Software Quality Management

Dr. Stefan Wagner
Technische Universität München

Garching
11 June 2010

Last QOT: Why do we need continuous quality control in software development?

"Failures are cheaper to fix if catched early."

"To correct the standard of quality assurance in any time."

"For maintaining the aggregation properties"

The most obvious reason for continuous quality control is that you detect defects earlier when they
are still cheap to fix. Failures have a specific meaning!

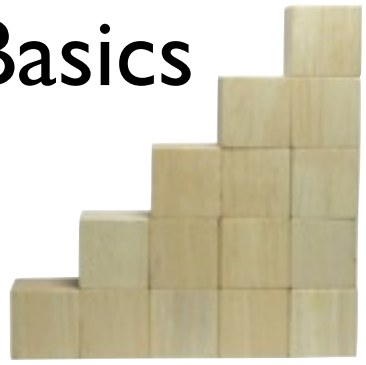
The control loop also helps to adjust your quality assurance approach.

Continuous quality control is not necessary for aggregating properties of the software. If this
comment aims more into the direction of integration, it could be reasonable if early detection
of problems by continuous integration is meant.

New QOT: "Why is software reliability a random process?"

Measurement theory

Review of last week's lecture:
Scales
Aggregation operators
GQM

Basics 	**Product**  Quality	Metrics and  Measurement
Process Quality	Quality Management	Certifi- cation 

We are in the part "Metrics and Measurement".

Reliability models

Quality measures

Visualisation

This lecture covers three parts:
Reliability (growth) models
An overview of quality measures (and a classification)
Visualisation of quality measures

Reliability models

Software reliability

Probability of a

failure-free operation of a software system for a

specified period of time in a specified environment.

The standard definition of software reliability adopted by various standardisation bodies such as IEEE.

It shows that reliability depends on the definition of failure, that reliability is a stochastical concept,

and that it can only be defined for a specified period and a specified environment.

In contrast to hardware, software does not wear off. Hardware can become disfunctional just

by mechanical influence. This does not hold for software. Theoretically, software could run

forever without any failure. The change in reliability in software comes from changing the

software, i.e., from fixing bugs.

Measures

- Probability of failure on demand (POFOD)
- Mean time to failure (MTTF)
- Mean time to repair (MTTR)
- Availability (MTTF/(MTTF+MTTR))
- Failure intensity
- Rate of fault occurrence (ROCOF)

We can find various measures that describe different aspects of reliability in the literature.

