Software Technology for Internet of Things

Performance Evaluation

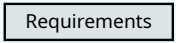
Aslak Johansen asjo@mmmi.sdu.dk

April 22, 2025

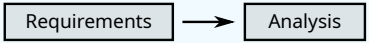
SDU

Part 0:
Introduction

Performance Evaluation

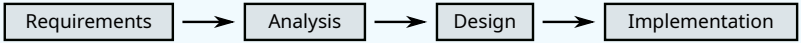


Performance Evaluation



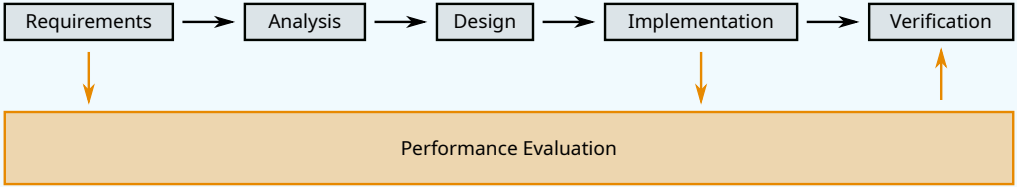
Performance Evaluation

Performance Evaluation

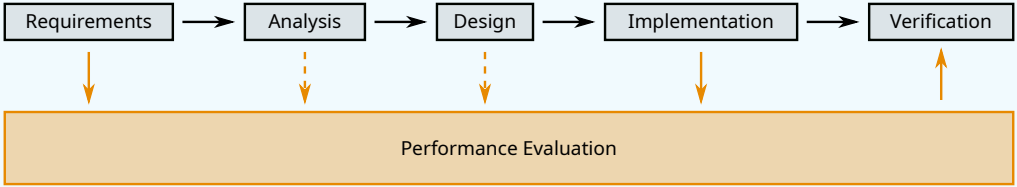


Performance Evaluation

Performance Evaluation



Performance Evaluation



Systems

When discussing the performance of a system, we need to have a shared understanding of *what* a system is.

For the purpose of this class *a system is a collection of independent components that has been combined in a particular configuration in order to solve some problem*.

Certain components – such as IoT devices – may not be available to us when we need to ascertain the performance of the full system.

Accordingly, a subset of these components may have to be replaced with stubs when the system is under evaluation.

Often, we are evaluating a subsystem or even parts of a single component.

Part 1:
Experimental Computer Science

Introduction

"Science classifies knowledge. Experimental science classifies knowledge derived from observations. The experimenters construct apparatus to measure a given phenomenon; they then attempt to collect data sufficient to support their hypotheses about that phenomenon."

— Peter J. Dennings, `https://doi.org/10.1145/359015.359016`

This is how we construct arguments supporting conclutions on peformance-related non-functionality.

It is not specific to work within the course domain, but it is highly relevant.

Hypothesis

When designing a system we make decisions that we expect to result in possitive qualities of the resulting system.

Our expectations can be formulated as a set of hypotheses.

Expectations are often based on non-functional requirements and rooted in intuitions.

"The throughput of the system will gracefully decline as the number of concurrent connections outnumber the number of physical CPU cores."

"The introduction of the Internet will increase the productivity of programmers."

*"The proposed networking topology between Odense and Copenhagen is able to limit the number of dropped UDP messages to 7% during the course of a typical day."**

Hypothesis Testing

Hypothesis testing is the act of testing to which degree a hypothesis holds true. Results are usually either boolean, floats or probabilistic.

How to accomplish that depends on the formulation of the hypothesis.

Hypothesis testing has its roots in statistics. In this course, we won't require formal statistical arguments to support conclusions on hypotheses.

However, we will be constructing aparatus that allows us to collect large amounts of data measuring the phenomenons referenced in the hypotheses.

Apparatus

The apparatus is the system under evaluation.

It is constructed to support conclusions on the hypothesis probability.

