Measurement theory basics

Ivano Malavolta

VU VRIJE UNIVERSITEIT AMSTERDAM
LOOKING FURTHER

Roadmap

Concepts

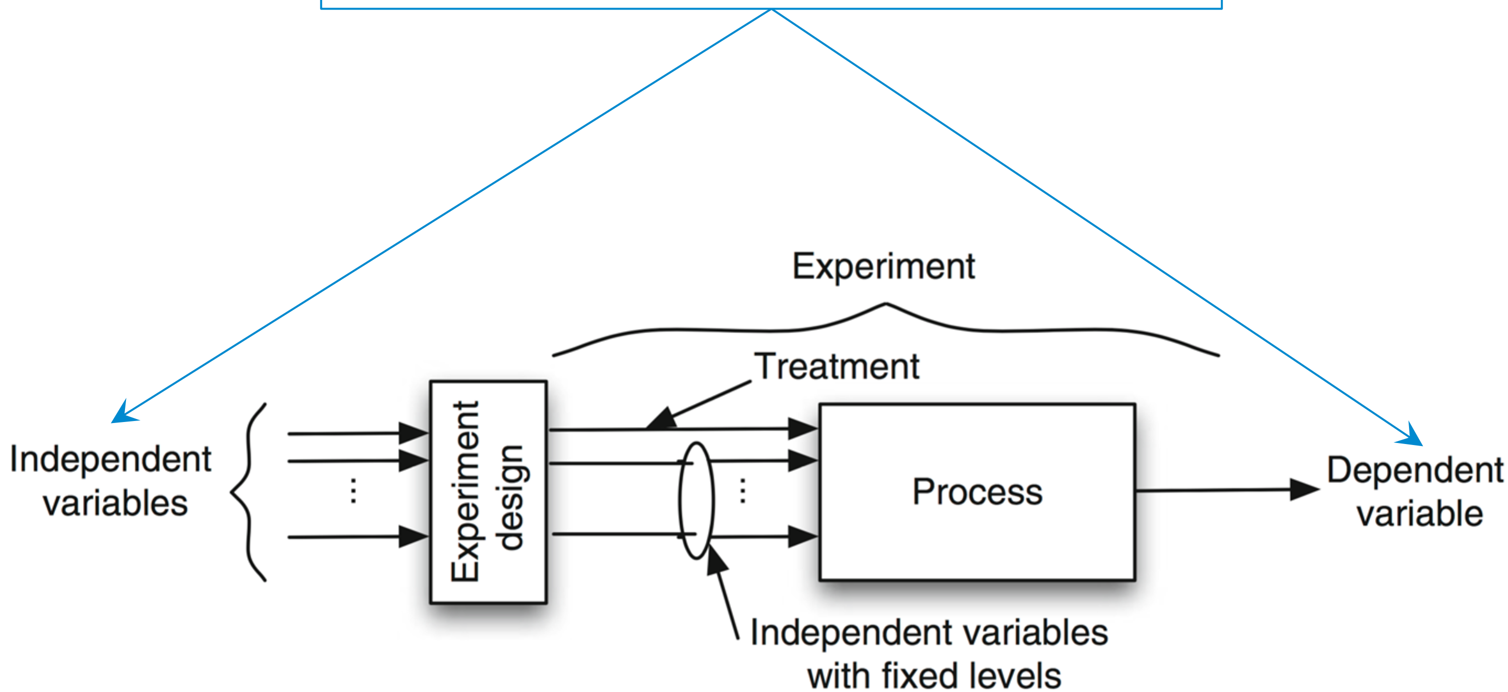
Scale types

Homework

Measurement

Measurement is the central part of empirical software engineering

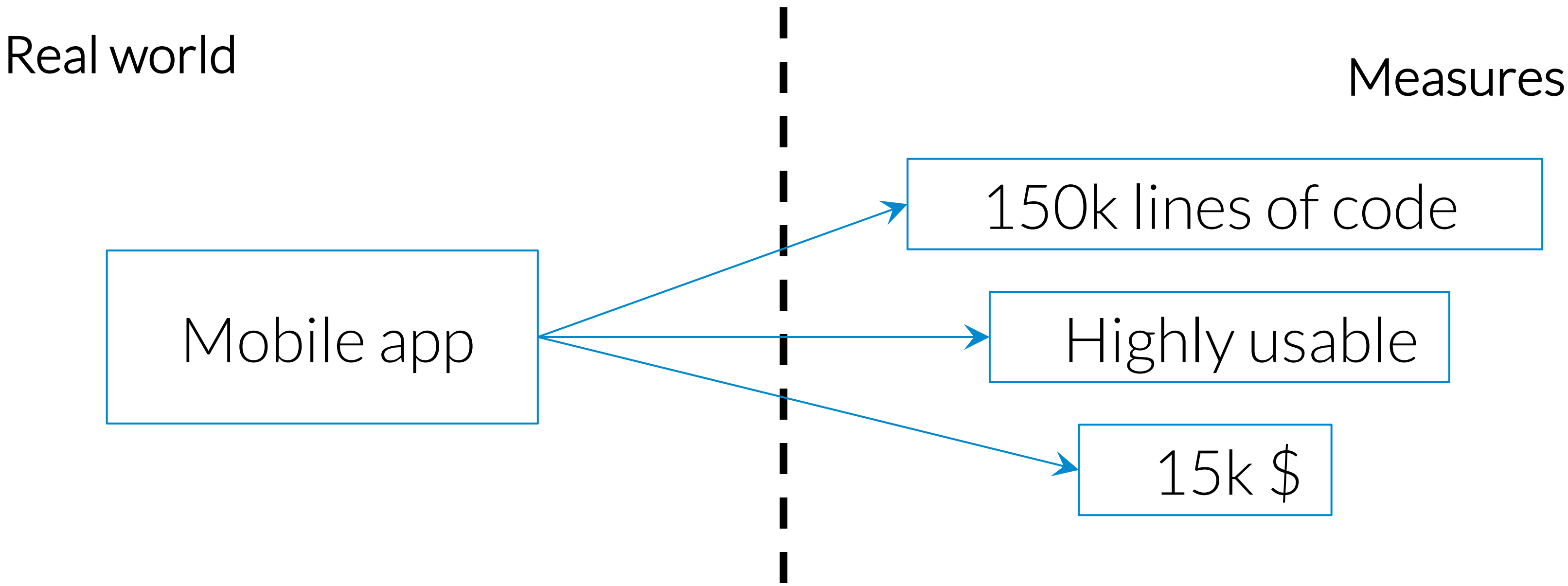
We need to measure these



Measurement

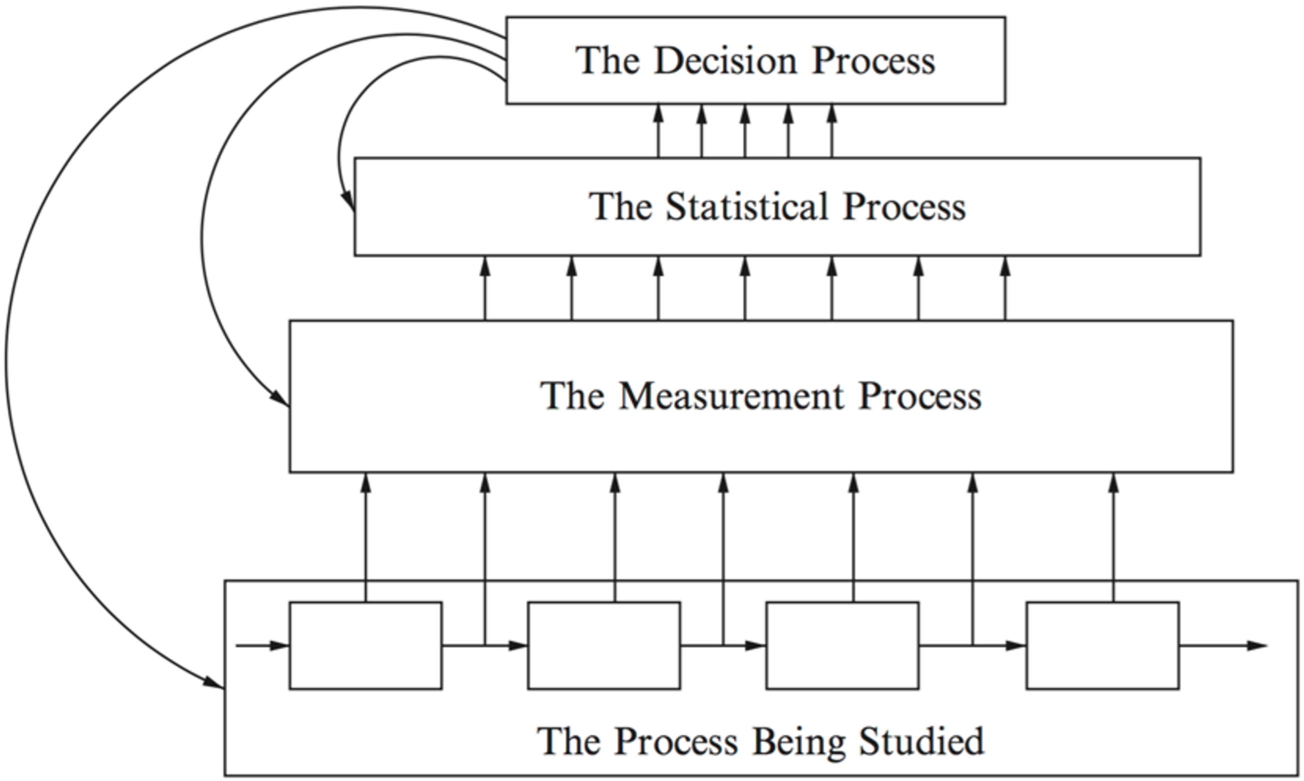
Measurement: the process of assigning numbers or symbols to attributes of entities in the real world

Measure:
the actual number or symbol assigned to an attribute of an entity



Conceptual framework

The overall goal of measuring is to trait entity attributes formally (e.g., statistically) for making claims or decisions



Conceptual framework

Remember that we are measuring for answering questions

Usually a single metric is not sufficient to adequately answer even an apparently simple question

→ we will have a set of measures for each question, each measure must be:

- precise
