Software Quality Management

Dr. Stefan Wagner
Technische Universität München

Garching
14 May 2010

Last QOT: difference between functional and quality requirements

"Functional requirements can be built in a framework and be measureable. A quality requirement is harder to measure."

"Example: quality req: minimalistic design of GUI; func req: user friendliness of the interface"

"The difference could be that the functional requirement is more inclined towards developing processes & quality requirement is inclined towards developing product."

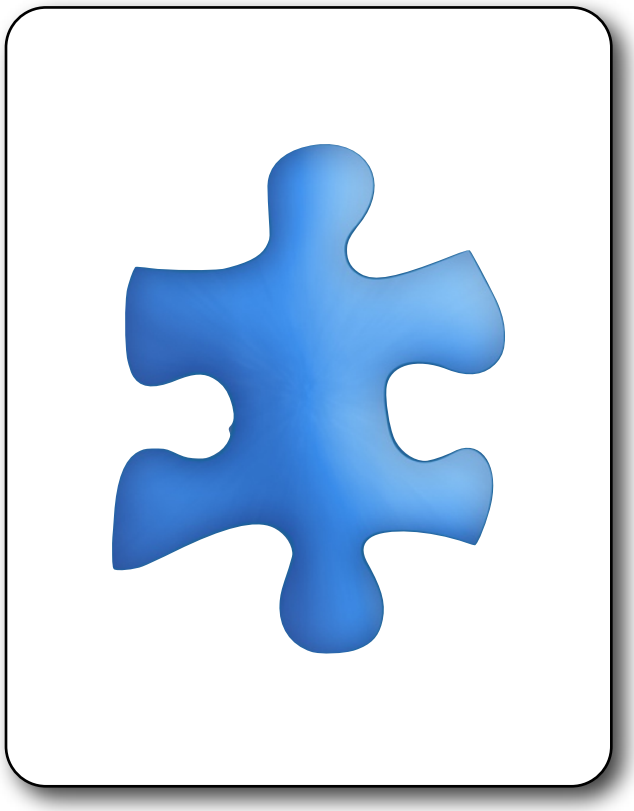
It is not possible to say in general that quality requirements are harder to measure than functional requirements. This may hold in certain cases, but not in all.

Both requirements in the example are quality requirements. The user friendliness of the interface talks about a quality, i.e., usability.

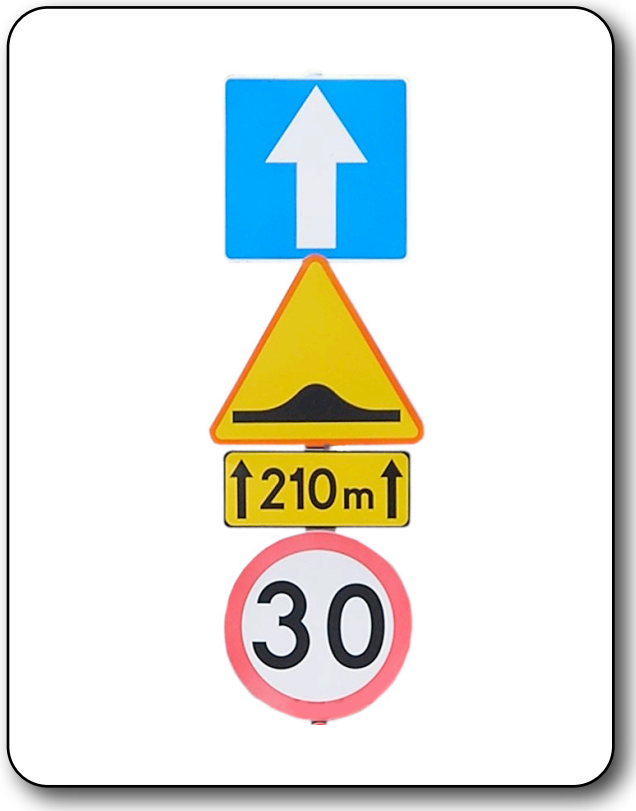
Process/product and quality/functionality are two different dimensions. Functional as well as quality requirements specify the product. The process should enforce that these requirements are fulfilled in the product.

New QOT: "What quality attribute is the hardest to evaluate with tests?"

Quality Models



Quality Require-ments

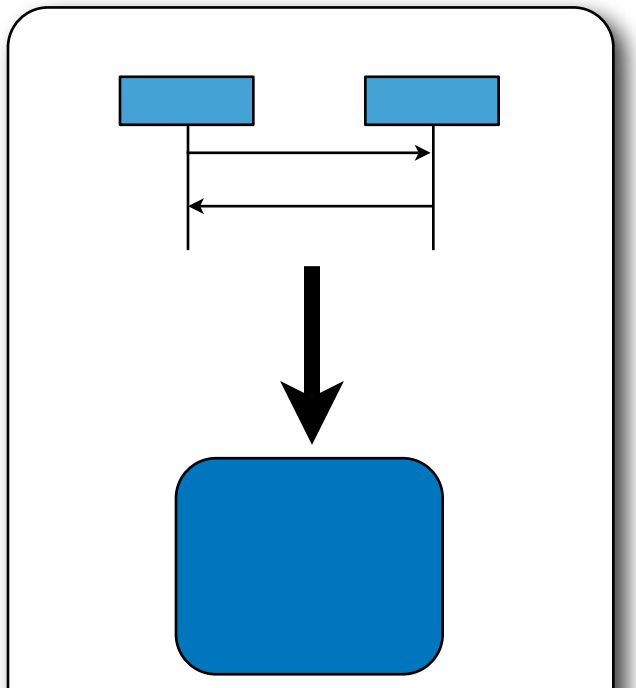


The last lecture covered quality models and quality requirements.

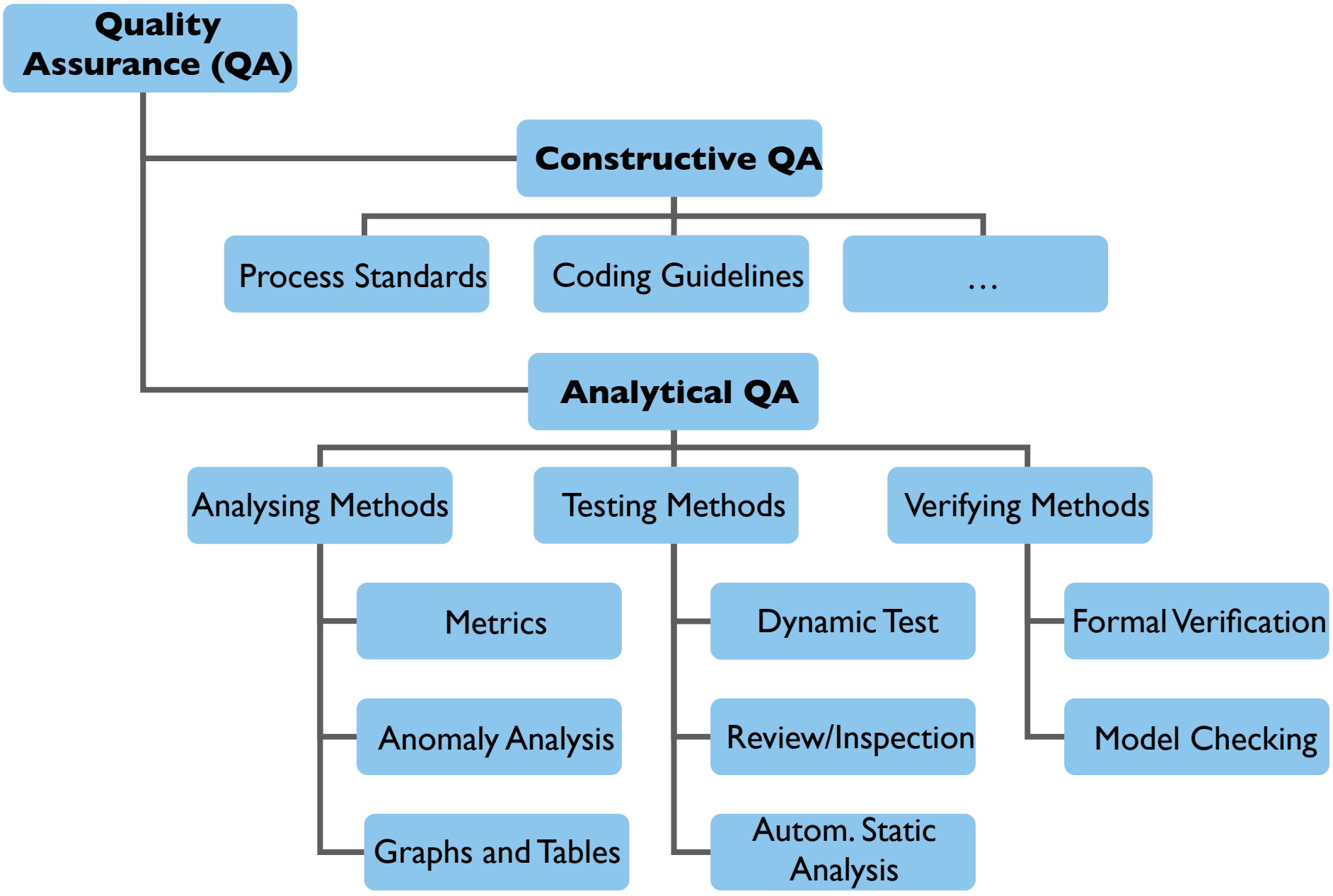
We are still in the part "Product Quality".

