Testing the Executable Digital Twin

Bernhard Sputh

SIEMENS

About me
Dr. Dipl Ing (FH) Bernhard H.C. Sputh

- Dipl. Ing. (FH) Communications Engineering @ FH der Deutschen Telekom Dieburg, Germany.
- Phd in Electronics @ University of Aberdeen, Scotland, UK.
- Joined Siemens in 2019 as Sr Software Engineer Model-based System Testing.
- Background in formal modelling of concurrent and parallel systems.
- Experience with safety critical systems and their qualification / certification.
- Current responsibilities:
  - Specification of new features;
  - Backend / Frontend testing coordinator;
  - Anything related to real-time, or communication.

SIEMENS

Outline

SIEMENS

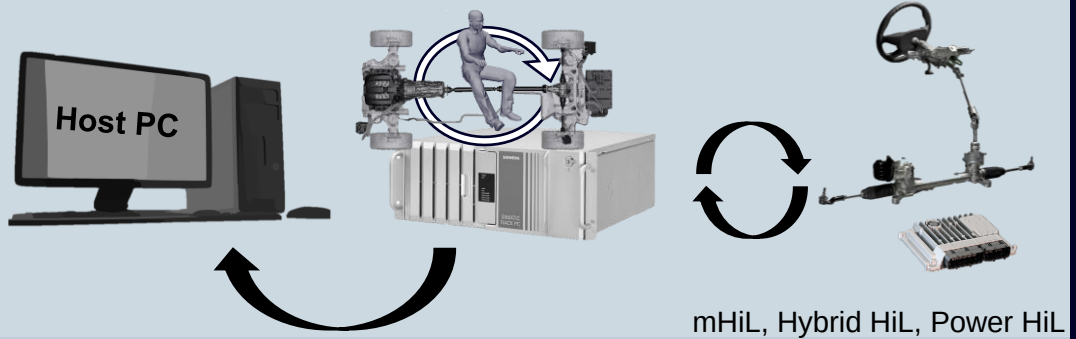
Executable Digital Twin

SIEMENS

Executable Digital Twin

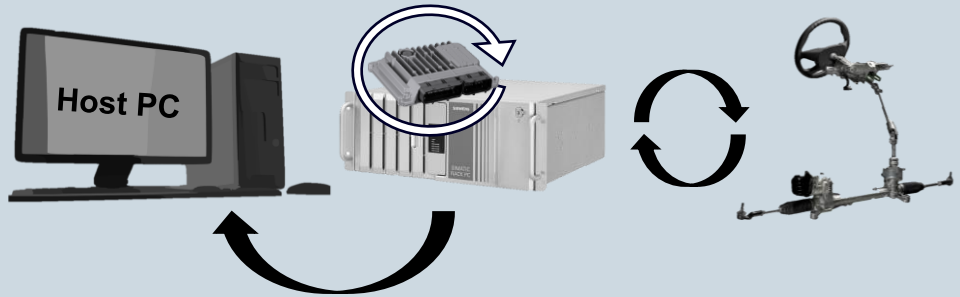
Types of testing enabled by the Executable Digital Twin

SiTL = System-in-the-loop



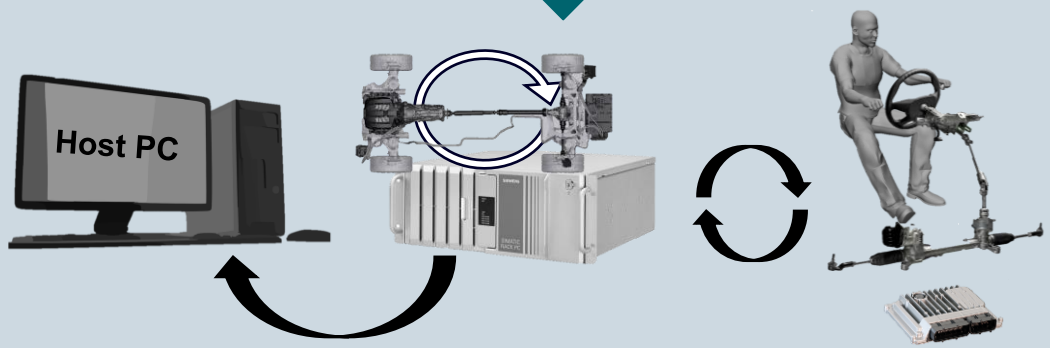
Early system testing in its simulated environment

RCP = Rapid Control Prototyping



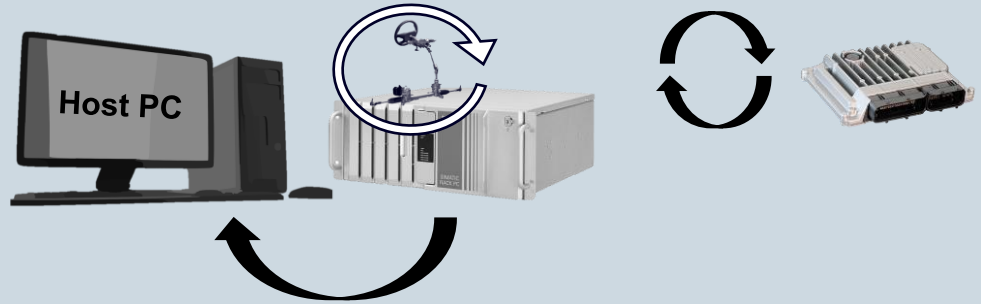
Control algorithm development and testing in its real environment