- reliable
- valid

Preciseness

Preciseness: the size of a measure's smallest unit

For example:

- does the height of a person need to be measured to the millimeter?
- does the energy consumption of a mobile app need to be measured to the mJ?

Tip: the precision of any derived measures (e.g., the average) must have the same precision of the original measured

e.g. the average height of Dutch people is 183.67893 cm
e.g. the average launch time of app X is 57 nanoseconds

Compound metrics: remember that the arithmetic combination of measures propagates and magnifies the error inherent in the original values

Reliability

Reliability: measurements must be consistent across repeated observations in the same circumstances

Relatively easy for physical measures, difficult for unstable or human-based ones

e.g. energy consumption
launch time of a mobile app
rating scales

Typically quantified by:

- the standard deviation/coefficient of variation or
- other coefficient measures like the Cronbach's coefficient alpha, or the Cohen-Kappa coefficient
  - they can be viewed as a correlation among repeated measurements

Classical technique for mitigating unreliable measures: **repetitions**

Validity

A measure is valid if it:

- does not violate any properties of the attribute it measures
- is a proper mathematical characterization of the attribute

- **Content validity**: how the measure reflects the <u>domain</u> it is intended to measure
  - e.g. measure program complexity according to the language used for the names of the variables? No
- **Criterion validity**: how the measure reflects the <u>measured</u> <u>object</u> w.r.t. to some criterion
  - e.g., a complexity measure should assign high values to programs which are known to be highly complex
- **Construct validity**: how a measure actually represents the <u>conceptual</u> <u>entity</u> of interest. e.g., #lines of code for measuring program size? Yes