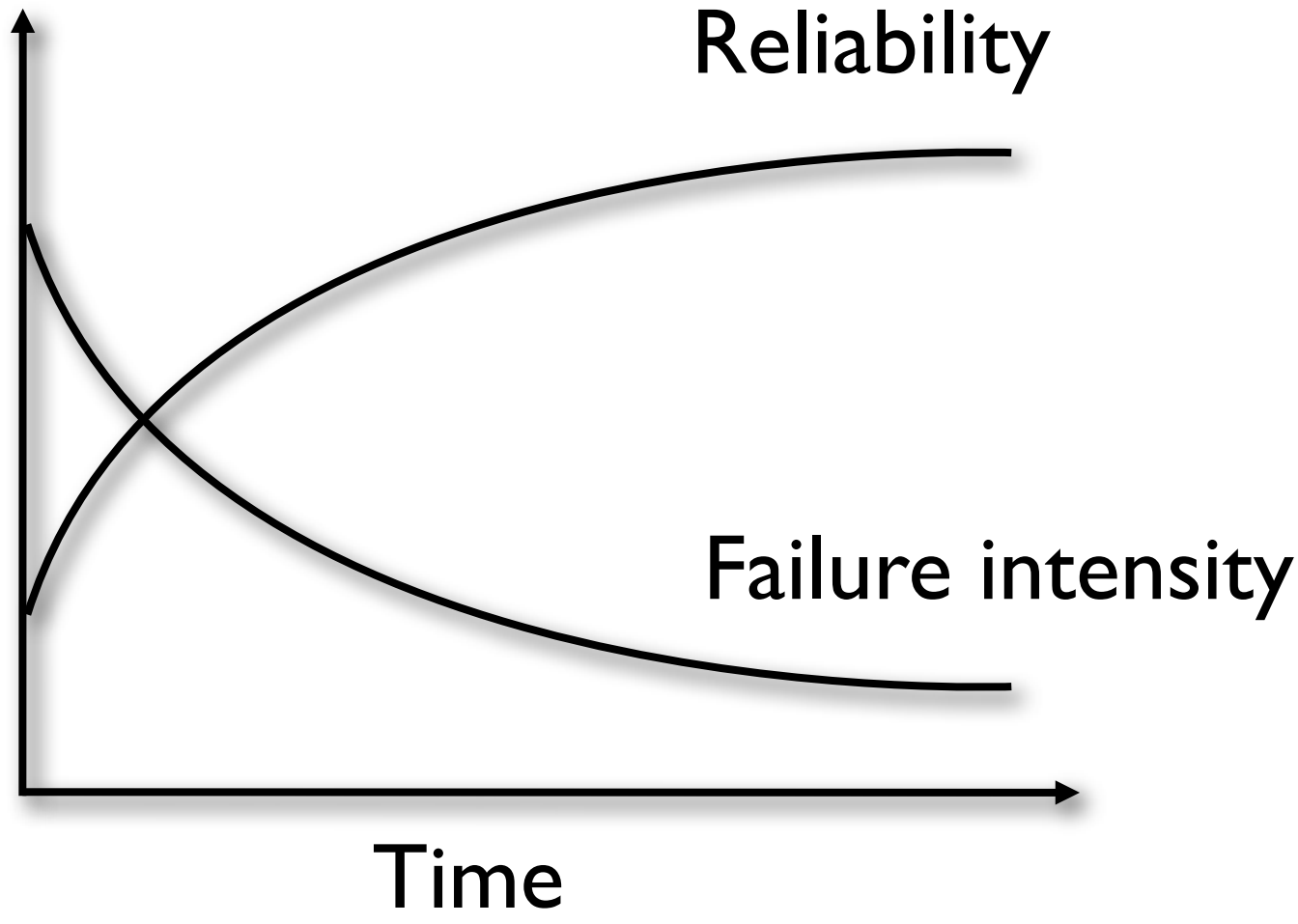
Most of them come from hardware reliability engineering.

MTTR, for example, is mostly interesting in high-availability systems. Otherwise, most software

systems are not optimised for MTTR.

ROCOF is a synonym for failure intensity

Reliability changes



