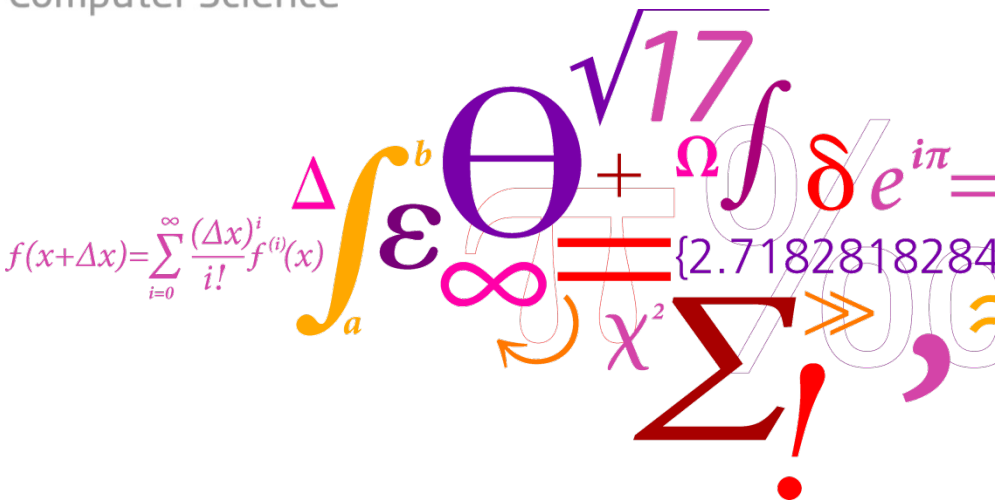
Software Engineering 2

A practical course in software engineering

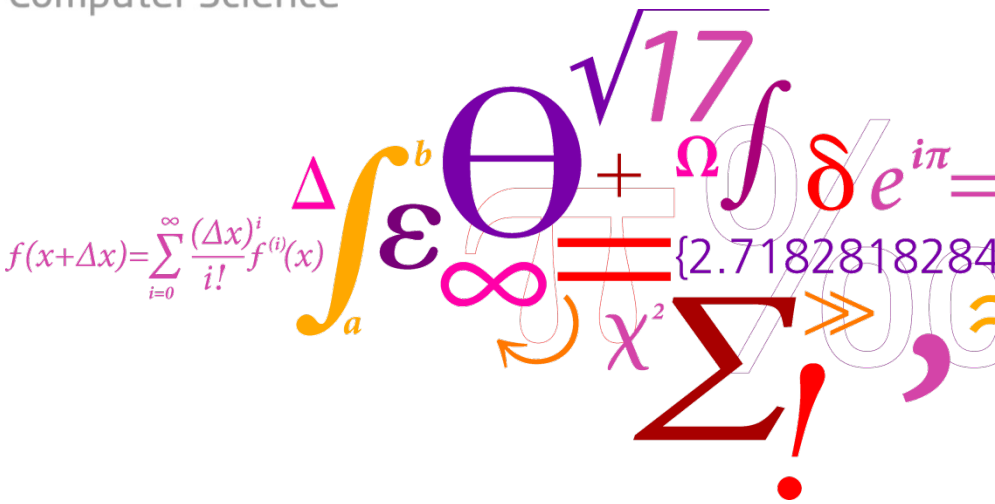
Ekkart Kindler

DTU Compute
Department of Applied Mathematics and Computer Science



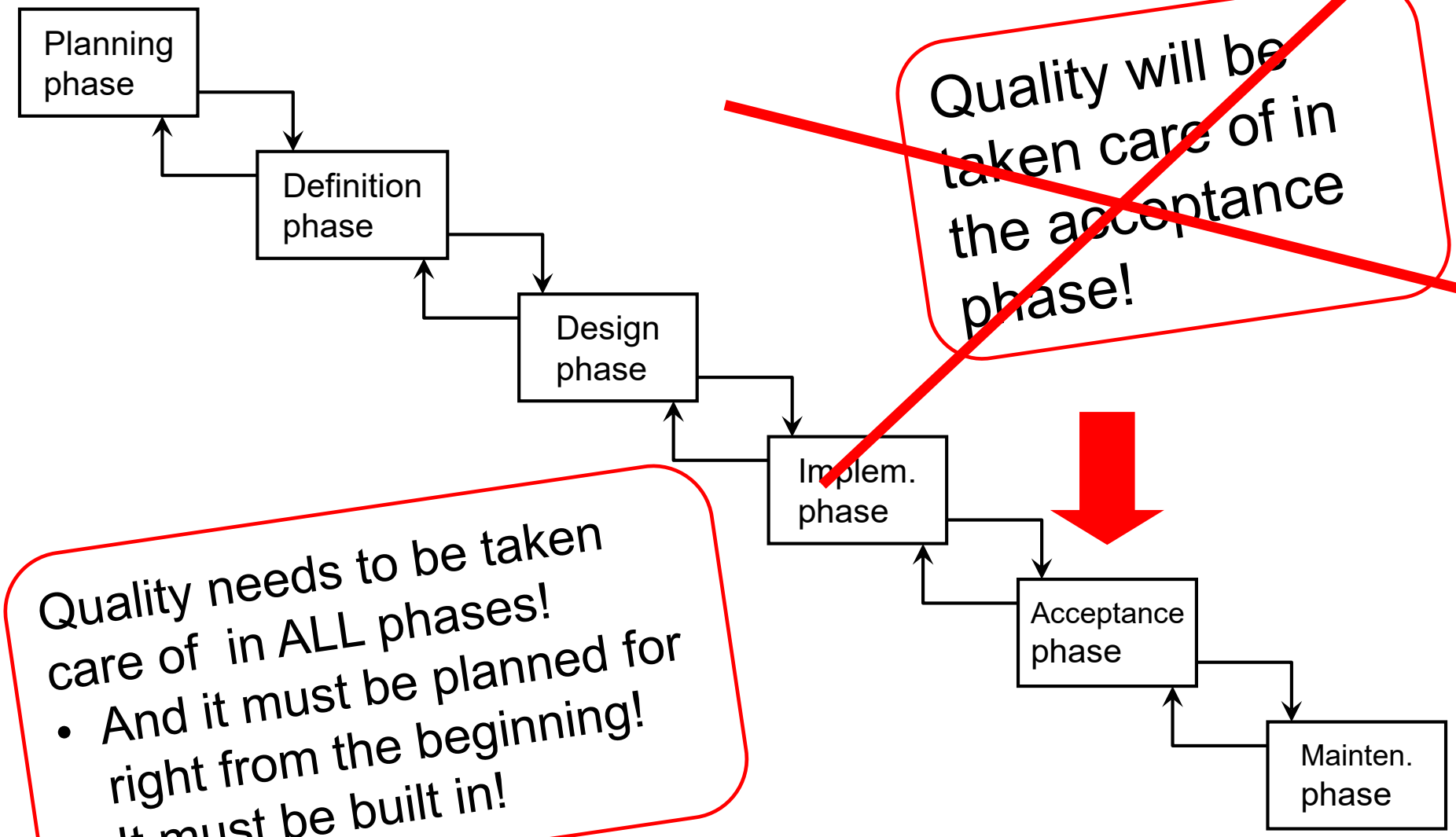
VII. Quality Management

DTU Compute
Department of Applied Mathematics and Computer Science



Main Message

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler



1. Overview

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Questions

- What is quality?
- How can we measure it?
- How can we “produce” it?

Quality

The totality of features and characteristics of a product or service that bear on its ability to satisfy stated or implied needs
(ISO 8402, ISO 9126)

This also applies to documents of the development process; their quality must be assessed too. In this sense, documents are considered to be „products“ too.

Quality Characteristics (ISO 9126)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- A set of attributes of a product by which its quality is described and evaluated
- A quality characteristic may be refined into sub-characteristics (over several levels)

Quality characteristics (ISO 9126)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Functionality
- Reliability
- Usability
- Efficiency
- Maintainability
- Portability

Sub-characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Functionality
  - Suitability
  - Accuracy
  - Interoperability
  - Compliance
  - Security

Sub-characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Reliability
  - Maturity
  - Fault-tolerance
  - Recoverability

Sub-characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Usability
  - Understandability
  - Learnability
  - Operability

Sub-characteristics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- ...