When doing experimental evaluation the apparatus (i.e., the system) is constructed to fit the requirements rather than to fit the experiment.

Note: That difference may represent a roadblock in practice, and one would thus have to *instrument* the system to allow for evaluation.

Tests

Tests are designed to highlight relevant performance metrics of the phenomenon of interest.

These tests are administered using an experimental setup surrounding the apparatus.

They can be conducted manually, or in a partial or fully automatized fashion.

However, since repetition is key to most (if not all) experiments, some degree of automation is to be preferred.

Repeatability

For a conclusion to be trustworthy it needs to be reproducible.

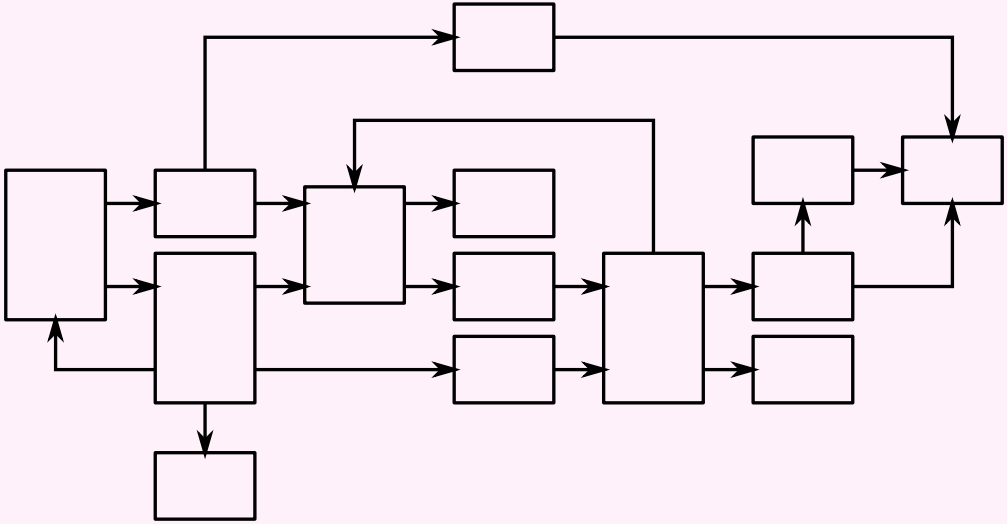
That requires:

- Thorough documentation.
- Enough repetitions to provide enough resolution in the produced distribution of the results to highlight the true key characteristics.

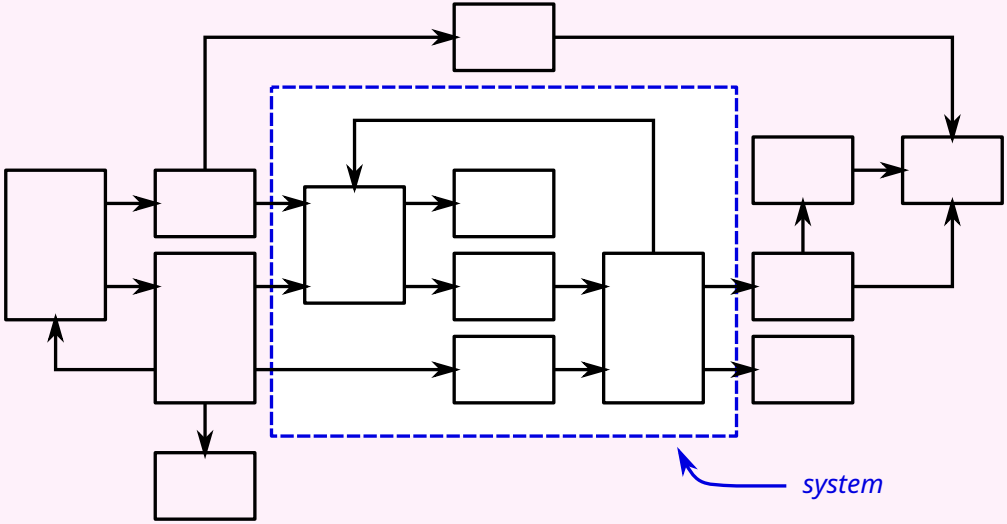
If a repetition experiment fails (to repeat), there should be a difference in the experimental setup or a flawed hypothesis.