HiTL = Human-in-the-loop



Human early product assessment in its simulated environment

HiL = Hardware-in-the-loop



Physical ECU software testing & calibration in its simulated environment

SIEMENS

Executable Digital Twin

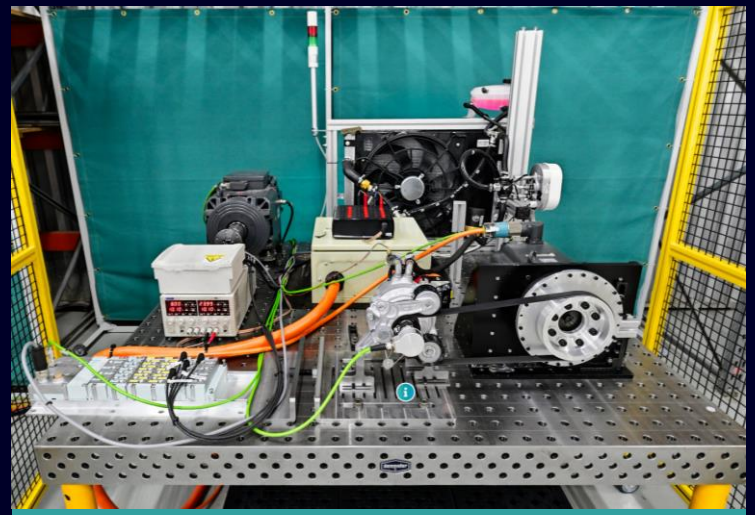
Examples of testing with the Executable Digital Twin.

Vehicle energy management testing
with virtual tire/roads and physical vehicle

Vehicle e-engine testing
with virtual vehicle, virtual road & physical e-engine

Motorcycle performance testing
with virtual chassis, virtual road & physical gearbox

Driver behavior testing
with virtual vehicle, virtual road & physical HID



Electric vehicle component testing
with virtual vehicle, virtual road & physical subsystem

Vehicle powertrain testing
with virtual vehicle, virtual road & physical PWT

SIEMENS

Executable Digital Twin

Example application: driver-in-the-loop simulator for autonomous driving

1. Driver-in-the-loop simulator

- https://blogs.sw.siemens.com/simcenter/chengdu-driver-experience/
- Use-case: driver behavior studies for autonomous driving

2. Simulation models

- Advanced vehicle dynamics real-time models for Internal Combustion Engine, Hybrid Electric Vehicle, Battery Electric Vehicle including realistic steering behavior and powertrain behavior in Simcenter Amesim and Simcenter 3D Motion RT.
- Data-driven sound models for accurate vehicle sound synthesis and active sound in real-time in Simcenter Vehicle Sound Simulator

3. System integration

- Hard-real-time co-simulation of all executable digital twins in Simcenter Testlab RT
- Real-time interfacing of Simcenter Testlab RT to Cruden Panthera controller for platform motion control