Lines of code for program complexity?

```
public static void main(String args[]) {
    /*
        Creating instance of the controller.
        Some more comments...
    */
    final SalesDomainController domainController =
            new SalesDomainControllerImpl();
    if (args.length == 1
            && args[0].equals("console")) {
        ConsoleUI cui =
                new ConsoleUI(domainController);// a small console UI
        cui.run();
    } else {
        // Swing UI
        final SalesSystemUI ui =
                new SalesSystemUI(domainController);
        ui.setVisible(true);
    }
    log.info("SalesSystem started");
}
```

Reliable
Easy to measure
How to interpret:
- empty lines
- comments
- several statements on one line

Language dependent
Is it related to complexity?

Scale types

Scale types

- Most used scale types:
  - Nominal
  - Ordinal
  - Interval
  - Ratio

Choosing a scale for a variable means constraining the statistical analysis that you can do on it

Nominal

You can see it as "tagging"

It maps the attribute of an entity to a name or symbol
e.g.
caching strategy: {no-cache, cache-only, cache-first, cache-network-race}

It is the least powerful from a statistical point of view

Name	Examples outside SE	Examples inside SE	Constraints
Nominal	Colours: 1. White 2. Yellow 3. Green 4. Red 5. Blue 6. Black	Testing methods: • type I (design inspections) • type II (unit testing) • type III (integration testing) • type IV (system testing) Fault types: • type 1 (interface) • type 2 (I/O) • type 3 (computation) • type 4 (control flow)	Categories cannot be used in formulas even if you map your categories to integers. We can use the mode and percentiles to describe nominal data sets.

Ordinal

It ranks the entities after an **ordering criterion**
→ you can see it as "tagging with a given order"

Examples of criteria: "greater than", "more complex", "more recent"
e.g. type of available network connection: {wifi, 3G, 2G}

Name	Examples outside SE	Examples inside SE	Constraints
Ordinal	The Mohs scale to detect the hardness of minerals or scales for measuring intelligence.	Ordinal scales are often used for adjustment factors in cost models based on a fixed set of scale points, such as very high, high, average, low very low. The SEI Capability Maturity Model (CMM) classifies development on a five-point ordinal scale.	Scale points cannot be used in formulas. So, for instance, 2.5 on the SEI CMM scale is not meaningful. We can use the median and percentiles to describe ordinal data sets.

Interval

Used when the difference between two measures are meaningful, but not the value itself

Example – levels of satisfaction on a Likert scale

Similar to the ordinal scale, but there is a notion of “relative distance” between two entities

Name	Examples outside SE	Examples inside SE	Constraints
Interval	Temperature scales: -1 degree centigrade 0 degrees centigrade 1 degree centigrade etc.	If we have been recording resource productivity at six-monthly intervals since 1980, we can measure time since the start of the measurement programme on an interval scale starting with 01/01/1980 as 0, followed by 01/06/1980 as 1, etc.	We can use the mean and standard deviation to describe interval scale data sets.

Ratio

Used when:

- the values are ordered
- the values have equal intervals
- there is a meaningful zero value

Eg - energy consumption in Joules

Name	Examples outside SE	Examples inside SE	Constraints
Ratios	**Length, mass, length**	**The number of lines of code in a program is a ratio scale measure of code length.**	**We can use the mean, standard deviation and geometric mean to describe interval data sets.**

Question: are "x" Joules a lot or not?

- In research the goal is almost always to compare different alternatives and compare them against each other, it is not a good practice to focus on the absolute numbers alone
- If you want to have/give an intuition about the values that you are going, transform the Energy values (in Joules) into Battery lifetime (in seconds):