Constructive Quality Assurance

Testing



This lecture covers an overview of constructive quality assurance and a discussion how testing can be used to analyse quality properties.



A common classification of quality assurance (QA) is into constructive QA and analytical QA.

Constructive QA is anything you can do to "build quality in". This means all methods and techniques you use to achieve the desired level of quality while building ("constructing") the system. The goal is to avoid or prevent quality defects.

Analytical QA checks for quality defects in artefacts or the process. There are different ways to classify analytical QA. One possibility is into analysing methods, testing methods, and verifying methods.

Analysing methods are means to learn something about the current state of the artefact. It includes any kinds of metrics, anomaly analysis (for issues that are "not normal" or suspicious), and graphs and tables for an overview. It helps in the identification of areas in the artefacts that might have quality defects. An example is a list of very long methods.

Testing methods include the most common quality assurance methods: dynamic testing, reviews and inspections, and automatic static analysis. Dynamic tests involve defining test cases that give inputs to the system while executing and observing the output. Reviews and inspections are the method of reading the content of artefacts (requirements, design, code) often using a checklist. Automatic static analysis is a method in which tools are used to analyse the software without running it (e.g., its source code).

Finally, verifying methods formally or logically verify that the system conforms to its specification. The prerequisite is a formal specification of the desired properties. The conformance can be demonstrated by proof (formal verification) or by expanding the complete state space (model checking).

Examples for constructive quality assurance

Typing

Documentation

Design by Contract

Typing is the property of most programming language that a variable has to have a defined type, e.g., integer or string. If you then try to assign a string value to a variable of type integer, the compiler can find this problem and warn you.

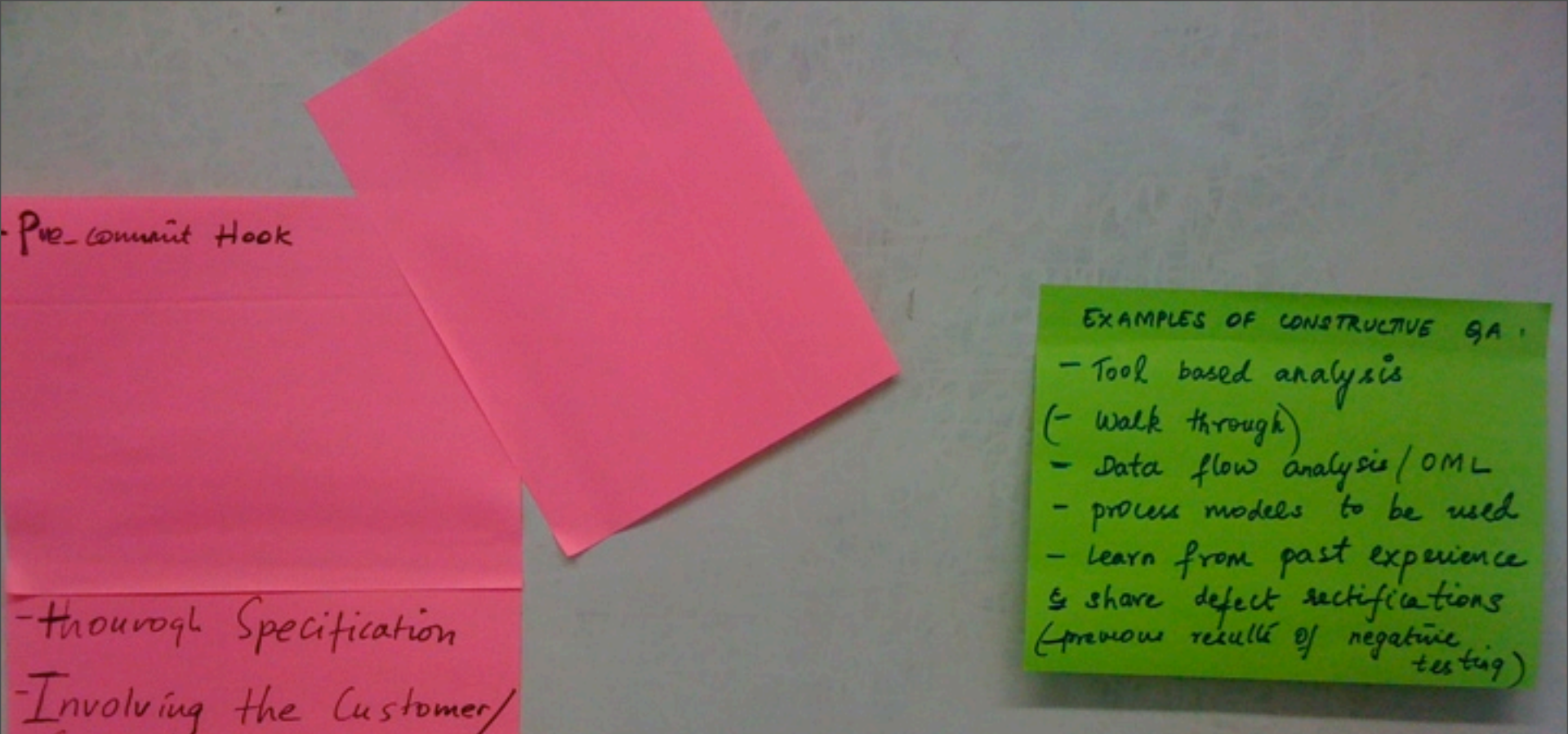
The documentation of the source code, e.g., inline comments or additional design documents, helps the maintainers of a system to understand it more easily and avoids problems that could be introduced by misunderstanding the code.

Design by Contract is the technique to specify pre- and post-conditions for functions or methods. This avoids interface defects between components as it is clearly specified what a function expects when it is called and what can be expected after it finished. For further information search for the work of Bertrand Meyer (ETH Zürich), the programming language Eiffel, and the Java extension JML.

Group work

What are further examples of constructive quality assurance methods?

10 minutes
On Post-its (Write large!)

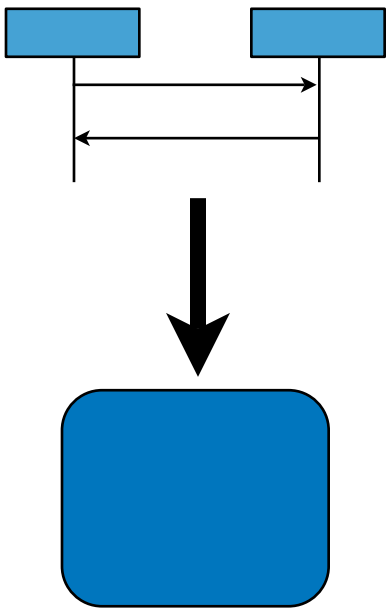


Group work results

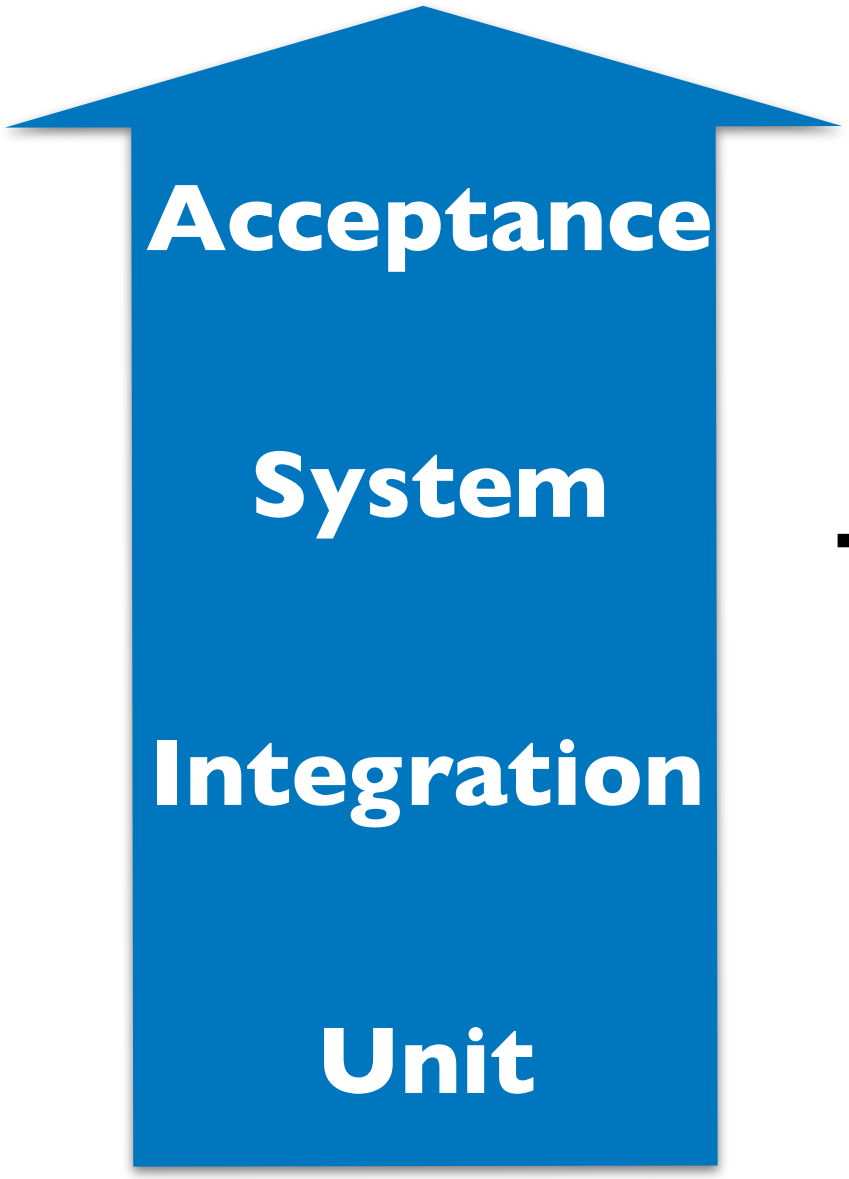
The difference of what is constructive and what is analytical is not completely sharp.