Part 2:
Performance Evaluation

System Definition



System Definition



Goals

The first step is to define a set of goals.

All choices from hereon out are made in order to fulfill those goals.

At times, part of the evaluation can be reused for multiple goals, but often multiple different setups are needed.

Clearly defined goals are easier to find a solution for.

Avoid bias: *"Show that system A is better than system B"*

Services

A system will provide a number of services.

These could be:

- *Route network packets to a gateway.*
- *Insert a time $\times$ value sample into a database.*
- *React to the production of a value from a sensor.*

Note: This is a different definition of a service that what you are used to from the software engineering educations.

Some service outcomes are desirable (e.g., correct response) while others are not (e.g., incorrect response or a late response).

When performing an experiment we usually focus on a single service or an observed mix of services usage.

Metrics

Metrics are criteria according to which we may quantify and compare performance.

Examples include:

- *The delay is measured as the time it takes from the sensor reporting a value until that value is presented on the users screen.*
- *The memory consumption is measured as the resident memory allocated to the process of the gateway component.*
- *The throughput is measured as the highest average frequency across 10s at which the database is capable of storing sensor readings.*

Note: Metrics are related to units, and yet a completely different thing (e.g., two metrics may be measured using the same unit).

In this course we are using SI units and prefixes, although the binary variants are also valid (e.g., 1kiB is 1024 bytes).

Parameters

Usually, systems under experimentation are highly parameterized, and the observed results express how a specific configuration performs.

Examples include:

- *Buffer sizes.*
- *Thread counts.*
- *Timeouts.*
- *Number of allowed retries.*

Documenting this configuration is key to ensure that the results are going to be understandable, reproducible and trustworthy.

Factors

While most parameters remain static throughout an evaluation, often one needs to tune a subset of the parameters, or select which of them to make factors of for studying.

Tuning is done by assigning a series of discrete values to a parameter – thereby making it a factor – and experimentally determining the best value.

Some of these factors may have complex interactions with each other, and examining all combinations (what is known as a full-factorial experiment) can be computationally expensive.

To deal with such realities several approaches exist, including hillclimbing one or more factors at a time.

Workload

A workload is a predefined sequence of inputs that are fed into the system (at pre-determined times) under observation.

Workloads are usually either synthetic or replays of historical recordings.

A common form of synthetic workload is to repeat some class of request at a particular frequency.

Often a separate process is designed to produce the workload.

Experimental Design

An experimental design is a construct that allows for experimentation. The goal is to come up with a sequence of experiments that offer maximum value at minimum effort.

It should

1. include the stubs necessary to execute the system as defined.
2. operate instances of the relevant components according to fixed values for all parameters.
3. include stub implementations for all dependencies to the system.
4. produce the needed workload.
5. iterate over relevant factors and workloads.
6. instrument relevant components to observe and log relevant metrics.
7. not influence the system under experimentation*.

Documenting the experimental design is key to any performance evaluation.

Executing an experimental design may be done automatically through the use of a testbed or manually (typically using some form of test harness).

Time

High-resolution timing is expensive, and is often not critical.

Two timestamps on a single machine are directly comparable . . . if it doesn't run NTP.

Monotonic clocks *may* help, especially for short periods of time.

On a general-purpose operating system, the scheduler makes it virtually impossible to perform high precision timing.

Two timestamps from different machines are not directly comparable:

- The clocks are unlikely to be in sync.
- The clocks are likely to drift at different rates.

In a distributed setting steps need to be taken to ensure that timestamps are comparable, or the resulting uncertainty need to be quantified (and bounded).