In reliability models, the most commonly used measure is failure intensity. It describes the

number of failures in a certain time period. Interesting is also to use other means to describe

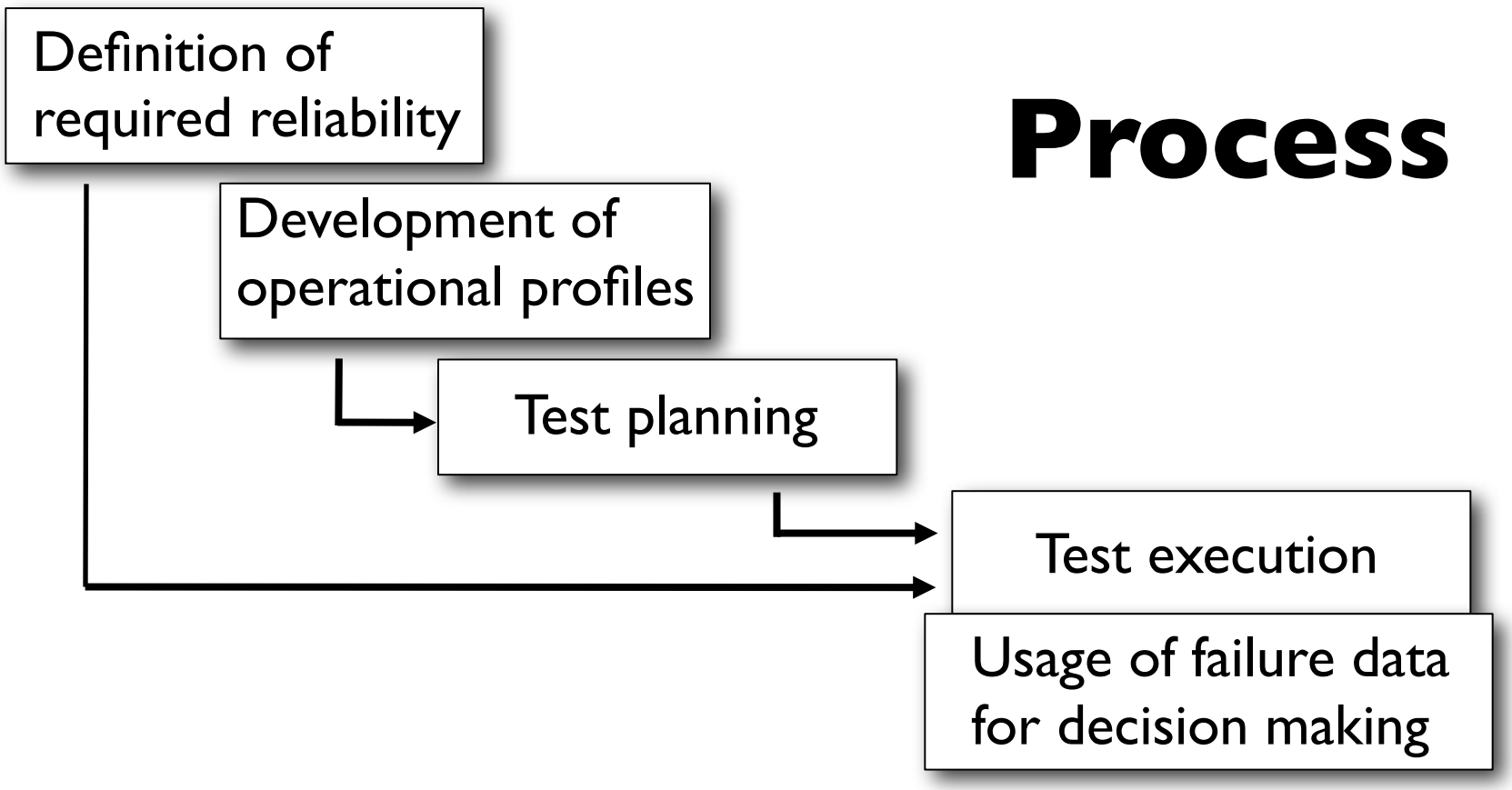
time periods. For example, in a telecommunication system, failure intensity is often defined

