances,
Driving style,
Car load,
Charging state,
Battery age)

A quantified Use Case defines under what circumstances the EV will achieve the specified range.

Published under GFDL, thanks to Orzetto
The **end-to-end** performance is the relevant performance as the **stakeholder** experiences it: from **initial trigger** to **final result**.

\[ t_{end-to-end} = t_{human\ activities} + t_{wait} + t_{elevator\ handling} + t_{move} \]

The end-to-end time consists of:
- **press button**
- **walk in**
- **walk out**
- nett moving time
- nett elevator time
- arrive at destination floor

**Architecting System Performance; Defining Performance**

version: 0.1
June 21, 2020
MAPEndToEnd
Abstract

Measuring is an essential part of architecting performance. Measurements provide quantified insight in actual behavior and performance. In this presentation, we discuss measuring, benchmarking, and instrumentation.
Performance Attributes in the Benchmark Stack

- CPU
- cache
- memory
- bus
- typical values
- interference
- variation
- boundaries

end-to-end function

network transfer

services

interrupts
task switches
OS services
CPU time
footprint
cache
applications

duration

services

interrupts
task switches
OS services

operating system

network transfer
database access
database query
services/functions

duration

CPU time

footprint

interrupts

task switches

OS services

(c Computing) hardware

locality
density
efficiency

overhead

tools
Performance as Function of the Layers

system performance = f(operating system, services, applications, hardware, tools)

how much does it cost?

what is used?

how often?
### Example $\mu$Benchmarks for Software

<table>
<thead>
<tr>
<th>Category</th>
<th>Infrequent Operations, Often Time-intensive</th>
<th>Often Repeated Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Database</strong></td>
<td>start session, finish session</td>
<td>perform transaction, query</td>
</tr>
<tr>
<td><strong>Network, I/O</strong></td>
<td>open connection, close connection</td>
<td>transfer data</td>
</tr>
<tr>
<td><strong>High Level Construction</strong></td>
<td>component creation, component destruction</td>
<td>method invocation, same scope, other context</td>
</tr>
<tr>
<td><strong>Low Level Construction</strong></td>
<td>object creation, object destruction</td>
<td>method invocation</td>
</tr>
<tr>
<td><strong>Basic Programming</strong></td>
<td>memory allocation, memory free</td>
<td>function call, loop overhead, basic operations (add, mul, load, store)</td>
</tr>
<tr>
<td><strong>OS</strong></td>
<td>task, thread creation</td>
<td>task switch, interrupt response</td>
</tr>
<tr>
<td><strong>HW</strong></td>
<td>power up, power down, boot</td>
<td>cache flush, low level data transfer</td>
</tr>
</tbody>
</table>
Measurements have stochastic variations and systematic deviations resulting in a range rather than a single value.
Be Aware of Error Propagation

\[ t_{\text{duration}} = t_{\text{end}} - t_{\text{start}} \]

- \[ t_{\text{start}} = 10 +/- 2 \mu s \]
- \[ t_{\text{end}} = 14 +/- 2 \mu s \]
- \[ t_{\text{duration}} = 4 +/- ? \mu s \]

**systematic errors:** add linear

**stochastic errors:** add quadratic
### Intermezzo Modeling Accuracy

**Measurements** have **stochastic variations** and **systematic deviations** resulting in a **range** rather than a **single value**.

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**.
**Tools and Instruments in the Benchmark Stack**

*typical small test program*

create steady state

\[ t_s = \text{timestamp()} \]

for \((i = 0; i < 1M; i++)\) do something

\[ t_e = \text{timestamp()} \]

duration = \( t_s - t_e \)
Abstract

The management of the resources largely determines system performance. This document discusses concepts related to resource management, such as caching, concurrency, and scheduling.
Generic Resource Model

Architecting System Performance; Resource Management

version: 0.1
June 21, 2020
ASPRMgenericResources
Performance depends on resource utilization and management.

The design of the logistics, how does EMI\(^1\) flow through the resources, is critical.

Critical design aspects are:

- concurrency (parallelism, pipelining)
- granularity of EMI
- scheduling (allocation of resources)

\(^1\)Energy Material Information
Granularity as Key Design Choice

<table>
<thead>
<tr>
<th>unit of buffering</th>
<th>== or &lt;&gt;</th>
<th>unit of synchronization</th>
<th>== or &lt;&gt;</th>
<th>unit of processing</th>
<th>== or &lt;&gt;</th>
<th>unit of I/O</th>
</tr>
</thead>
</table>

- **Video frame**
- **Video line**
- **Pixel**

*Fine grain*: flexible, high overhead
*Coarse grain*: rigid, low overhead
Size versus Performance Trade-off

---

**small capacity**
- fast technology
- small
- expensive

**large capacity**
- slow technology
- large
- low cost

**staircase effect:**
- performance and size are non-linear with thresholds

---

Architecting System Performance; Resource Management

version: 0.1
June 21, 2020

ASPRMsizeVsPerformance
Pipeline pattern

production line = pipeline

Car n+3 → Car n+2 → Car n+1 → Car n

Lean uses the notion of tact: f.i. every 10 minutes the products move to the next workspot.

