

Level Up Your Embedded Testing Game

FRETish, Robot, and Twister: A Dream Team

Christian Schlotter, Carl Zeiss Meditec AG

Stefan Kraus, UL Solutions SIS

Tobias Kästner, inovex GmbH

Open Source Summit Europe, Vienna

September 17, 2024

Medical is all about Trust

Supporting patients when they're most vulnerable



COPYRIGHT ©1999
TWENTIETH CENTURY FOX FILM CORPORATION
ALL RIGHTS RESERVED



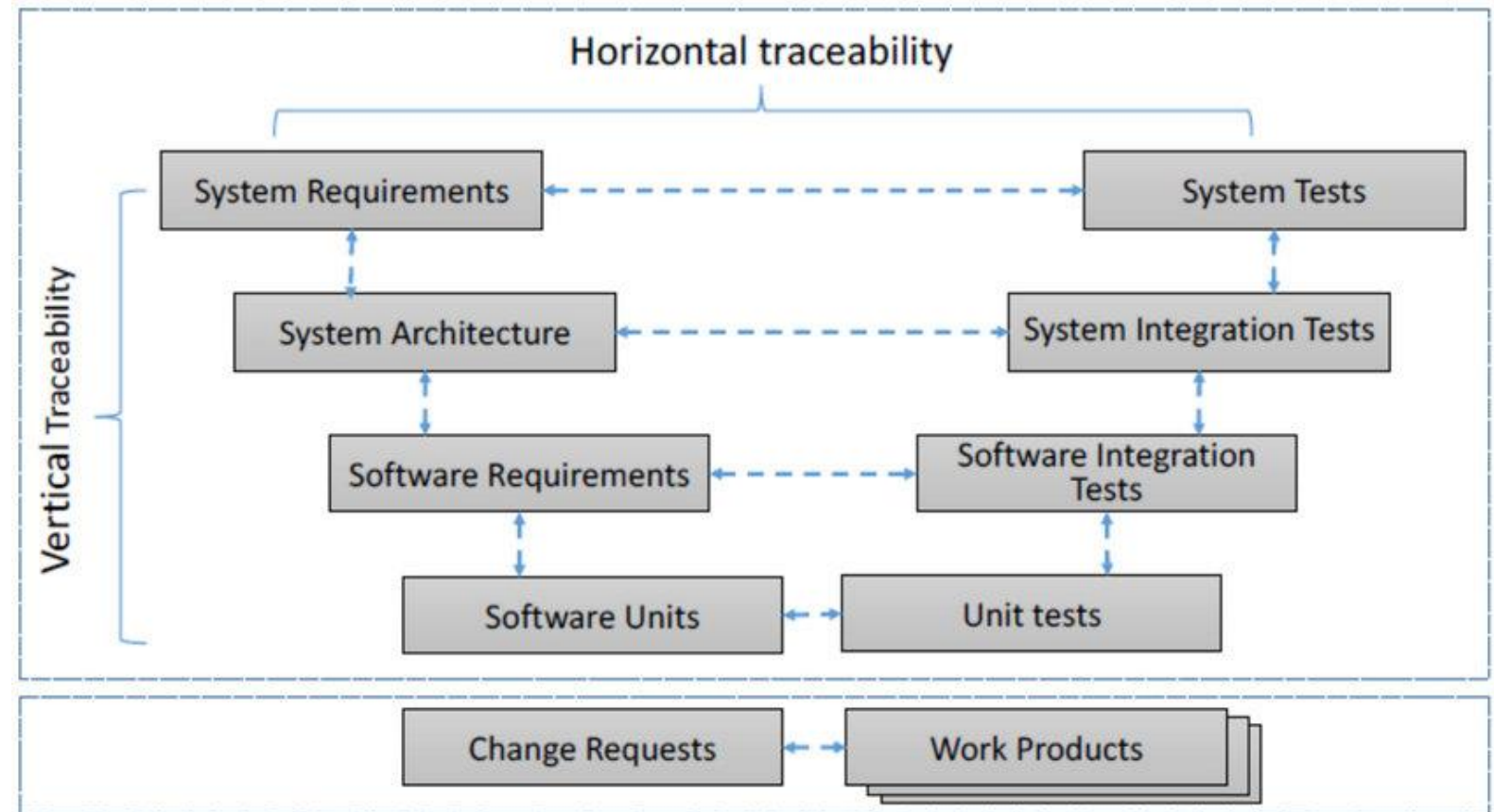
It's not only what we care about

Regulators want us to take care of some things, too

- Requirements Management ✓
- Traceability of
 - Requirements ✓
 - Tests ✓

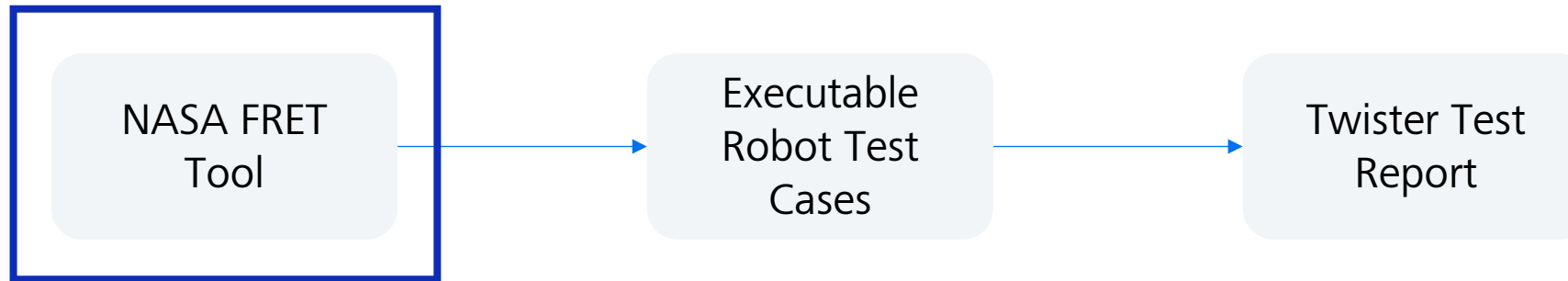
BUT:

- How to check requirements for consistency?
- How to derive tests from requirements?
- How to update tests for changed requirements?