Constructive Quality Assurance

Testing



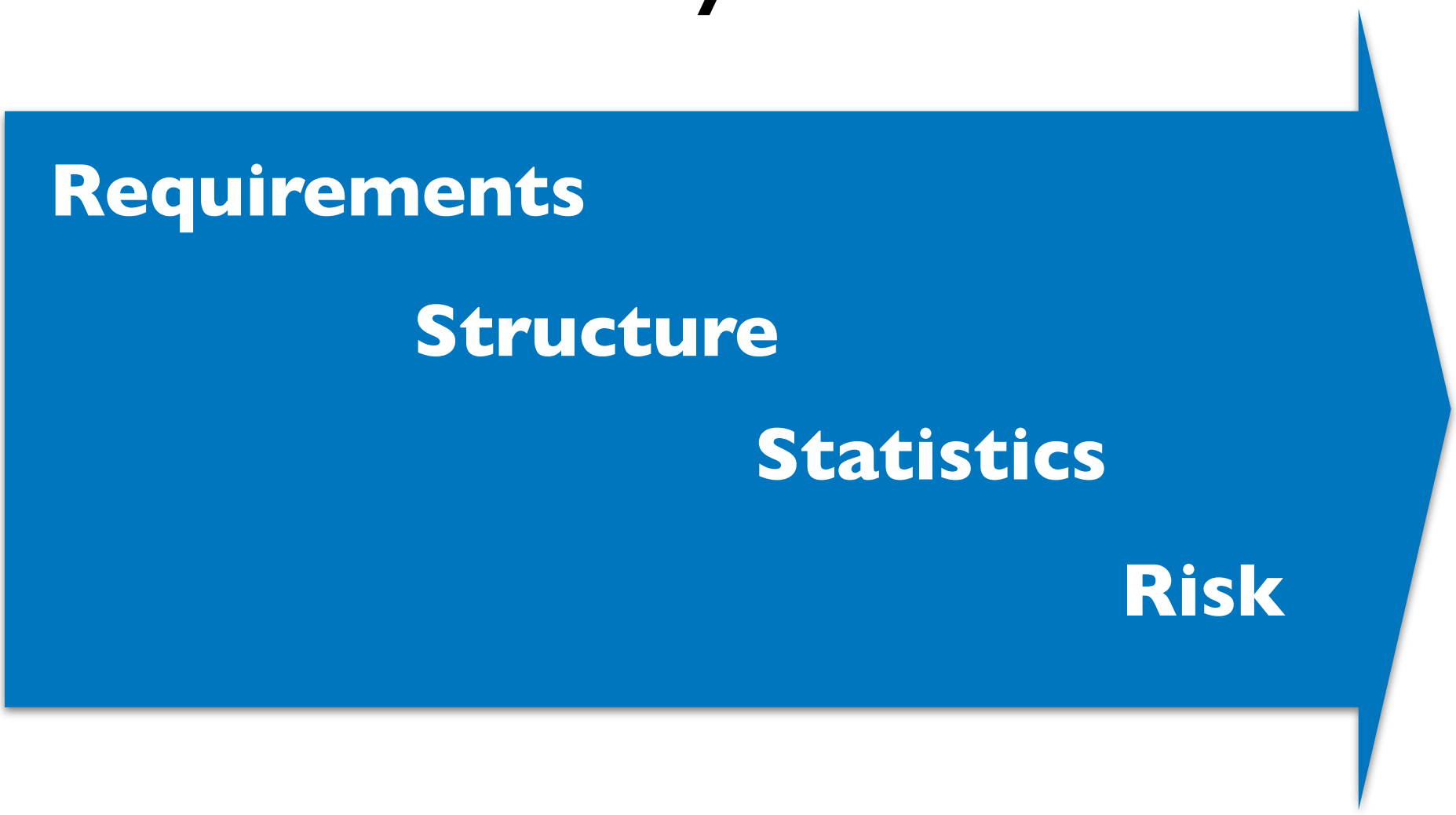
The following gives an overview of testing for quality attributes.
First, we classify testing methods along two dimensions.



Test phases

There are four test phases:
1. Unit testing means that single units or components (usually classes or modules) are tested seperately.
2. In integration testing, the single units are combined succesively to form larger units. In this phase, interface defects in the interplay of these units are detected.
3. The system test then tests the completely integrated system in the production environment or a test environment that is similar to the production environment.
4. The acceptance test is done by the customer usually after it was installed in the customer's environment.