SIEMENS

Executable Digital Twin

Example application: Fault injection testing of an in-wheel propulsion system

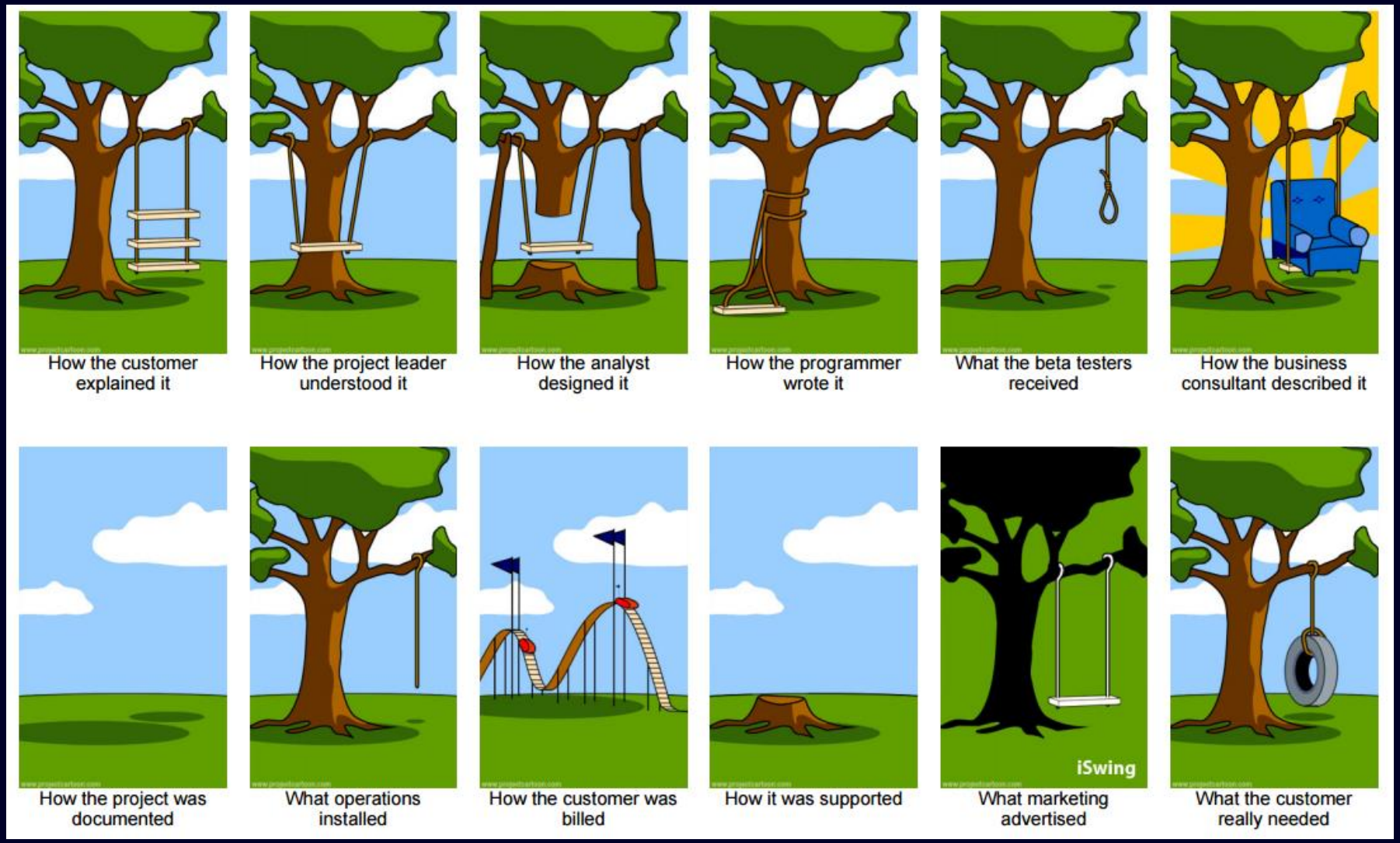
SIEMENS

Systems engineering in a nutshell

SIEMENS

Systems engineering in a nutshell

Why requirements capturing is essential



SIEMENS

Systems engineering in a nutshell
Requirements collection phase

Aims:

- To find out what the customer needs.
- Collect requirements from the customer.
- Understand the problem to be solved.

Typical customer requirement:

- “Build me a fast car.”

Customers are usually not sure about their needs, therefore it is critical to interact with the customer to determine what they really need!

SIEMENS

Systems engineering in a nutshell
Specification derivation phase

Aim:

- To convert the requirements of the customer into one or more *specifications* which that can be *implemented* and *tested*.

Initial input from the customer:

- “Build me a fast car.”

Derived specifications:

- “The car shall accelerate from 0-100 km/h in 3.9s.”
- “The car shall have a top speed of 300km/h”

SIEMENS

Systems engineering in a nutshell

Test case specification phase 1

Aim:

- Specify one or more *test cases* which can determine whether the *specifications* are met.

Specification to test:

- “The car shall accelerate from 0-100 km/h in 3.9s.”

What do we need to specify:

- Test setup.
  - What do we need to prepare for the test?
- Test execution:
  - How will the actual test be performed?
- Test teardown:
  - What do we have to do after the test, to allow the test to be repeated.

SIEMENS

Systems engineering in a nutshell

Test-case specification for: “The car shall accelerate from 0-100 km/h in 3.9s.”

Test setup:

- Track to use
- Driver
- Measurement equipment to use
- Markings to apply to the track
- Which fuel to use

Test execution:

- Accelerate from standstill to the speed of 100 km/h driving in a straight line. Measure how long it takes.