Throughput = products/time

\[ t_{\text{production 1 car}} = t_{\text{in}} - t_{\text{out}} \]
Y-chart Pattern

- structure and topology of resources
- design of dynamic behavior
- system performance

Feedback loop:
- mapping
- feedback
<table>
<thead>
<tr>
<th>Performance Pitfalls and Resource Management</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Overhead (control, handling)</strong></td>
</tr>
<tr>
<td><strong>Starvation (underrun)</strong></td>
</tr>
<tr>
<td><strong>Saturation/stagnation (overrun)</strong></td>
</tr>
<tr>
<td><strong>Variation (duration, quality)</strong></td>
</tr>
<tr>
<td><strong>Serialization</strong></td>
</tr>
<tr>
<td><strong>Interference with other work</strong></td>
</tr>
<tr>
<td><strong>Unnecessary conversions or adaptations</strong></td>
</tr>
</tbody>
</table>
Abstract

Greedy and lazy are two opposite patterns in performance design. An extreme application of both patterns is start-up, where greedy starts as much as possible, and lazy as little as possible.
<table>
<thead>
<tr>
<th>lazy (on demand, pull)</th>
<th>greedy (push, forecast)</th>
</tr>
</thead>
<tbody>
<tr>
<td>do nothing until someone needs it</td>
<td>prepare time consuming operations, when resources are idle</td>
</tr>
<tr>
<td>no resource usage unless needed</td>
<td>results are available immediately</td>
</tr>
<tr>
<td>time to result depends on execution time</td>
<td>some resource use is wasted</td>
</tr>
<tr>
<td>default</td>
<td>to achieve required performance (explore other concepts too!)</td>
</tr>
</tbody>
</table>

**Benefits**

- no resource usage unless needed
- results are available immediately
- some resource use is wasted
- to achieve required performance (explore other concepts too!)

**Disadvantages**

- time to result depends on execution time
- some resource use is wasted

**Default**

- default

**Applicable to**

- all domains (IT, goods flow, energy)

---

**Greedy and Lazy Patterns**

*Architecting System Performance; Greedy and Lazy Patterns*  
129 Gerrit Muller

*version: 0.1  
June 21, 2020  
ASPGpatterns*
Start up of Systems as Example

How much time does it take to start a laptop with Windows?

How much time does it take to start an application (e.g. Word)?
Example from Cloud Applications
### Caching Pattern (Physical Grab Stock)

<table>
<thead>
<tr>
<th>performance issues</th>
<th>solution patterns</th>
<th>design parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>long latency (mass) storage</td>
<td>low latency</td>
<td>caching algorithm</td>
</tr>
<tr>
<td>long latency communication</td>
<td>less communication</td>
<td>storage location</td>
</tr>
<tr>
<td>overhead communication</td>
<td>large chunks (less overhead)</td>
<td>cache size</td>
</tr>
<tr>
<td>resource intensive processing</td>
<td>processing once (keep results)</td>
<td>chunk size</td>
</tr>
<tr>
<td></td>
<td>frequently used subset in fast local storage</td>
<td>format</td>
</tr>
</tbody>
</table>
Many Layers of Caching

<table>
<thead>
<tr>
<th>Cache Type</th>
<th>Cache Miss Penalty</th>
<th>Cache Hit Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Cache</td>
<td>1 s</td>
<td>10 ms</td>
</tr>
<tr>
<td>Network Layer Cache</td>
<td>100 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td>File Cache</td>
<td>10 ms</td>
<td>10 µs</td>
</tr>
<tr>
<td>Virtual Memory</td>
<td>1 ms</td>
<td>100 ns</td>
</tr>
<tr>
<td>Memory Caches L1, L2, L3</td>
<td>100 ns</td>
<td>1 ns</td>
</tr>
</tbody>
</table>

*Typical cache 2 orders of magnitude faster*
Disadvantages of Caching Pattern

- robustness for application changes
- ability to benefit from technology improvements
- robustness for changing context (e.g. scalability)
- robustness for concurrent applications
- failure modes in exceptional user space

These patterns increase **complexity** and **coupling**.