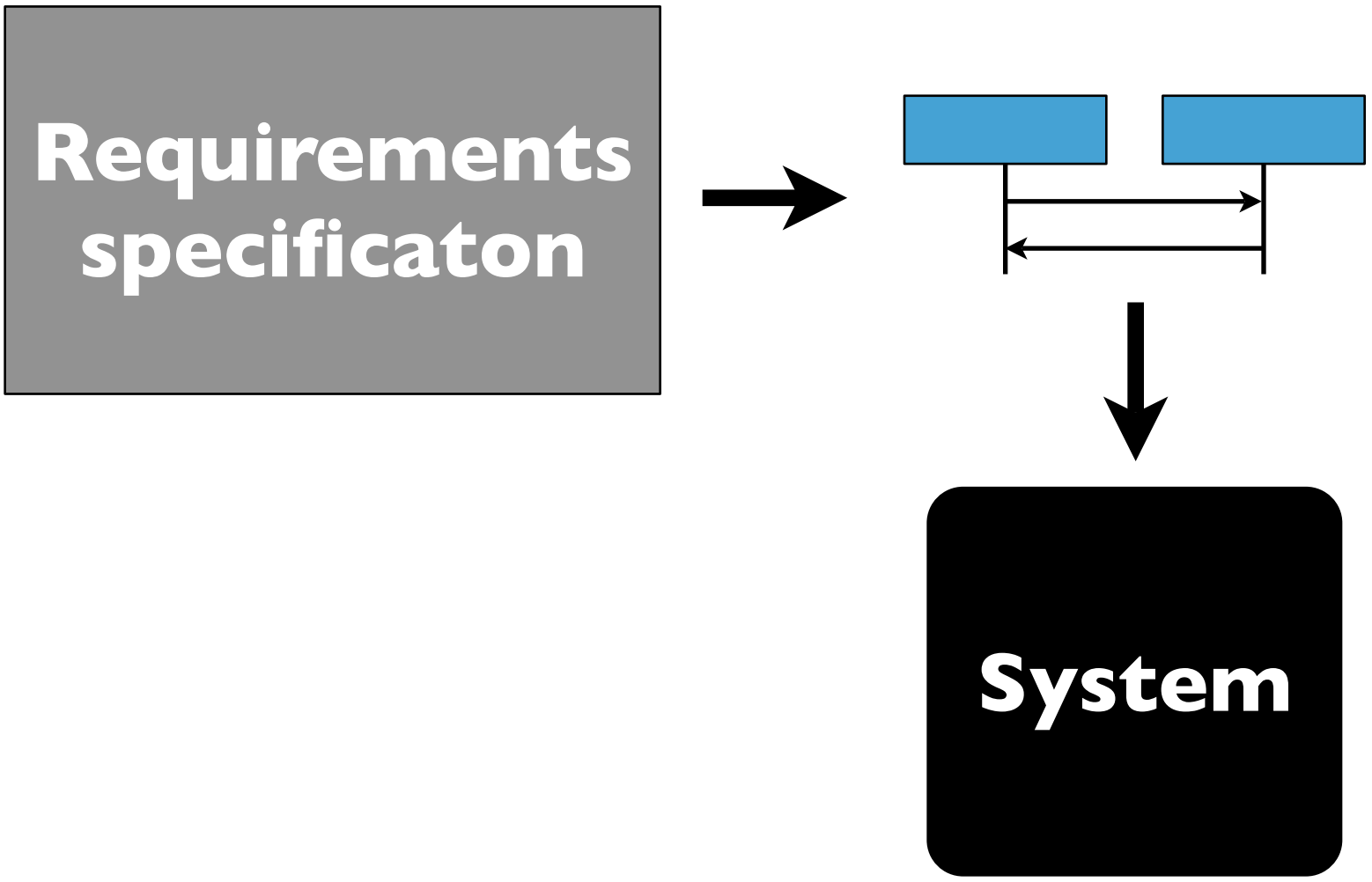
Tests driven by



Glass. A Classification System for Testing, Part 2. 2009

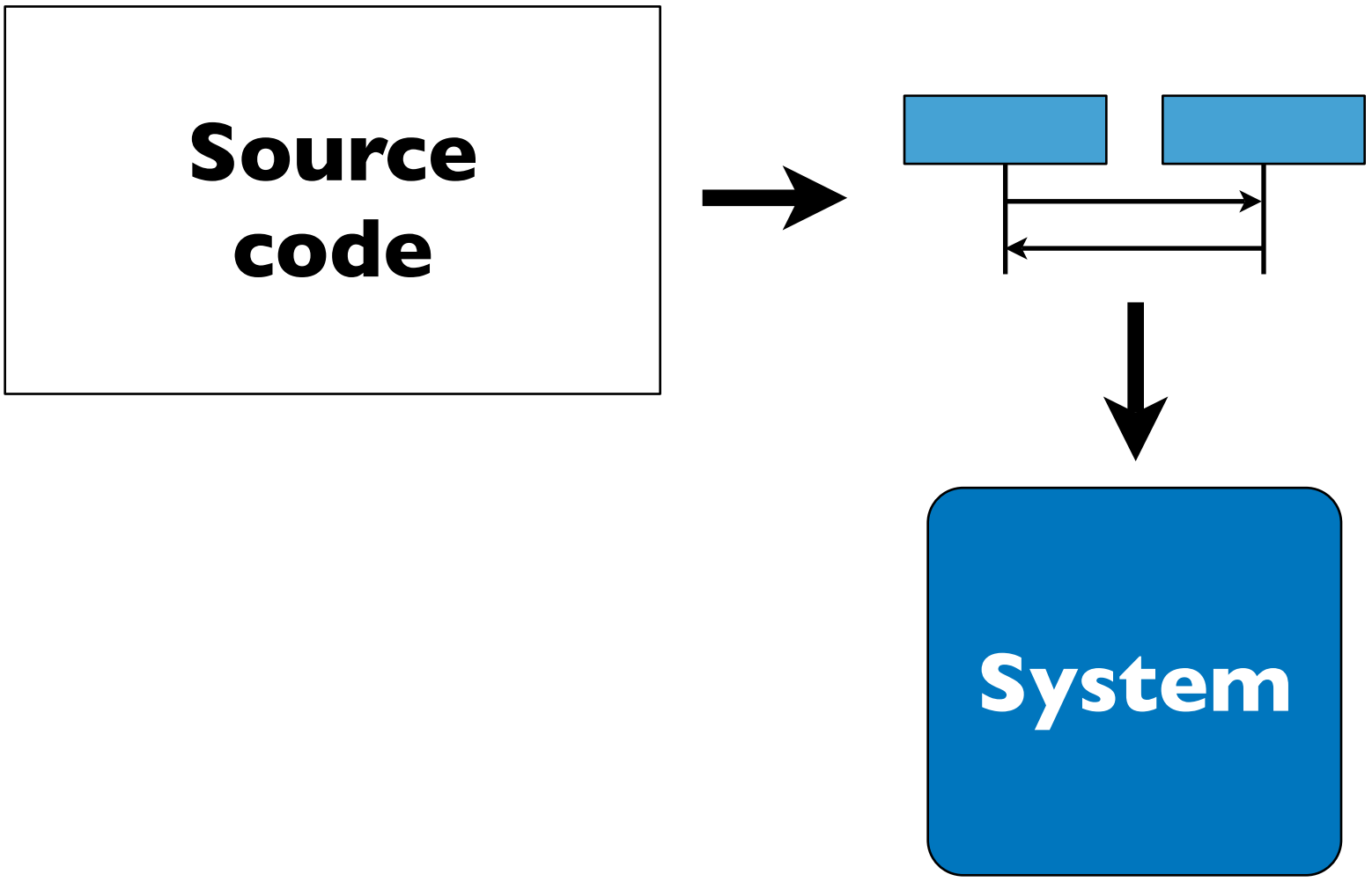
Following Glass (2009) there are four classes of drivers for specific tests:
1. Tests can be driven by requirements. The requirements specification is used to derive test cases usually with the intent to cover all requirements.
2. Test can be driven by structure. Most commonly, such tests follow the internal structure of the source code and they try to cover all aspects of this structure.
3. Tests can be driven by statistics. The inputs used in the test cases can follow a statistical distribution or be completely random.
4. Tests can be driven by risk. These risk can be diverse, e.g., financial risk of failure or high usage. The tests concentrate on the parts with high risk.

Black-box testing



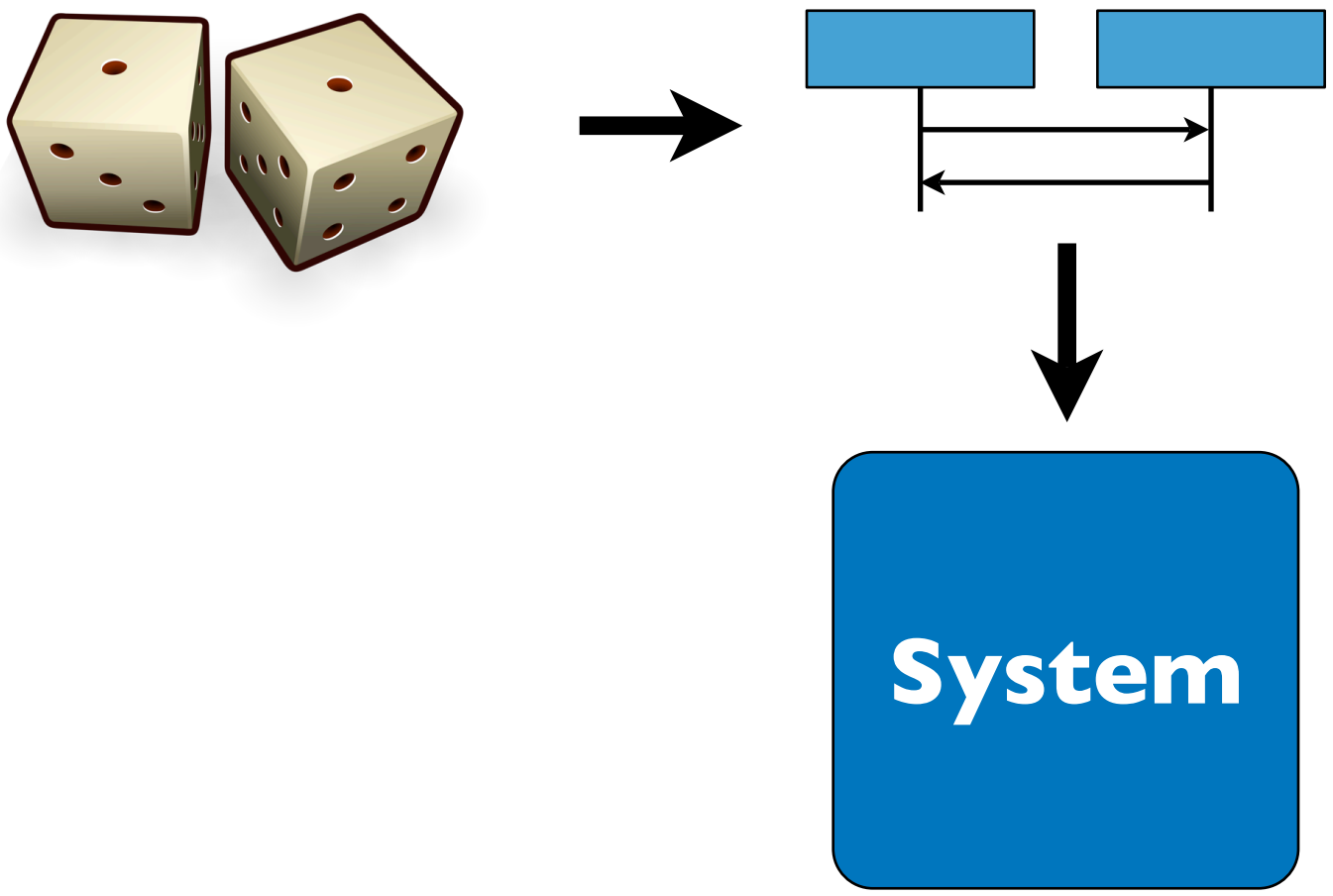
In black-box testing, the test cases are derived solely from the requirements specification. The test cases are run against the system and the outputs are compared with what is expected from the specification.