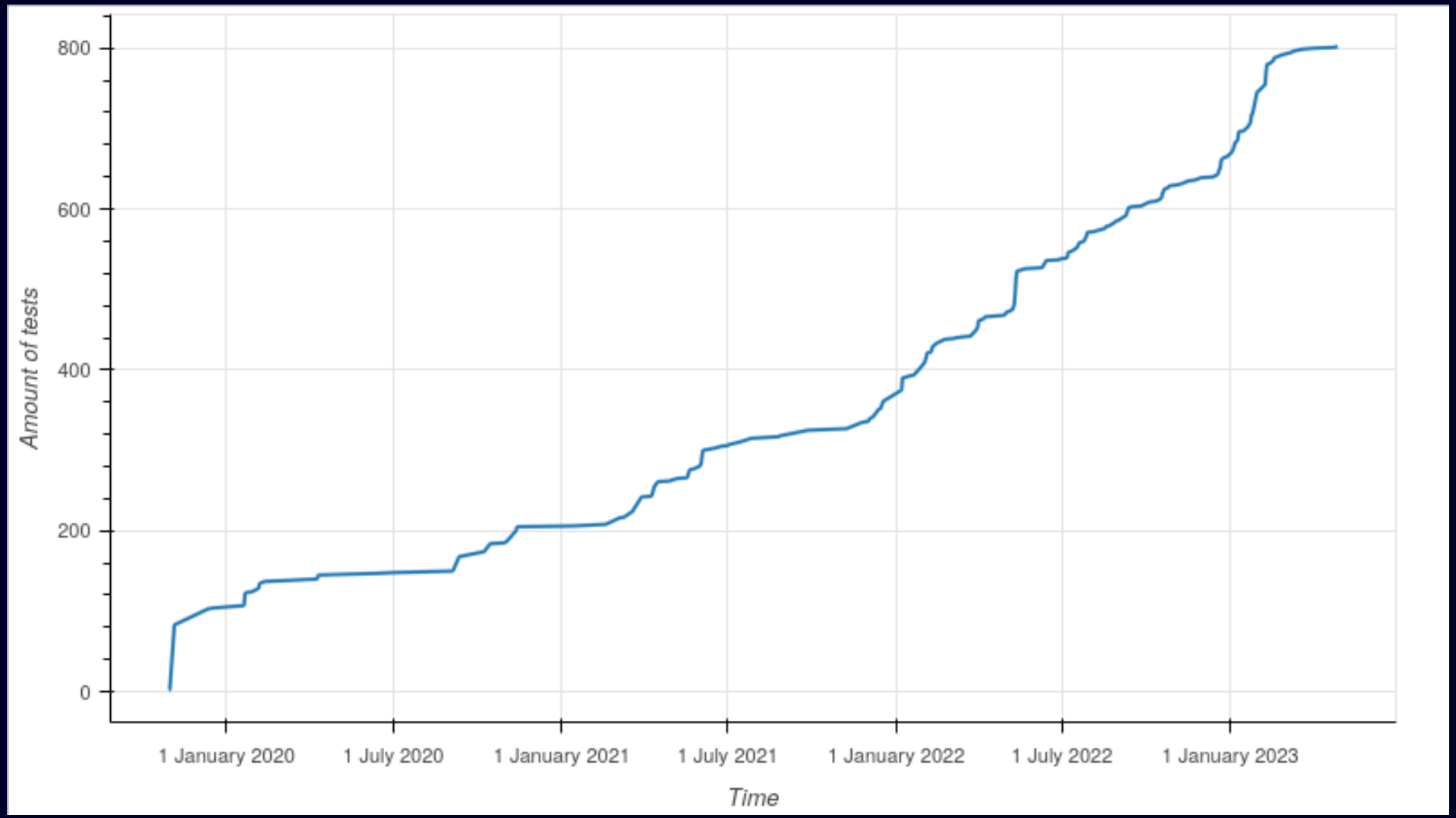
Test teardown:

- Remove all markings applied to the track.
- Prepare a report of the test run.

SIEMENS

Systems engineering in a nutshell

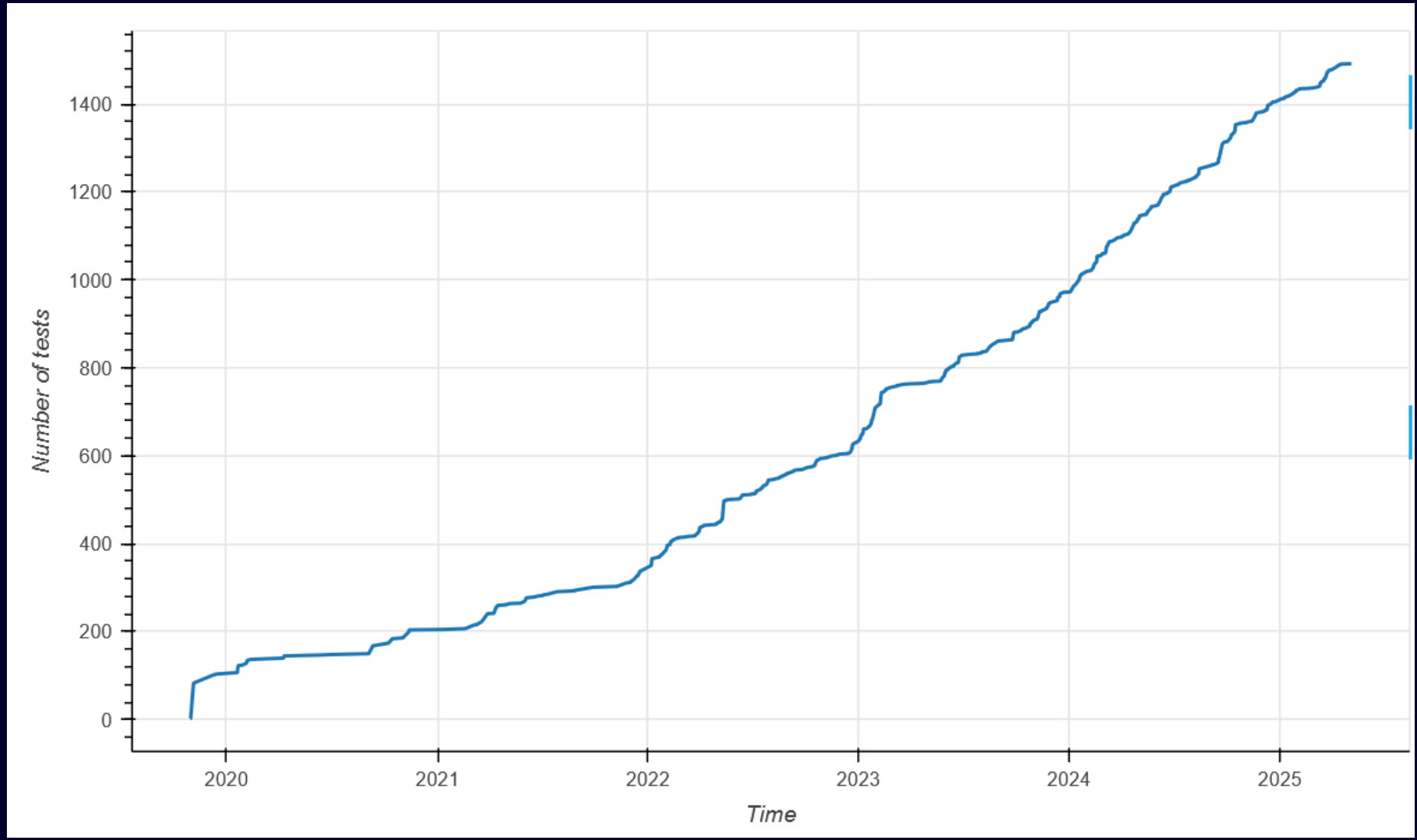
Number of automated test cases over time for the XDT in 2023