Data Analysis

After execution of an experiment you are left with a number of logfiles (e.g., n per machine in the system under experimentation).

Depending on how the execution was administered you may have to process them before analysis.

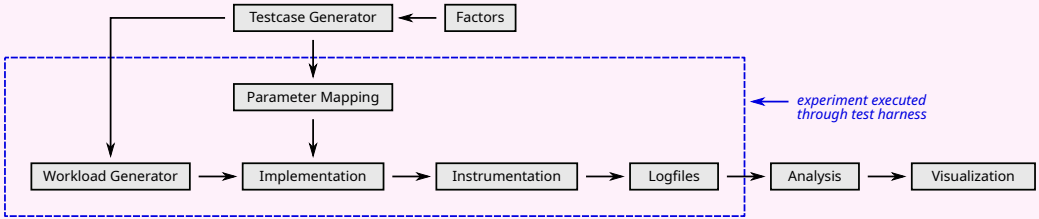
Usually the analysis is centered around finding events in the logs and extracting timing and resource consumption metrics from the relevant periods.

Data Presentation

Once the relevant metrics have been extracted it needs to be presented in ways that highlight how they relate to the goals.

The next section gives examples of how this could be done.

Summary


Testcase Generator
Factors
Parameter Mapping
experiment executed
through test harness
Workload Generator
Implementation
Instrumentation
Logfiles
Analysis
Visualization

Part 3:
Visualizing Results

Experimental Results

The best way to do an initial sanity check of experimental results is to visualize them.

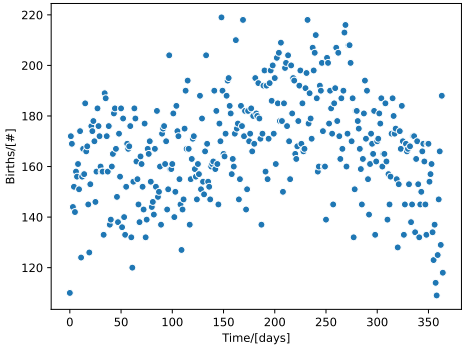
The following slides show various ways of illustrating a dataset.

The dataset in question: Live child births in Denmark throughout 2019 sourced from Danish Statistics.

In addition to the following types of visualization, I often use heatmaps or contour maps.

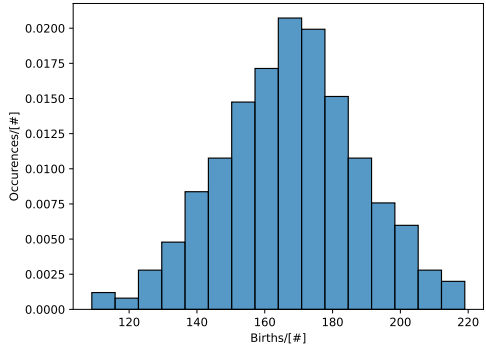
Timeseries Data

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('dkbirths2019.csv', names=['day', 'birth', 'weekday'])
plot = sns.scatterplot(x=df['day'], y=df['birth'])
plt.xlabel('Time/[days]')
plt.ylabel('Births/[#]')
plt.savefig('scatter.pdf')
```



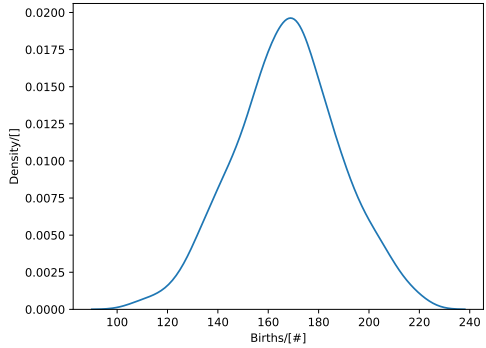
Histograms of Distributions

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('dkbirths2019.csv', names=['day', 'birth', 'weekday'])
plot = sns.histplot(df, x=df['birth'], stat="density", kde=False)
plt.xlabel('Births/[#]')
plt.ylabel('Occurences/[#]')
plt.savefig('hist.pdf')
```