White-box testing



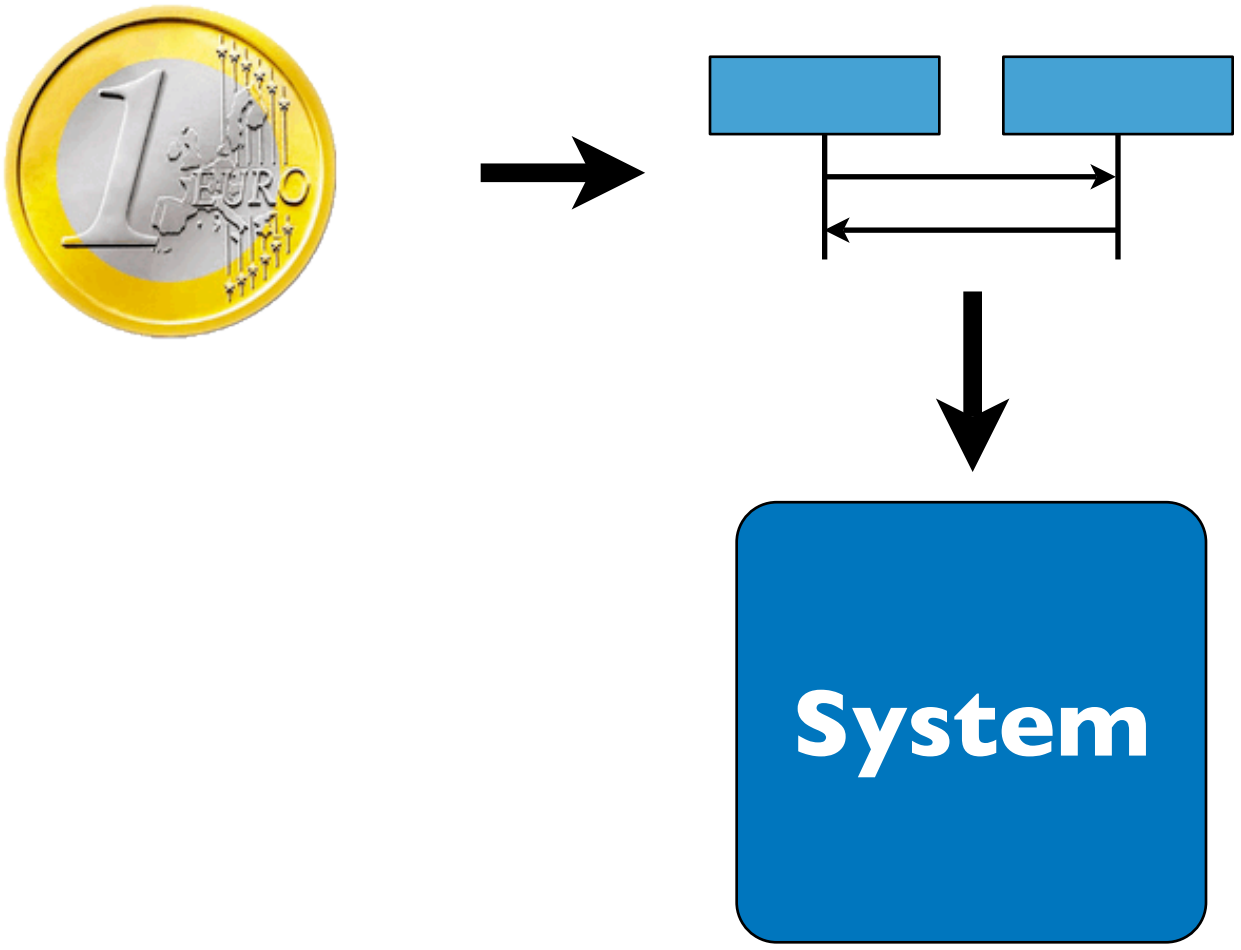
In white-box (or glass-box) testing, the test cases are derived from the internals of the system, usually the source code.
The test cases are specified in a way that at least this source code structure is covered (i.e., executed) by tests.
The outputs of the system are usually also compared with the expectation derived from a requirements specification.

Random testing



Random testing does not rely on the specification or the internals of the system.
It takes the interface of the system (or the unit if it applied for unit testing) and assigns random values to the input parameters of the interface.
The tester then observes whether the system crashes or hangs or what kind of output is generated.

Risk-based testing



In risk-based testing, the test cases are specified for those parts of the system that have a high risk.

The methods assess risks along a variety of dimensions:

Business or Operational

- High use of a subsystem, function or feature
- Criticality of a subsystem, function or feature, including the cost of failure

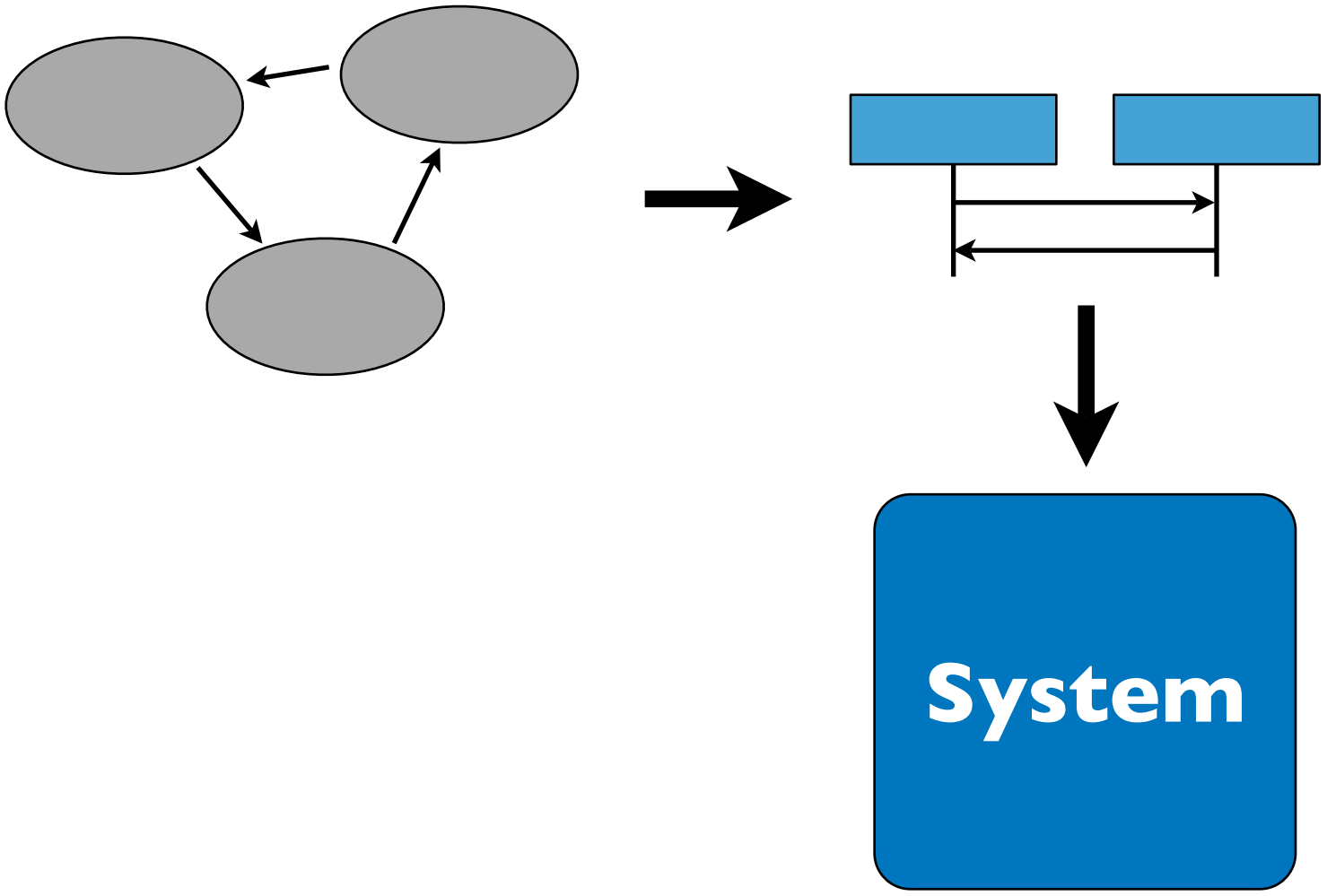
Technical

- Geographic distribution of development team
- Complexity of a subsystem or function