SIEMENS

Systems engineering in a nutshell

Number of automated test cases over time for the XDT, today



SIEMENS

If you have no specification you cannot test.

Bernhard Sputh 2023

SIEMENS

Testing software and systems

SIEMENS

Testing software and systems
Unit testing

Purpose:

- To verify the correct implementation of the independently testable parts of the component under test.
- Testing is possible function by function, which allows to test corner cases that are otherwise difficult to trigger.

When:

- Each developer shall run the unit test suite for the project he/she is working on before pushing changes.
- Every time a developer pushes changes to a feature branch the unit-tests get executed.
- Reviewers shall run the unit-tests as part of the review process.
- Scheduled test runs of the current production code to discover regressions in dependencies

Example for unit-tests in our solution:

- XML parser implementation for FMI standard.

Testing software and systems
Integration testing

Purpose:

- Test that the integrated solution works correctly.

When:

- Each developer shall run the integration test suite for the project he/she is working on before pushing changes.
- Every time a developer pushes changes to a project the integration-tests get executed.
- Reviewers shall run the integration-tests as part of the review process.
- Scheduled test runs of the current production code to discover regressions in dependencies

Examples:

- Each web application ships tests for the API it exposes which can only be run if the system is integrated.
- Typically, we integrate the full solution (>100 repositories) based on the latest production code, to ensure that we discover problems introduced by changes elsewhere.

SIEMENS

Testing software and systems
End-to-end testing

Purpose:

- To verify that the complete system functions as we specified.
- To spot issues in the API between the UI and the backend.

When:

- Each developer shall run the end-to-end test suite for the project he/she is working on before pushing changes.
- End to end tests are run in the CI at least daily using the current production code.
- Reviewers shall run the end-to-end tests as part of the review process.

Example for end-to-end tests in our solution:

- Testing whether a dummy hardware device can be correctly configured in the UI.
- Tests are running against production build backend running on a separate PC.
- Testing a full production system, by using the frontend and backend from a production system.

Testing software and systems
Performance testing / Profiling 1

Purpose:

- To determine whether the system meets the specified performance targets.
- To determine the impact of changes on the performance of the system.