Use only when necessary for performance.
Abstract

Scheduling plays a crucial role in resource allocation to get desired system performance. This document discusses local and global scheduling.
Scheduling of time critical operations on a single resource:

- Earliest Deadline First
  - optimal
  - complex to realize

- Rate Monotonic Scheduling
  - no full utilization
  - simple to realize
## Earliest Deadline First

<table>
<thead>
<tr>
<th><strong>Constraints</strong></th>
<th><strong>Determine deadlines</strong></th>
<th><strong>in Absolute time (CPU cycles or msec, etc.)</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Assign priorities</strong></td>
<td>Process that has the earliest deadline gets the highest priority (no need to look at other processes)</td>
<td></td>
</tr>
<tr>
<td><strong>Constraints</strong></td>
<td>Smart mechanism needed for Real-Time determination of deadlines Pre-emptive scheduling needed</td>
<td></td>
</tr>
</tbody>
</table>

EDF = Earliest Deadline First

Earliest Deadline based scheduling for (a-)periodic Processing

The theoretical limit for any number of processes is 100% and so the system is schedulable.
Calculate loads and determine thread activity (EDF)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Suppose at t=0, all threads are ready to process the arrived trigger.

Source: Ton Kostelijk - EXARCH course
## Rate Monotonic Scheduling

<table>
<thead>
<tr>
<th>• Determine deadlines (period)</th>
<th>in terms of Frequency or Period ( (1/F) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Assign priorities</td>
<td>Highest frequency (shortest period) ( \Rightarrow ) Highest priority</td>
</tr>
<tr>
<td>• Constraints</td>
<td>Independent activities</td>
</tr>
<tr>
<td></td>
<td>Periodic</td>
</tr>
<tr>
<td></td>
<td>Constant CPU cycle consumption</td>
</tr>
<tr>
<td></td>
<td>Assumes Pre-emptive scheduling</td>
</tr>
</tbody>
</table>

RMS = Rate Monotonic Scheduling

Priority based scheduling for Periodic Processing of tasks with a guaranteed CPU - load
Exercise Rate Monotonic Scheduling (RMS)

Calculate loads and determine thread activity (RMS)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Suppose at t=0, all threads are ready to process the arrived trigger.

<table>
<thead>
<tr>
<th>0</th>
<th>9</th>
<th>15</th>
<th>18</th>
<th>23</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Thread 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Source: Ton Kostelijk - EXARCH course
### RMS-RMA Theory

<table>
<thead>
<tr>
<th>Assumptions Rate Monotonic Analysis (RMA):</th>
</tr>
</thead>
<tbody>
<tr>
<td>periodic tasks with period $T_i$</td>
</tr>
<tr>
<td>process time $P_i$</td>
</tr>
<tr>
<td>load $U_i = \frac{P_i}{T_i}$</td>
</tr>
<tr>
<td>tasks are independent</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>RMA theory:</th>
</tr>
</thead>
<tbody>
<tr>
<td>schedule is possible when:</td>
</tr>
<tr>
<td>Load $= \sum_i U_i \leq n(2^{1/n} - 1)$</td>
</tr>
<tr>
<td>for $n = 1, 2, 3, \infty$</td>
</tr>
<tr>
<td>max utilization is:</td>
</tr>
<tr>
<td>$1.00, 0.83, 0.78, \ldots \log(2)$</td>
</tr>
<tr>
<td>$\approx 0.69$</td>
</tr>
</tbody>
</table>

Rate Monotonic Scheduling (RMS) uses fixed priorities

RMS guarantees that all processes meet their deadlines

Fixed priority $\rightarrow$ low overhead

Source: Ton Kostelijk - EXARCH course
## Answer EDF Exercise

### Answers: loads and thread activity (EDF)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td>21.7%</td>
</tr>
</tbody>
</table>

\[
\text{Total Load} = 88.3\%
\]

Source: Ton Kostelijk - EXARCH course
## Answers: loads and thread activity (RMS)

<table>
<thead>
<tr>
<th>Thread</th>
<th>Period = deadline</th>
<th>Processing</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thread 1</td>
<td>9</td>
<td>3</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 2</td>
<td>15</td>
<td>5</td>
<td>33.3%</td>
</tr>
<tr>
<td>Thread 3</td>
<td>23</td>
<td>5</td>
<td>21.7%</td>
</tr>
</tbody>
</table>

**88.3%**

**Source:** Ton Kostelijk - EXARCH course

---

**Diagram:**

Thread 1: 3, 9, 3, 15, 3, 23
Thread 2: 5, 9, 3, 20, 3, 18, 3, 22
Thread 3: 1, 13, 3, 22, 3, 19, -1, ??
A perspective on dynamic behavior is to view the system as set of periodic behaviors.

Periodic behavior is easier to model and analyze, e.g. using RMS and RMA.

Modern systems and Systems of Systems consists of complex networks of concurrent resources.