External

- Sponsor or executive preference
- Regulatory requirements

Model-based testing

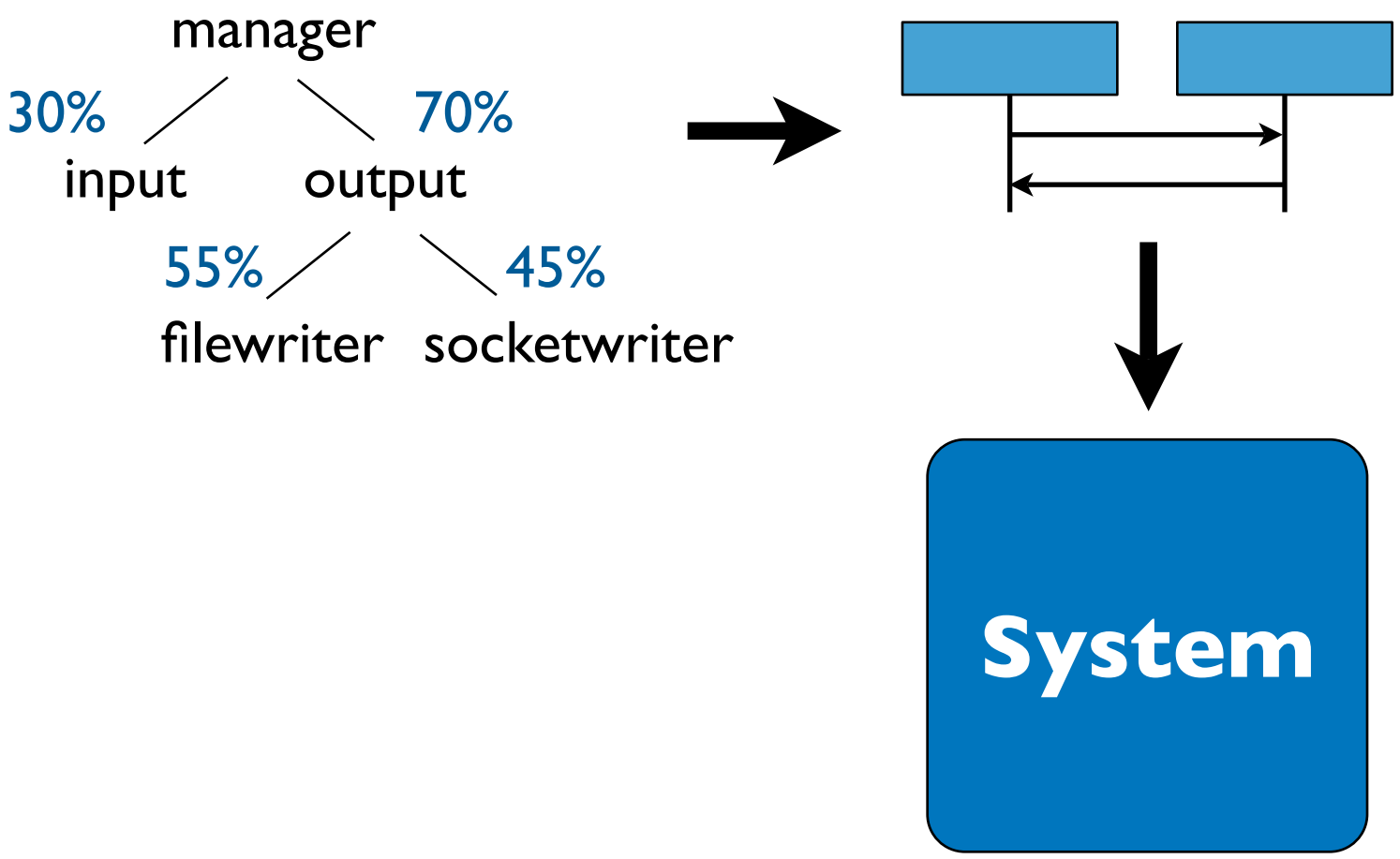


In model-based testing, the requirements specification is substituted by an explicit model. The model is significantly more abstract than the system so that it can be analysed more easily (e.g., by model checking).

The model is then used to generate the test cases so that the whole functionality is covered. For example, if the model is a state chart, state and transition coverage can be measured.

If the model is rich enough, the test case generation and even the expected output can be generated automatically from the model. The model then also acts as oracle.

Testing based on operational profiles

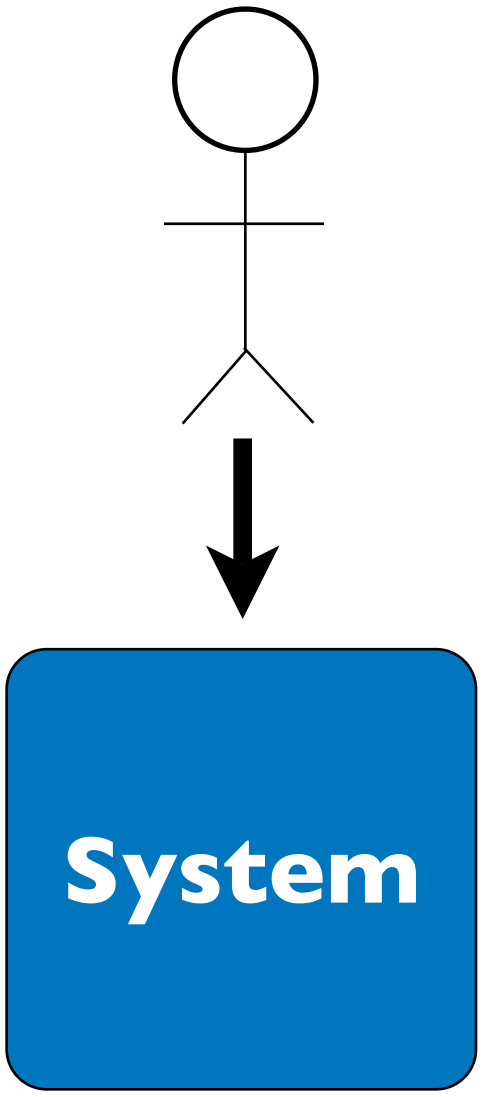


Operational profiles describe how the system is used in real operation. This usage then drives which test cases are executed.

In the example, "manager" is a component in the system. Its subcomponents "input" and "output" are used differently. In 30% of the usages, the "input" component is executed, in 70% the "output" component.

Tests driven by operational profiles act as a good representation of the actual usage.

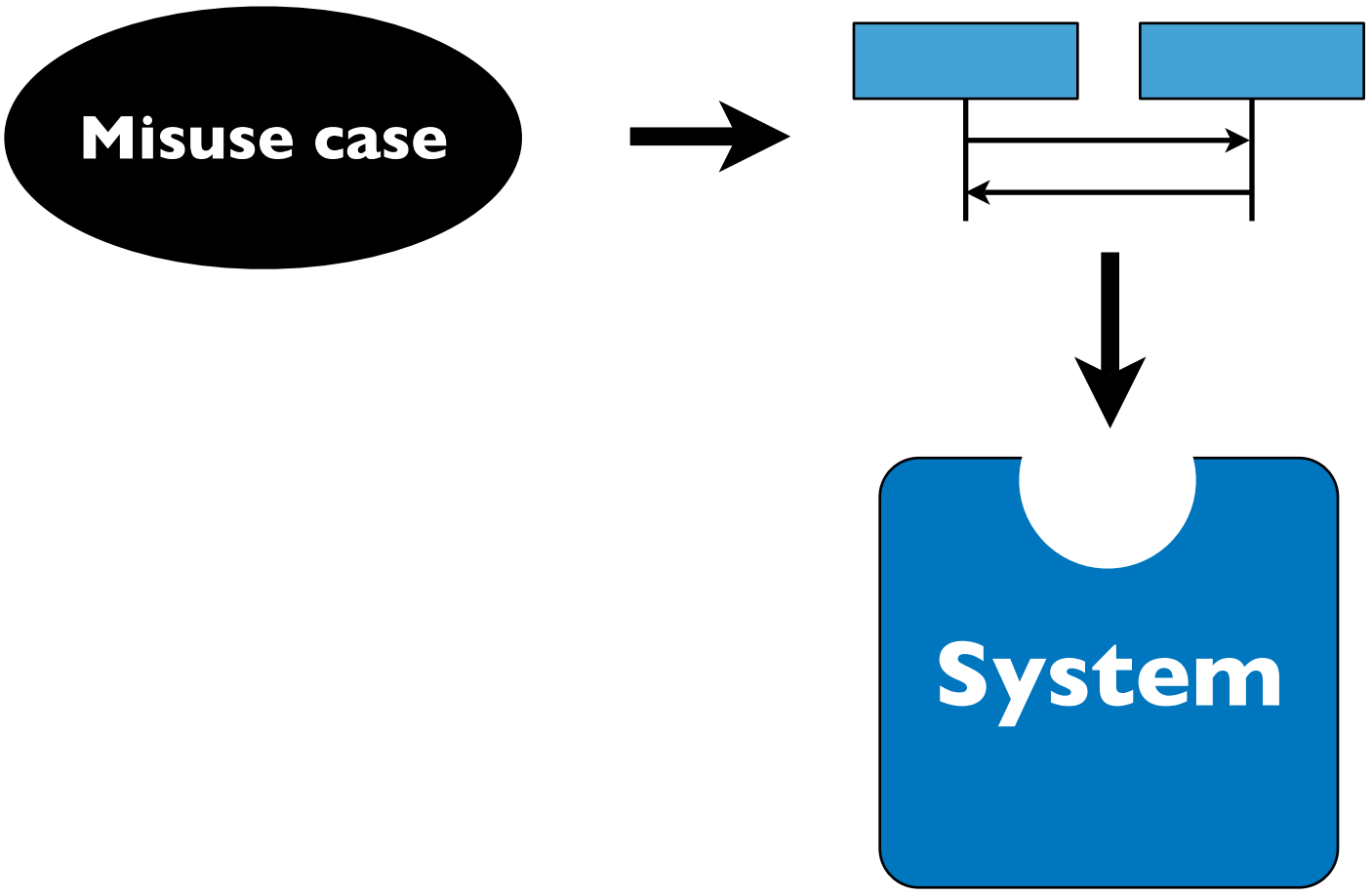
User testing



User testing means that an actual (future) user of the system is observed while interacting with the system.