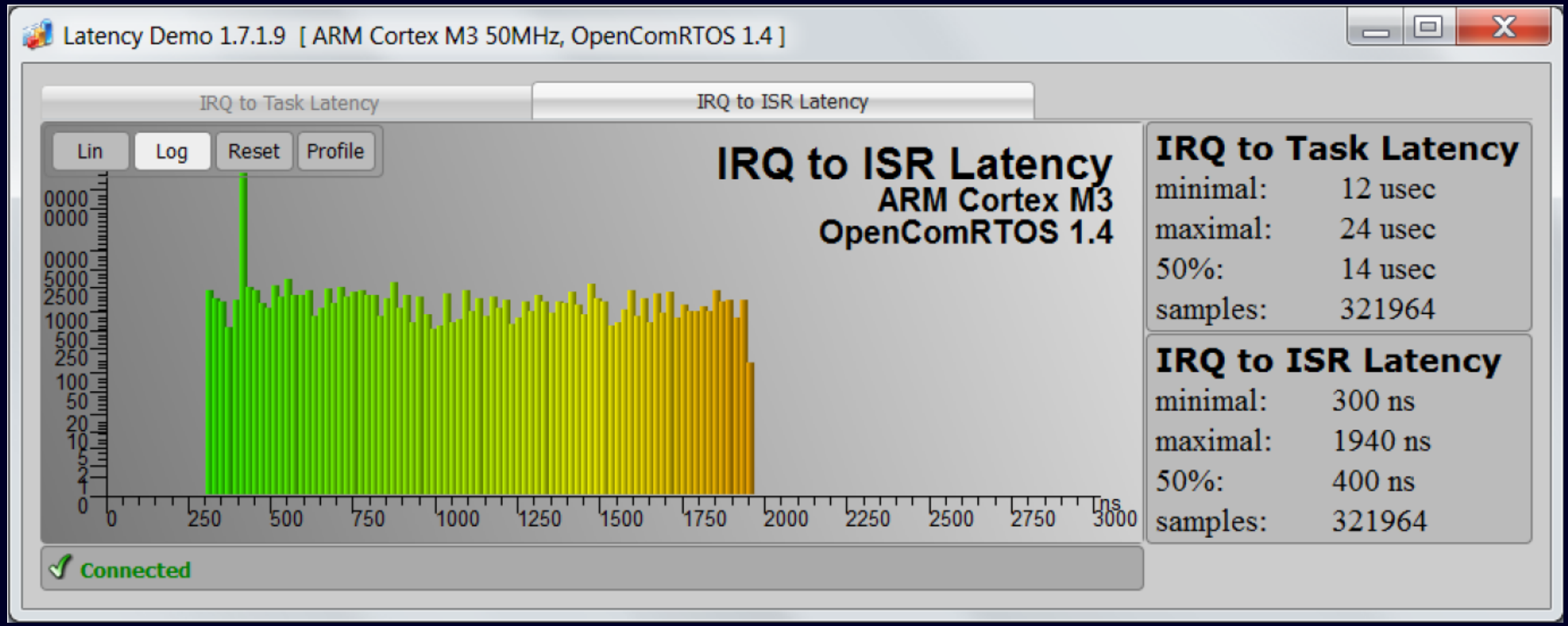
When:

- Whenever one suspects that a change might have an impact on the performance.
- To determine the impact of a proposed change on the performance.
- Characterizing hardware.

Example of performance tests:

- Measure the time it takes to compute a given workload.
- Measuring the jitter of the periodic timer. (Histogram)
- Measuring how long it takes to process an interrupt. (Histogram)

Testing software and systems
Performance testing / Profiling 2



SIEMENS

Testing software and systems
Factory testing

Purpose:

- To detect that the system works as expected before delivering it to a customer.

When:

- Before delivering a system to a customer.

What we do:

- We run all applicable integration and UI tests against the system to be delivered.
- Known customer specific setup might get replicated (for special deliveries). Customer defined acceptance tests might be run.

SIEMENS

Testing software and systems
Customer Integration Testing

Purpose:

- To find any problems related to interfacing into the customer system.

When:

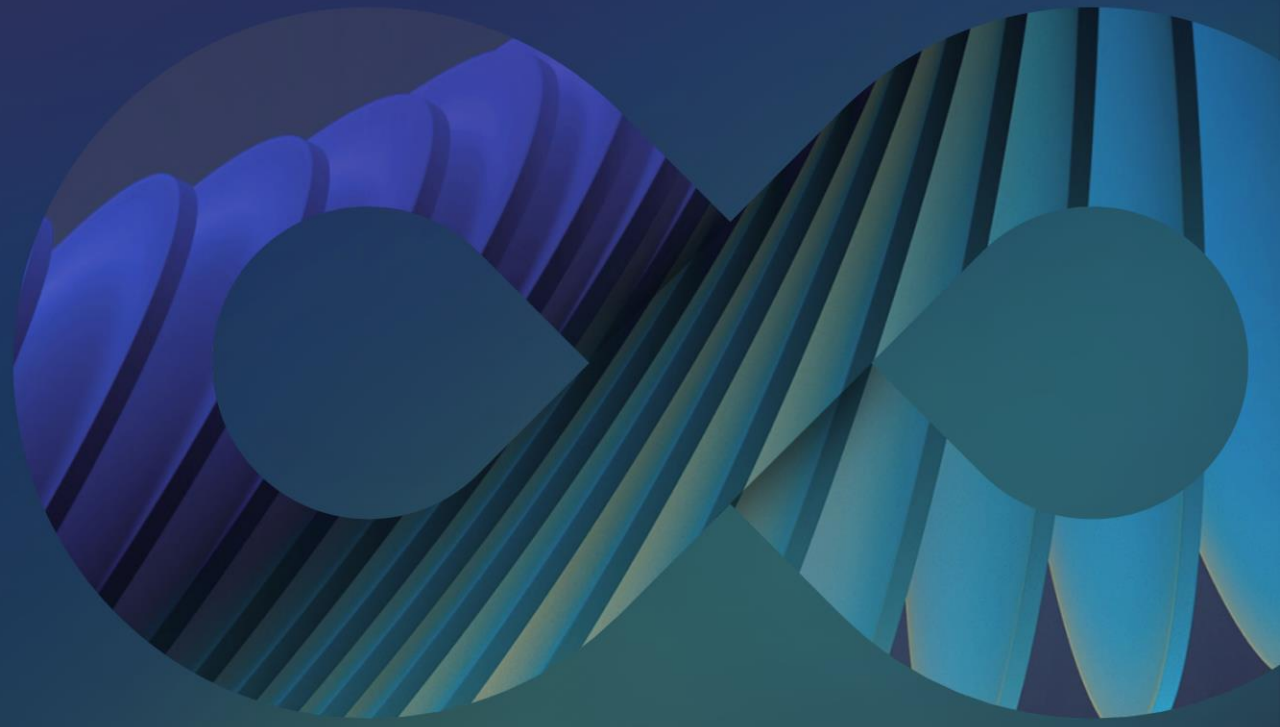
- When the system is onsite at the customer.