Features & Metrics

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Feature**: property for assessing the quality (sub-)characteristics of a product

  Comments, structure of the code, number of methods or classes, …

- **Quality metric**: a quantitative scale and method which can be used to determine the value a feature takes for a specific software product

→ Software metrics

Quality requirement

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Defines, for every feature, the minimum quality metrics to be reached (quality level)

Quality Assurance (QA)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- All actions to provide confidence (assure) that the product meets the quality requirements

Example: Action

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Tests are an action for QA

Problem:

- Tests can assert quality
  (if it is there)
- But test do not “generate quality”
  (they can only sort out products with bad quality)

Quality Management (QM)

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- is much more than just quality assurance!
- Quality management comprises all measures and actions to “generate” and “assure” quality

Quality needs to be **planned**, **controlled** and **assured**!

see slide 3

Quality Planning

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Which product (part) needs to be checked

- when
- by whom, and
- with respect to which quality requirements!

2. Principles

- Product centred QM:

  Quality will be assured directly at the product (→ QA).

- Process centred QM:

  “Quality of the process” assures that the produced product has the required quality (→ ISO 900x, CMM, …)

QM in Agile?

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Discussion:

- Which QM principle does agile development follow?

Experience shows:

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Purely product oriented QM turned out to be impractical for software development!

- Constructive measures

  during the development take care that, at the end, quality requirements are met

- Analytical measures

  check, at the end, whether the product meets the quality requirements

Constructive Measures

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- **Notations**
- **Methods**
- **Tools**
- …

→MDA
→Code generation

Can enforce syntactical conventions!

- Predefine schemas or templates
- Conventions
- Standards
- Check lists
- …

Analytical Measures

- Testing procedures

  run the product (program) for checking the quality

- Analytical procedures

  asses the quality without executing it (in particular for documents)

Testing Procedures

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Dynamic test
- Simulation
- Symbolic tests
- …

Analytical procedures

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Program analysis (static analysis)
- Program verification
- Review (→ Section 3.1)
- …

The transition from testing to analysing procedures is continuous!

Ex.: Model checking, slicing, symbolic execution …

Principles

- Explicit definition of the quality requirements and quality planning
- Constructive Measures
- Early and continuous
- Independent
- Quantitative (metrics/measurable)

Early measures

- The earlier errors are found, the less follow-up costs it will cause

→ Errors should be detected as early as possible

Independent QA

- Nobody likes to invalidate the own product ("psychology of testing")!
- If you forget a special case while programming, you are likely to forget to test exactly this case too!

→ QA actions should not **only** be taken by the developer or programmer himself

3. Assessment Methods

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- "Review"
  - Audit
  - Inspection
  - **Review**
  - Walkthrough
  - …
- Testing
  - Coverability
  - Unit tests
  - Integration tests
  - Acceptance tests
  - …

3.1 “Review“

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- More or less formal process, with the goal to identify errors, inconsistencies, ambiguities (weaknesses in general) of a document
- To this end, the document is inspected and discussed in a **systematic** way (with the authors present)
- The result is a review report (recording the tracked problems) or the release of the document (possibly after several iterations)

Psychology of “Reviews”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Problem:

- Authors “are in the line of fire” or are “grilled”

- Psychological aspects need to be considered: e.g.
  - No superiors present
  - No evaluation of persons based on reviews
  - …

Inspection

- Very formal form of a „review“

Participants:

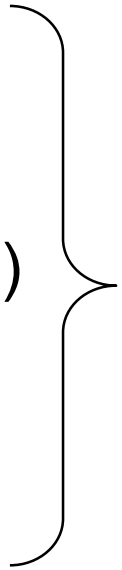
- Moderator
- Author
- Reviewer
- Recorder

Inspection

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Procedure:

- Initial check
  (Moderator can refuse inspection)
- Planning
- Individual review
  (by reviewers)
- Inspection session
  (all participants; result: records)
- Revision
- Final check
- Release



iteration
(if necessary)

Review

- Simple form of a “review“

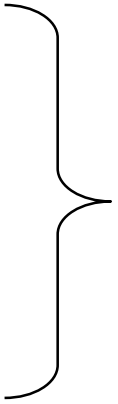
Participants:

- Moderator
- Authors
- Reviewer
- Recorder

Review

Procedure:

- Individual review
  (result: comments on document)
- Inspection session
  (result: record)
- Revision
- Final check



Iteration

Walkthrough

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Informal version of “reviews”

Participants:

- Author
- Reviewer

Walkthrough

Procedure:

- Maybe, individual review
- Inspections session
  (author moderates; result: record)

3.2 Testing

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

„Testing is the execution of a program in order to find (as many as possible) errors“

Test

- A test should find deviations between the actual and the expected behaviour of a program
- A **test** consist of a set of **input data** along with the **expected result**
- Running a **test** means **executing the program** with the input data and comparing the actual result with the expected result

Testing and QA

- Tests can **increase the confidence** in the **absence of errors**; this provides a measure for
  - Accuracy (functionality) and the
  - Fault-tolerance and maturity (reliability)

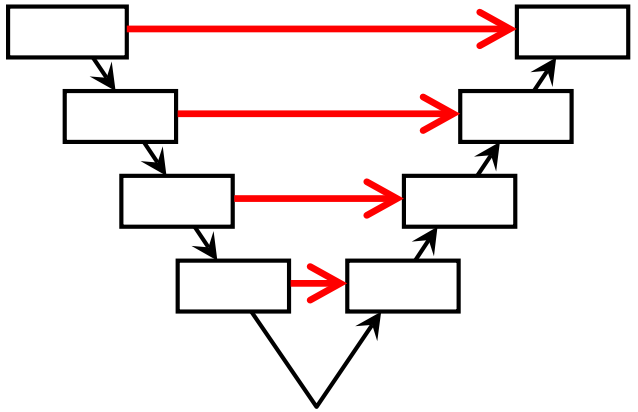
  of the software

Test Levels

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

There are different levels of test:

- Acceptance tests (by/with client)
- System tests
- Integration tests
- Unit tests



Automation

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Regression tests should guarantee that the quality of the software does not "regress" over time.

- Test can be executed automatically (→ JUnits)
- Regression test will be executed automatically whenever some part of the system was changed (in order to ensure that the quality of the software does not regress).

  Then, a system will be released only when all regression test were passed successfully; this guarantees that supposed corrections did not inject new errors (→ JUnits)

- Concurrent (multi-threaded) programs and GUIs are still problematic

The Limitations of Testing

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Tests can only show the presence of errors
- Tests cannot guarantee the absence of errors

→ Absence of errors can be shown by verification; but, complete verification of software is impractical today (but sometimes required for parts)

Questions

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- What are good test?
- How do I test properly?
- When did I test enough?
- How can I make sure to find as many errors as possible?

→ Systematic testing (see below)

→ Principles of testing (see below)

Principles of Testing

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

- Author does not (exclusively) test
  (psychology of testing)
- Expected result should be defined before executing the test
  (define tests early → XP/agile: before implementation)
- Rigorous testing
  - Check result carefully
    (at best automatically)
  - Check “everything”
    (→ systematic testing)
  - “over and over again” testing
    (regression tests: executed after every change for the complete system)

Systematic Testing

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Systematic construction of tests

- **Black-box Test** from specification (without knowing the implementation):
  - Normal cases from specification
  - Special cases from specification
  - Illegal input from specification
- **Glass-box Test** from implementation:
  - Normal- and special cases from program conditions (alternatives and loops)
  - Coverage criteria (→ next slides)

Coverage

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
if (x == 0)
  z = x-y;
else
  x = x/y;

if (y > 0)
  z = 27;
else
  z = y/x;
```

Statement coverage:
Number of (different) statement executed by at least one test divided by the number of all statements of the program.

100%: Every statement was executed at least once

Coverability

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