Typically, a combination of more advanced global scheduling is combined with simple local scheduling.
Abstract

Performance should be robust. The performance should be reproducible and it should be well-behaving in extreme conditions.
Variations are Suspect

Poorly understood variations require analysis

Performance measurements in standardized conditions

What causes these outlyers?

Performance smaller is better

What causes this variation?

# of cases
Coping with Disturbances

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?

steady state performance
degradation
disturbance
recovery time
degradation
disturbance
recovery time

How does the system respond to disturbances?
How quickly does it recover?
How far does performance degrade?
A system design assumption is often: the performance of this function
{ is constant | is linear | doesn't exceed x | ...}.

The working range is the interval where this assumption holds.
## Validate Understanding of System Performance

| **Characterize the system** | use the system in varying conditions  
measure performance  
as function of the conditions |
|----------------------------|----------------------------------|
| **Stress testing**         | where does the design fail?  
(go beyond specified limits) |
| **Load testing**           | keep the system in heavy load condition  
observe how it keeps performing  
measure variations |
| **(Accelerated) Lifetime testing** | age the system  
observe how it keeps performing |
Abstract

A threat to performance is the combination of feature creep and technical debt. This combination causes bloating of the design. In Lean terms, the combination causes waste. A crucial question is where is the value, and is the value in balance with the potential degradation of performance.
From Feature Creep to Performance Problems

- Maturing of systems
- Increasing number of features (feature creep)
- Bloating of design
- Increase in resource usage
- Design complexity
- Loss of knowledge
- Performance problems
- Lack of overview insight understanding
- Time effort gain by taking shortcuts
- Technical debt

Loss of knowledge
Technical Debt is a **metaphor** used within the software industry to communicate the consequences of **pragmatic design decisions** deviating from the **intended design** of a system.

from: [http://gaudisite.nl/INCOSE2016_Callister_Andersson_SMARTtechnicalDebt.pdf](http://gaudisite.nl/INCOSE2016_Callister_Andersson_SMARTtechnicalDebt.pdf)

Value versus Performance Degradation

Are benefits (value) in balance with the costs (such as performance degradation)?

- increasing number of features (feature creep)
- time effort gain by taking shortcuts
- technical debt
- increase in resource usage
- performance problems
- bloating of design
- design complexity
- lack of overview insight understanding

<table>
<thead>
<tr>
<th>Benefits (Value)</th>
<th>Costs (such as performance degradation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Increasing number of features</td>
<td>Technical debt</td>
</tr>
<tr>
<td>Time effort gain by taking shortcuts</td>
<td>Increase in resource usage</td>
</tr>
<tr>
<td>Design complexity</td>
<td>Performance problems</td>
</tr>
<tr>
<td>Lack of overview insight understanding</td>
<td></td>
</tr>
</tbody>
</table>
Exploring bloating: main causes

- Poor specification ("what")
- Poor design ("how")
- Core function
- Support for unused legacy code
- Genericity
- Configurability
- Provisions for future
- Support for unused legacy code
- Dogmatic rules for instance fine grain COM interfaces

Legend:
- Overhead
- Value
Necessary functionality ≫ the intended regular function

- testing
- regular functionality
- instrumentation
diagnostics tracing asserts
- boundary behavior:
  - exceptional cases
  - error handling

Bloating, Waste, and Value
155 Gerrit Muller

version: 0
June 21, 2020
BLOATcoreFunctionality
The danger of being generic: bloating

"Real-life" example: redesigned Tool super-class and descendants, ca 1994
Shit propagation via copy paste

- needed code
- copy paste
- modify
- bad code
- new needed code
- code not relevant for new function
- repair code
- bad code
- new bad code

Bloating, Waste, and Value

157 Gerrit Muller
Example of shit propagation

Class Old:
    capacity = startCapacity
    values = int(capacity)
    size = 0
    def insert(val):
        values[size]=val
        size+=1
        if size>capacity:
            capacity*=2
            relocate(values, capacity)

Class New:
    capacity = 1
    values = int(capacity)
    size = 0
    def insert(val):
        values[size]=val
        size+=1
        capacity+=1
        relocate(values, capacity)

Class DoubleNew:
    capacity = 1
    values = int(capacity)
    size = 0
    def insert(val):
        values[size]=val
        size+=1
        capacity+=1
        relocate(values, capacity)

    def insertBlock(v,len):
        for i=1 to len:
            insert(v[i])
Bloating causes more bloating

**Bloating, Waste, and Value**

159   Gerrit Muller

version: 0  
June 21, 2020  
EASRTbloatingCausesBloating
Bloating causes performance and resource problems. Solution: special measures: memory pools, shortcuts, ...

Bloating causes performance and resource problems...

Solution: special measures: memory pools, shortcuts, ...

Bloating, Waste, and Value
160 Gerrit Muller

version: 0
June 21, 2020

EASRT bloating Causes Bloating More