What do we do:

- A colleague from the support / delivery organization in the target region delivers the system to the customer and assists with the integration. Together with the customer the colleague runs any necessary test to ensures that the system meets the expectations of the customer.

SIEMENS

Exploratory testing



SIEMENS

Exploratory testing

Purpose:

- To trigger conditions in the system that we did not think about yet.
- To improve our test suites

When:

- Manual activity performed on a monthly basis.

Examples:

REST API fuzzing to spot problems at the interface level (security and safety).

- RESTLER REST-API fuzzer.

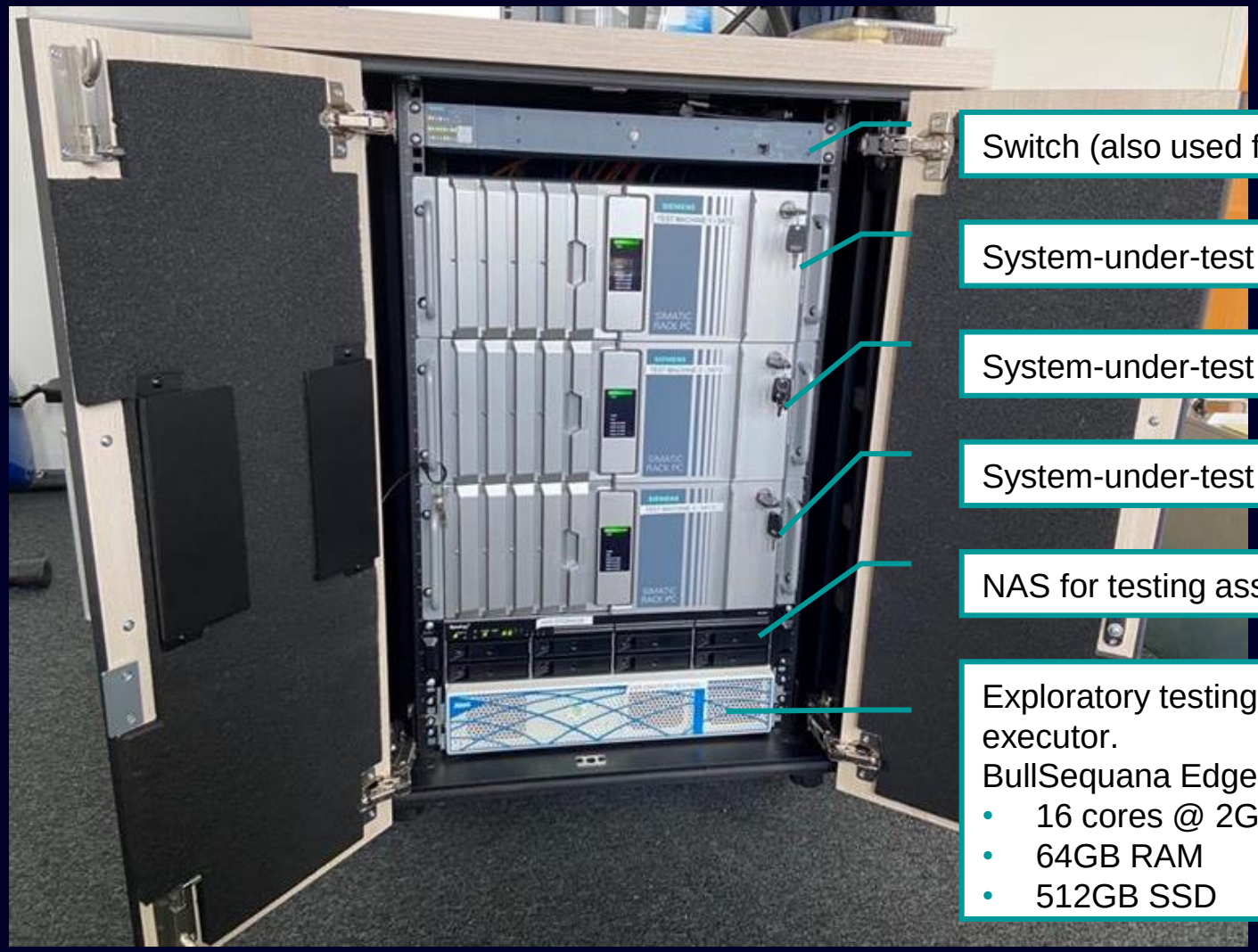
Mutation testing of our test suites, and then improving the test suite.