V-model

Our plan for today

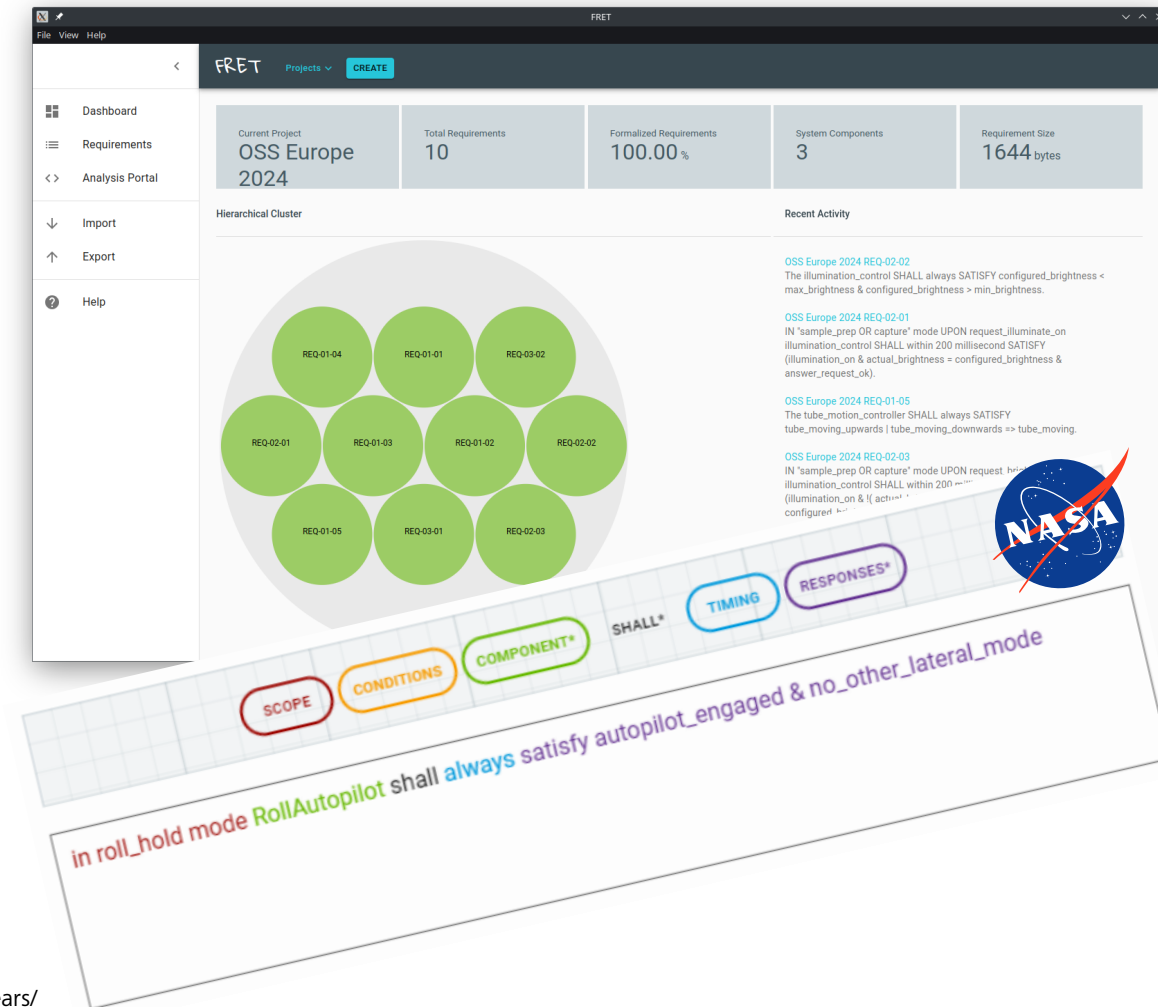


Requirements Engineering

"It's not exactly rocket science, is it?"⁴⁾



- Many approaches for writing "better" requirements described in the literature
 - Easy Approach for Requirements Syntax (E.A.R.S) by Alistair Mavin¹⁾
 - INCOSE Guide to Writing Requirements²⁾
 - **Functional Requirements Elicitation Tool (FRET)**³⁾
- FRET: New semi-formal approach to requirements pioneered by Anastasia Mavridou et al. @ Robust Software Engineering Group at NASA
 - FRETish Requirements are written in controlled natural language
- Requirements are machine-parsable and can be transformed to (temporal) logic formulas
 - Automated consistency & realizability checking
 - Simulation, ...



(1) <https://alistairmavin.com/ears/>

(2) https://www.incose.org/docs/default-source/working-groups/requirements-wg/rwg_products/incose_rwg_gtwr_summary_sheet_2022.pdf

(3) <https://github.com/NASA-SW-VnV/fret>

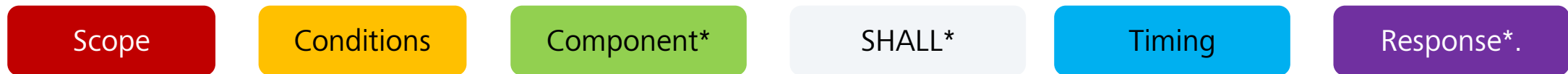
(4) <https://www.youtube.com/watch?v=THNPmhBI-8I>

FRETish Requirements

A very basic "101" in 5 mins or less



General structure of a FRETish Requirement:



* = mandatory



What part of the system has to fulfill this requirement?