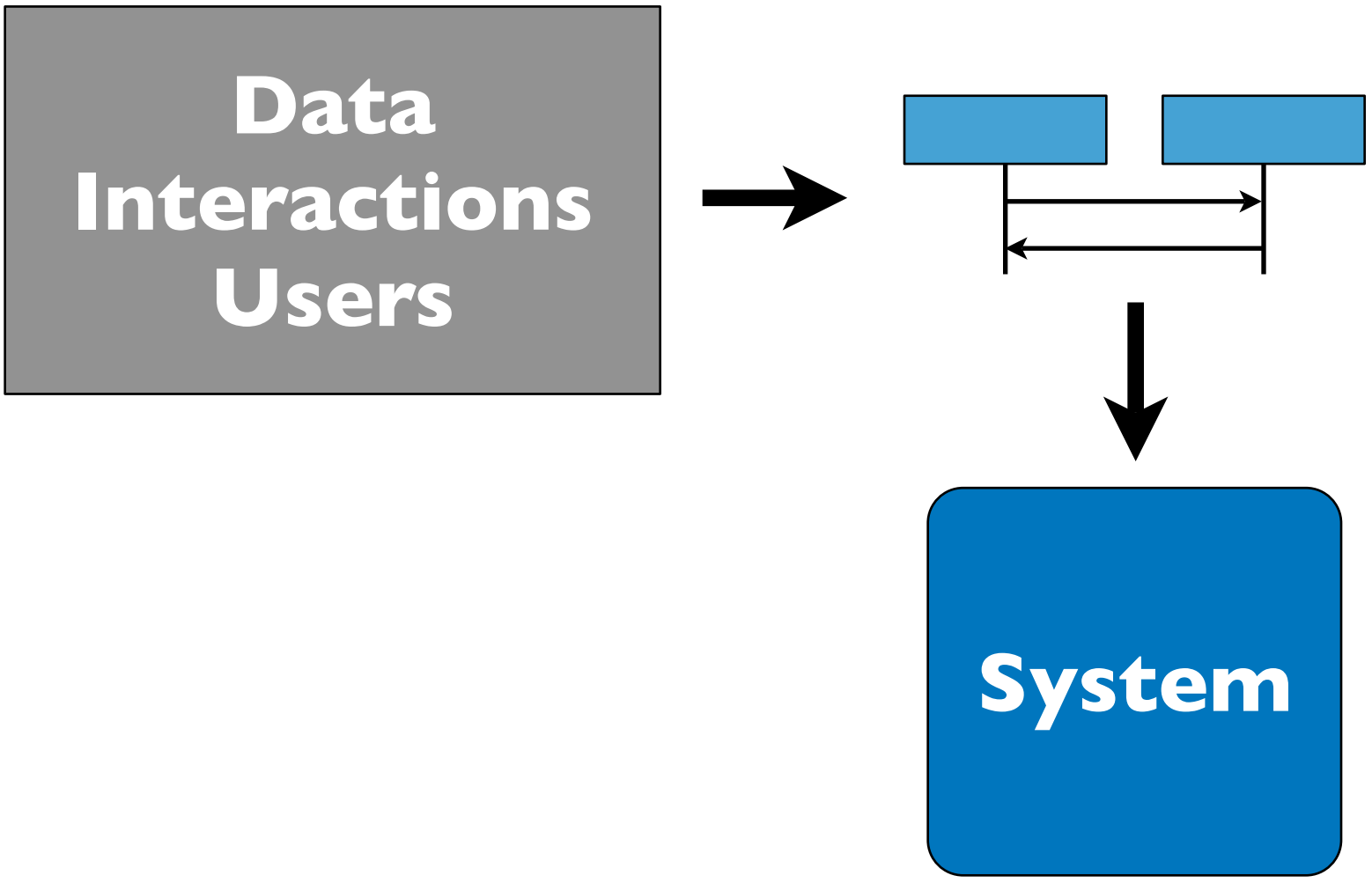
This can have different aspects to it. For example, a user can be just given the system without any further explanaitions to analyse how easy it is to learn the system.

Penetration testing



In penetration testing, misuse cases are the basis for test cases. Misuse cases describe unwanted or even malicious use of the system.
These misues are the basis for penetration the system to find vulnerabilities in it.
The result can be to either
- crash the system (than it is similar to destructive testing),
- get access to non-public data, or
- change data.

Performance/load/stress testing



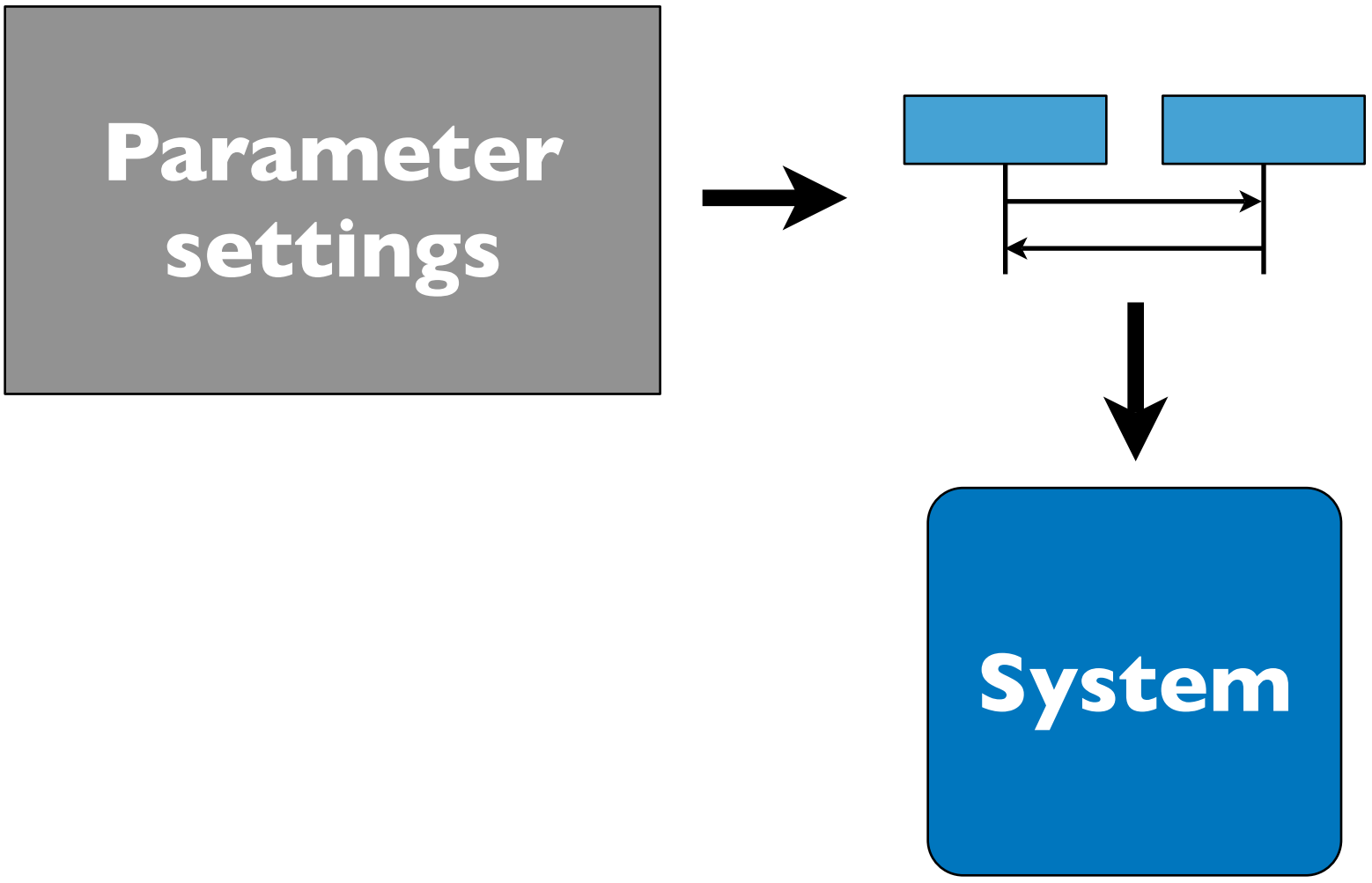
Performance, load, and stress testing all put a certain load of data, interactions, and/or users on the system by its test cases.

In performance testing, usually the normal case is analysed, i.e., how long does a certain task take to execute under normal conditions.