- Mutmut for Python mutation testing
- Dextool for C++ code mutation testing.

SIEMENS

Exploratory testing

Test farm for exploratory testing and DevOps testing



Switch (also used for DevOps testing)

System-under-test 1

System-under-test 2

System-under-test 3

NAS for testing assets

Exploratory testing orchestrator and executor.
BullSequana Edge

- 16 cores @ 2Ghz
- 64GB RAM
- 512GB SSD

SIEMENS

Exploratory testing
Mutation testing with mutmut

- mutmut – python mutation tester
- Interfaces seamlessly with pytest.
- Allows parallel execution of the tests, to do faster mutation testing on multi-core systems.

```
mutations_by_type = {
    'operator': dict(value=operator_mutation),
    'keyword': dict(value=keyword_mutation),
    'number': dict(value=number_mutation),
    'name': dict(value=name_mutation),
    'string': dict(value=string_mutation),
    'argument': dict(children=argument_mutation),
    'or_test': dict(children=and_or_test_mutation),
    'and_test': dict(children=and_or_test_mutation),
    'lambdef': dict(children=lambda_mutation),
    'expr_stmt': dict(children=expression_mutation),
    'decorator': dict(children=decorator_mutation),
    'annassign': dict(children=expression_mutation),
}
```

Exploratory testing
Properties of project where we applied mutmut

We applied mutmut to one of our repositories that had a >90% code coverage

Code under test properties

- 1259 statements
- 508 branches

Test suite properties

- Test suite 176 tests;
- Test suite execution time (one run):
  - Single core: 66s
  - Parallelized (16 cores): 15s

SIEMENS

Exploratory testing
Baseline results of applying mutmut

```
Legend for output:
🎉 Killed mutants.   The goal is for everything to end up in this bucket.
⏰ Timeout.          Test suite took 10 times as long as the baseline so were killed.
🤔 Suspicious.       Tests took a long time, but not long enough to be fatal.
😕 Survived.         This means your tests need to be expanded.
🔇 Skipped.          Skipped.

1. Using cached time for baseline tests, to run baseline again delete the cache file

2. Checking mutants
.: 2413/2413  🎉 1689  ⏰ 0  🤔 0  😕 724  🔇 0
```

724 surviving mutants ☹
2413 total mutants

SIEMENS

Exploratory testing
What we learned from using mutmut

Test suite improvements

- 37 tests were modified in the process.
- The bulk of the changes were assertions for log messages generated by the code.
- Generated status messages were not always checked.
- When a value was not expected, the absence of it was not always checked.

What we learned

- We discovered that tests for log messages and status messages are missing.
- Need to not only check for the presence of expected values, but also for the expected absence of values.
- Our tests suite was in a good state regarding the functionality of the component.

Exploratory testing
Improved test suite results

```
Legend for output:
🎉 Killed mutants.   The goal is for everything to end up in this bucket.
⏰ Timeout.          Test suite took 10 times as long as the baseline so were killed.
🤔 Suspicious.       Tests took a long time, but not long enough to be fatal.
😕 Survived.         This means your tests need to be expanded.
🔇 Skipped.          Skipped.

1. Using cached time for baseline tests, to run baseline again delete the cache file

2. Checking mutants
⠸ 2413/2413  🎉 1808  ⏰ 0  🤔 0  😕 605  🔇 0
```

119 additional mutants killed by extending the test suite (additional checks implemented)

SIEMENS

Typical software testing mistakes

SIEMENS

Typical software testing mistakes

1. Only testing for the good case.
2. Not testing corner cases.
3. Not testing at the boundaries.
4. Not looking at the code coverage of the test suite.
5. Not testing in an environment that is close to production.
6. Not writing automated test cases as part of the development process.
7. Not looking at the code (closed box testing approach).
8. Not specifying all preconditions that must be met for a test to succeed.
9. Not cleaning up after a test (successful or failing) and thus influencing the next test.
10. Not running the test suites because nothing has changed.
11. Doing big bang testing at the end.

Only tests that exist can fail!

Bernhard Sputh 2023

SIEMENS

Only running tests can fail!

Bernhard Sputh 2023

SIEMENS

Conclusions / Summary

SIEMENS

Summary

- Introduction to the executable digital twin
- Systems engineering
- Testing software and systems
- Exploratory testing
- Typical testing mistakes

SIEMENS

Thank you

For more information

Bernhard Sputh
Sr Software Engineer Model-based System Testing

Website
https://www.plm.automation.siemens.com/global/en/products/simulation-test/model-based-system-testing.html

Youtube Channel
https://www.youtube.com/c/SiemensSoftware/ (Simcenter – Testing Solutions)

SIEMENS