```
if (x == 0)
  z = x-y;
else
  x = x/y;

if (y > 0)
  z = y/x;
else
  z = 27;
```

100% Statement coverage:
Every statement was at least executed once

Tests with 100% statement coverage:

- x=0, y=0
- x=1, y=1

Coverage

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

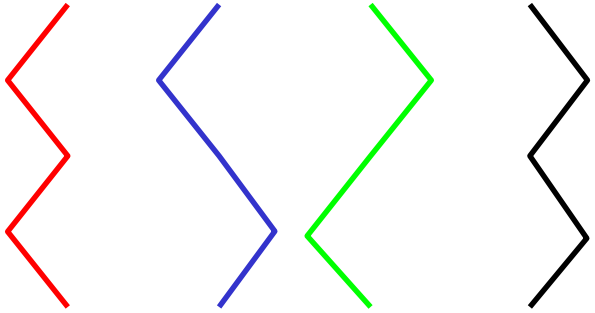
```
if (x == 0)
  z = x-y;
else
  x = x/y;

if (y > 0)
  z = y/x;
else
  z = 27;
```

100% Path coverage:
Every path of the program was executed at least once.

Tests with 100% path coverage :

- x=0, y=1
- x=0, y=0
- x=1, y=1
- x=1, y=0

Problem: Loops give rise to infinitely many paths! We don't go into details here.



Construction vs. Counting

- For glass-box test, we can construct test in such a way that a specified coverage will be reached
- By instrumenting our software, we can also "count" which coverage is reached (if it is not high enough, we can add further tests to the test set; some test suits do this automatically)

- Both approaches can be combined

Warning

- Even if you have achieved 100% path coverage, this does not guarantee that all errors are found!!!!

- Nevertheless this is a quality metric for the sub-characteristics “Accuracy” of the product and increases the confidence in the quality of the product

Test Driven Development

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

eXtreme Programming (agile) mindset:

- Only features that are tested by an **automated test** "exist"
- Each user story is associated with a set of tests (**customer tests** / **functional tests**)
- Tests for a user story are written before you begin implementing the user story
- The implementation of a user story is finished, when the tests associated with it run through
- Each method should be associated with a test (**programmer tests** / **unit tests**)

Agile Test Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it
- Unit test are **automated**, and **run every time** new code is checked in
- **Everybody** is responsible for **fixing** broken code (code which results in failed tests)

Agile Test Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it (tests first)

This helps with becoming clear of what should be implemented and how exactly the interface should be!

Raises level of detail and makes things technical.

Agile Test Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it
- Unit test are **automated**, and **run every time** new code is checked in

Indicator of quality all the time!

Agile Test Strategy

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Test driven development:

- **Before implementing functionality**, you write its interface and tests for it
- Unit test are **automated**, and **run every time** new code is checked in
- **Everybody** is responsible for **fixing** broken code (code which results in failed tests)

100% success rate (for unit tests) is the norm!

Testing

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Two sources of tests:

- **Programmers**:
  Unit tests for everything which potentially or likely could be wrong (or which went wrong at a some time)
- **Customers** (maybe implemented by programmer):
  Functional tests for user stories

Example: When someone moves in front of the Q-motion, the Q-plug is switched on; after a while without motion, the Q-plug is switched of!

Tip: For testing complex user stories with hardware, it might be worthwhile to realize an emulator for hardware devices!
→ slide 60/61

Testing

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Two sources of tests:

- **Programmers**:
  Unit tests for everything which potentially or likely could be wrong (or which went wrong at a some time)
- **Customers** (maybe implemented by programmer):
  Functional tests for user stories

Functional / integration tests fail for some time. But should be eventually fixed. Functional tests might not run all the time; but, someone (→ **roles**: tester) is responsible for running the tests an a regular basis.

Other tests

Unit tests and functional tests are the most important tests. But, others might be relevant

- Parallel tests (compare an old and a new system)
- Stress test (performance)
- Monkey test (unexpected input)
- ...

Testing in the “real world”

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

Problem:

- When hardware and devices are involved, it is difficult to test some functionality automatically:

Example:

rule engine: when CO2 level rises above 1000ppm, eventually the window is opened and the thermostat is set to “*” (or 4 degrees)

How do you make the CO2 level rise above 1000ppm by a test?

Solution

DTU Compute
Department of Applied Mathematics and Computer Science
Ekkart Kindler

For testing such functionality, implementing an emulator of hardware (in the case of the example from SE2 2019, a Raspberry Pi) with some sensors and actuators, which can be set manually (via a graphical interface) or programmatically (via an API), might help.

This year, it could be an emulator simulating the upload of new car data?