as failures per incident where an incident is one call made with the system.

Reliability is the reciprocal value of the failure intensity.

This reliability growth over time only occurs if we fix defects.

Process



This is the process for software reliability engineering in a nutshell. The development process

is reduced to requirements specification, design and implementation, and test. During the

requirements specification, we define the required reliability of the software to be built. Along with

it, we develop operational profile, i.e., how will the users work with the system? During design

and implementation we start with planning tests according to the operational profiles. The test

plan as well as the goal of the required reliability is the basis for executing the tests. The failure

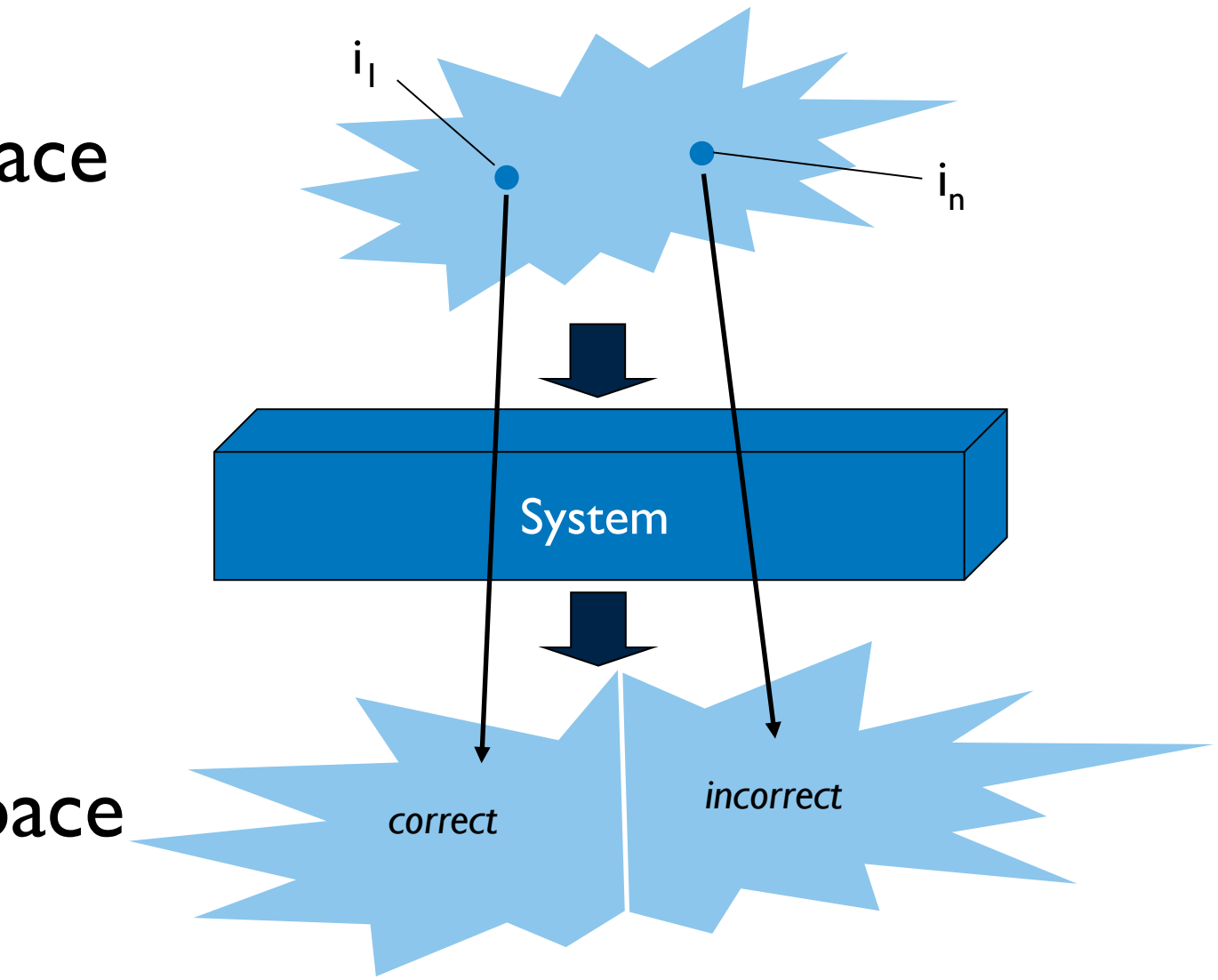
data from the tests (usually system and field tests) is used as basis for decision making.

This is usually called the "When to stop testing?" problem. When is testing finished? When have

I reached the required reliability?

Testing less or more can be expensive!

Reliability theory



As we need to analyse the current level of reliability and how it will change, we need a theory of

reliability that is the basis for the analyses.

The simple model that is usually employed sees the system as a function that transforms values

from the input space to the output space. The output space is divided into correct and incorrect

values. If the system outputs an incorrect value, a failure occured.

The transformation from the input to the output is (usually) deterministic for a software system.

Where stochastics come in is the distribution of the input values. What input values are put into

the system is seen as a random process.

Reliability growth model: Musa basic

Parameter:

v_0: total number of faults

λ_0: initial failure intensity

$$\mu(t) = v_0 \left(1 - \exp\left(-\frac{\lambda_0}{v_0} t\right)\right).$$

$$\lambda(t) = \lambda_0 \exp\left(-\frac{\lambda_0}{v_0} t\right).$$

μ(t): number of failures up to time *t*

λ(t): failure intensity at time *t*

There are various models that formalise this random process based on different assumptions.

A well-known model is the Musa basic model developed by John Musa. This model assumes there

is an exponential change of the initial failure intensity over time that is influenced by the total

number of faults that were initially in the system.

This allows to calculate the number of failures that will have occured at a time t in the future as

well as the failure intensity that the system will have at time t.

Reliability growth model: Musa basic

We have a program with an initial failure intensity of 10 failures/hour and 100 faults in total.

How many failures will have occured after 10 hours? How high is the failure intensity afterwards?

$$\mu(10) = 100(1 - \exp(-\frac{10}{100} \times 10)) = 63 \text{ failures}$$

$$\lambda(10) = 10 \times \exp(-\frac{10}{100} \times 10) = 3.68 \text{ failures/hour}$$

This example is simple, because we calculate only with hours. The difficulty in practice usually

lies in finding a useful measure for time, because only passing clock time does not make failures

occur. The system has to be used. Therefore, the notion of time should represent this usage

somehow. For a web server, this could be number of requests served.