Assuming that your experiment was performed on a Google Nexus 5X
(battery capacity= 6700 mAh, voltage=3.8V)
Total energy in the battery (in Joules) = charge x (3600 / 1000) *(mAh)* x voltage *(v)*
Total energy = (6700 x 3.6) x 3.8 = 91656 J
If you have that on average the runs with treatment A last 2 minutes and consume 100J
91656 : X = 100 : 2 → X = (91656 x 2)/100 = 1833.12 minutes
Total lifetime of the battery: 1833.12 minutes ~= 30 hours
If the runs with treatment B consume 130J…
91656 : X = 200 : 2 → X = (91656 x 2)/**130** = 1410 minutes ~= 23.5 hours
→ 1833.12 – 1410 = 423.12 minutes ~= 7 hours
→ **Treatment B can lead to a reduction of 7 hours of the battery life of a Nexus 5X**

Some hints: repetitions

The general rule of thumb is to have **30 repetitions** (I never saw more than them)
But this number is relatively high, you should make the math according to the design of your experiment and scale down this number accordingly in order to make the experiment feasible.

For example, if we have the RQ about the image formats (jpeg vs png), **100** subject apps (50 with jpeg and 50 with png images), and **30** repetitions, the execution time of the experiment is:

> 100 X 30 X (2 minutes of idle time between runs (this is standard) X 1 minute of loading time of the app) = 9000 minutes = 150 hours = **6.25 days** of sheer execution time. In short: it is too long!

The rule of thumb for having a "good enough" experiment is about 30-40 hours, no more than that

Some hints: generalizability vs feasibility

Again, you need to do the math here and find a good trade-off.

Rule of thumb: For experiments involving only the initial load of a web app, you can have ~50 subjects.
For experiment involving the execution of usage scenarios (whose single run generally take longer than simply loading the app and closing it) you can stay around 10-20 apps.

Of course, the more subjects the better, depending on the feasibility of the experiment.

Suggestion 1: split the experiment execution into batches, where the first one is the minimum (according to the numbers above), then you add other subjects if you will have time.

Suggestion 2: do a full run of a "mini-version" of your experiment, for example by having only 2 subjects and 2 repetitions each. In this way you will:

- be sure that you are able to complete the experiment
- know already the structure of the measures, allowing you to already start working on the R scripts for data analysis

- analysing the new data coming from the new subjects will just consist in rerunning the analysis scripts on the new data

What this lecture means to you?

Now you know:

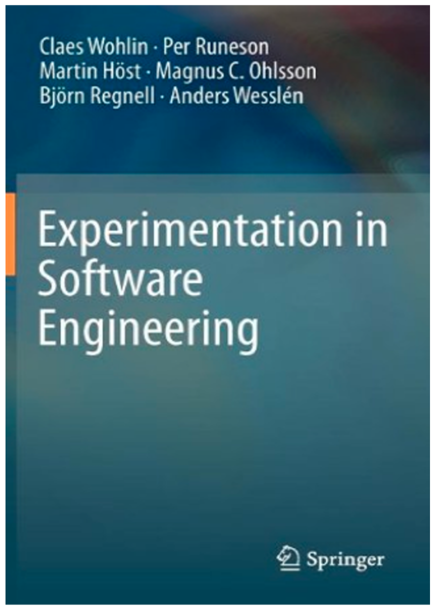
- The basics of software measurement theory
- How to define the measures

  they will map 1:1 to your experiments variables

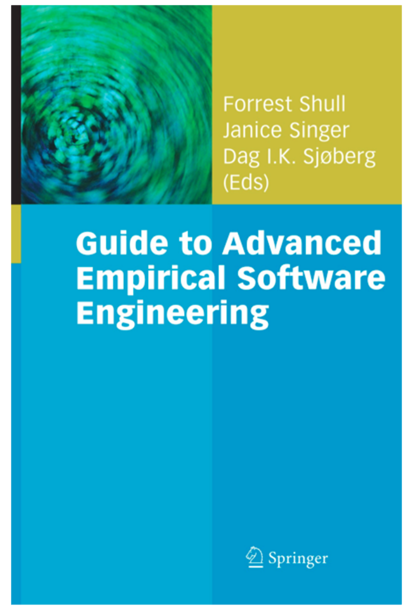
IMPORTANT!

The type of your measures will heavily impact the statistical analysis you will perform

Readings



Chapter 3



Chapter 6