In load testing, we test similar to performance testing but with higher loads, i.e., a large volume of data in the database.

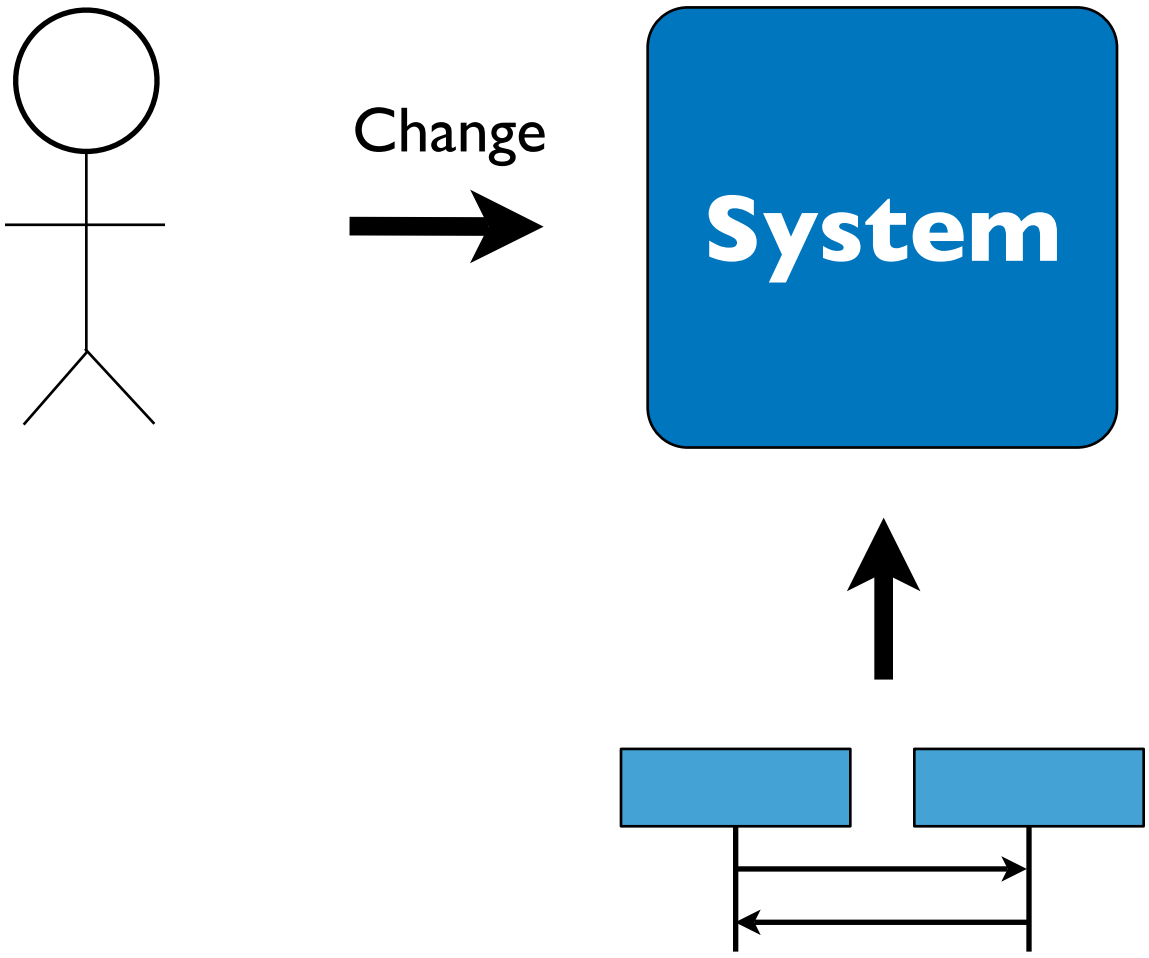
Stress testing has the aim to stretch the boundaries of what is possible or expected load on the system. The system should be placed under stress with too many users interactions or too much data.

Configuration testing



Modern software systems come with a huge amount of parameters that can be set to change the behaviour of the system.
These settings are also called configurations. It is usually practically impossible to test all configurations.
Configuration testing is the method that generates test cases out of possible configurations in a structured way.

Regression testing



Regression testing is a method that is different than the others so far, because it is not a means to generate test cases.
It rather is a method to re-apply test cases after changes. The idea is that once the test suite (i.e., a set of test cases) is developed, it can be run again after a change of the system was done (either bug fix or new feature) to make sure that existing functionality and quality has not been broken by the change.

Group work

Which testing methods can be used for which quality attributes?

10 minutes
On whiteboard

Group work results

Testing and quality attributes

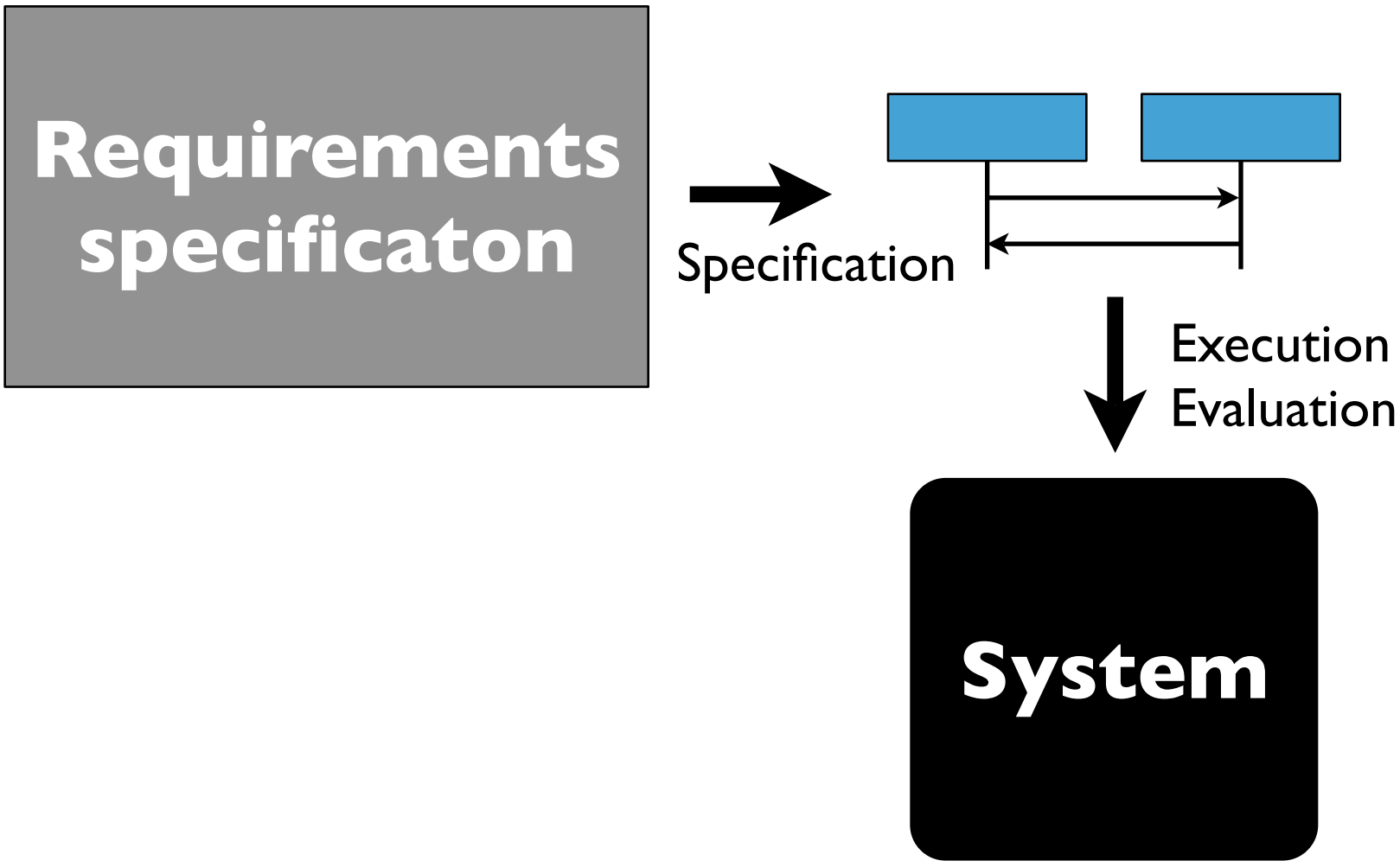
Wagner et al., Quality Models in Practice, 2010

In a recent study, we asked quality practicioners for which quality attributes they use testing.

To the most degree, performance, functional suitability, and reliability are analysed by testing.

Portability and maintainability are less analysed by testing.

Automation



Testing should be automated as far as possible in order to be able to retest often (regression testing).
What can be automated is
1. Test case specification: for example, in random testing, the inputs can be generated with a random generator
2. Test case execution: if the test cases are specified in an executable language, they can be run automatically (e.g., TTCN-3)
3. Test case evaluation: if there is an oracle, the output of the system can be automatically compared with the expected output

Constructive Quality Assurance

Testing