In practice, we do not have number of initial failure intensity and total number of faults. This

is either done by estimating from earlier, similar projects, or by using the first data from system

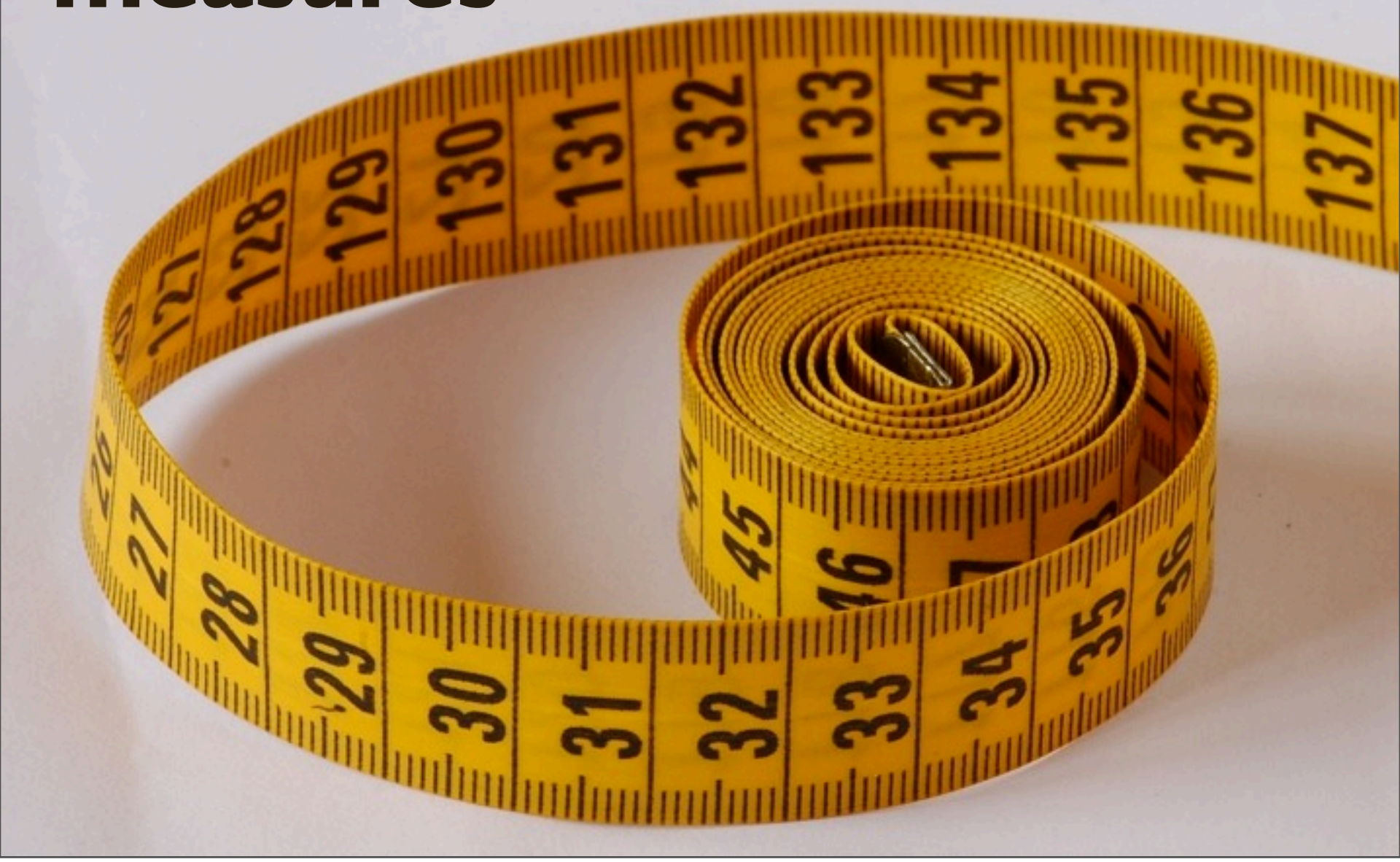
tests to fit the failure intensity curve. This is done, for example, using the least squares method.

Reliability models

Quality measures

Visualisation

Quality measures



Exercise

- Each of gets names of quality measures.
- Look on the Web for information.
- Make yourself an expert and find an example.
- Which quality attribute does it measure?
- 15 Minutes
- You will present each metric.
- Assign it to one of Garvin's quality approaches (on the white board).
- You can discuss with your neighbours.

34 measures

The assignment of measures to the user, product, or process level is not always easy. Some measures,
such as "Lenght of method" clearly measure directly something of the product. Others, such as
"Percentage of successful bug fixes" says mostly something about the process, but also about the
product.

Reliability models

Quality measures

Visualisation