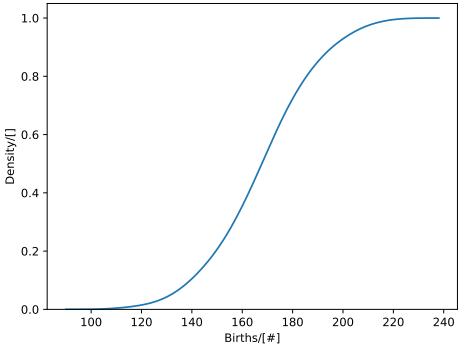
KDE's of Distributions

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('dkbirths2019.csv', names=['day', 'birth', 'weekday'])
plot = sns.kdeplot(x=df['birth'])
plt.xlabel('Births/[#]')
plt.ylabel('Density/[]')
plt.savefig('kde.pdf')
```



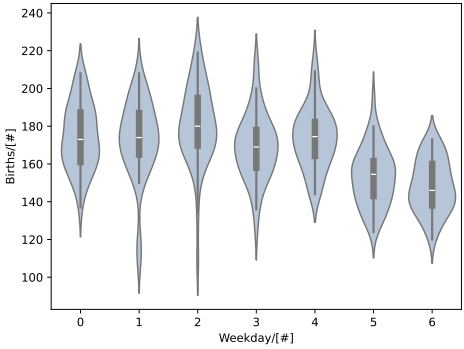
Cumulative Distribution Functions

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('dkbirths2019.csv', names=['day', 'birth', 'weekday'])
plot = sns.kdeplot(x=df['birth'], cumulative=True)
plt.xlabel('Births/[#]')
plt.ylabel('Density/[]')
plt.savefig('kde-cumulative.pdf')
```



Factor-Dependent Distributions

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

df = pd.read_csv('dkbirths2019.csv', names=['day', 'birth', 'weekday'])
plot = sns.violinplot(data=df, x='weekday', y='birth', color="lightsteelblue")
plt.xlabel('Weekday/[#]')
plt.ylabel('Births/[#]')
plt.savefig('violin.pdf')
```



Presenting Plots

Any presenteded plot of data should be self-contained:

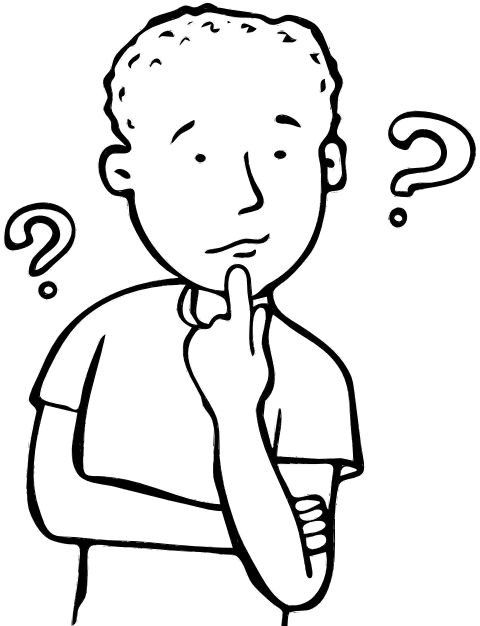
1. Context (or context reference) should be obvious.
2. Any axis in a plot should be labeled with metric and unit.
3. In case of multiplots a legend should be used to differentiate between different datasets.

Make axes start in zero, except when

- the axis represents absolute time.
- the axis is logarithmic.
- you have a good reason (which you choose to express).

Make systematic use of colors (static $\{\mathit{implementation} \mapsto \mathit{color}\}$ mapping)

Questions?



https://openclipart.org/detail/238087/boy-thinking-of-question