Examples

- The "Flight Controller"
- The "Lighting Subsystem"

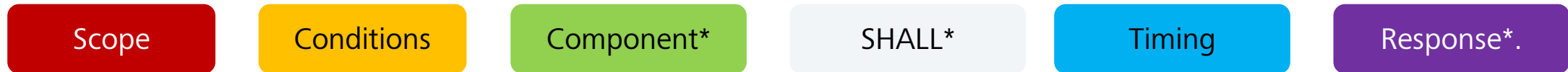
Learn more at <https://github.com/NASA-SW-VnV/fret> and
in the excellent tutorial <https://ntrs.nasa.gov/api/citations/20220007610/downloads/NFM22Tutorialv5.pdf>

FRETish Requirements

A very basic "101" in 5 mins or less



General structure of a FRETish Requirement:



* = mandatory

What is the system supposed to do?

Examples

- Satisfy "maintain speed & maintain altitude"
- Satisfy Illumination_ON

Boolean & Arithmetic expressions:

- `!`, `&`, `|`, `=>`, ...
- `=`, `!=`, `<`, `>`, `+`, `*`, `-`, ...

Learn more at <https://github.com/NASA-SW-VnV/fret> and

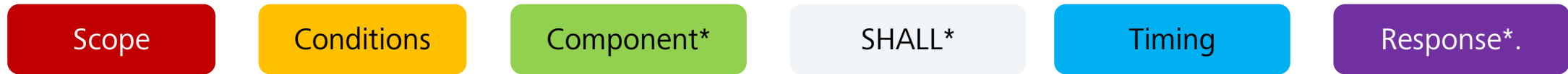
in the excellent tutorial <https://ntrs.nasa.gov/api/citations/20220007610/downloads/NFM22Tutorialv5.pdf>

FRETish Requirements

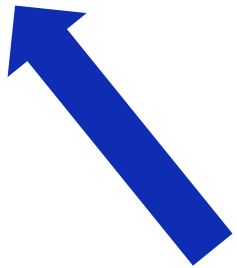
A very basic "101" in 5 mins or less



General structure of a FRETish Requirement:



* = mandatory



During what portion of the execution is the requirement supposed to hold?

Examples:

- "During landing_operation"
- "In factory mode"

Keywords:

- In, before, after, notin, onlyin, ...

Learn more at <https://github.com/NASA-SW-VnV/fret> and

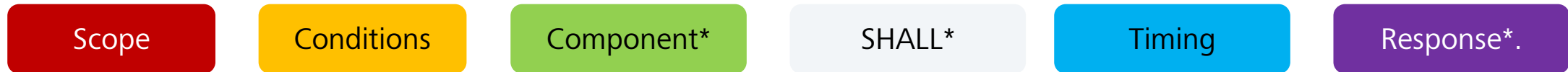
in the excellent tutorial <https://ntrs.nasa.gov/api/citations/20220007610/downloads/NFM22Tutorialv5.pdf>

FRETish Requirements

A very basic "101" in 5 mins or less



General structure of a FRETish Requirement:



* = mandatory

What condition triggers the response?

Examples:

- "Upon auto_pilot_enable"
- "Upon request_dim_lighting"

Keywords:

- Upon, if, where, when, unless, ...

Boolean & arithmetic expressions:

- !, & , | , => , = , != , < , > , + , * , - , ...

Learn more at <https://github.com/NASA-SW-VnV/fret> and

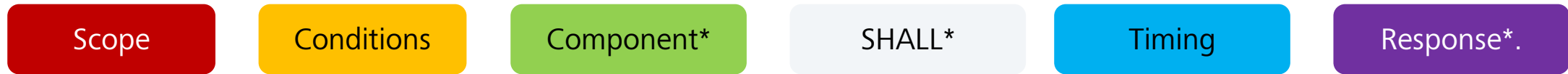
in the excellent tutorial <https://ntrs.nasa.gov/api/citations/20220007610/downloads/NFM22Tutorialv5.pdf>

FRETish Requirements

A very basic "101" in 5 mins or less



General structure of a FRETish Requirement:



* = mandatory



When is the response due relative to scope and condition?

Examples:

- "always"
- "within 1 second"
- "after 200 ticks"

Keywords:

- Always, never, eventually, until, within, before, after

Learn more at <https://github.com/NASA-SW-VnV/fret> and

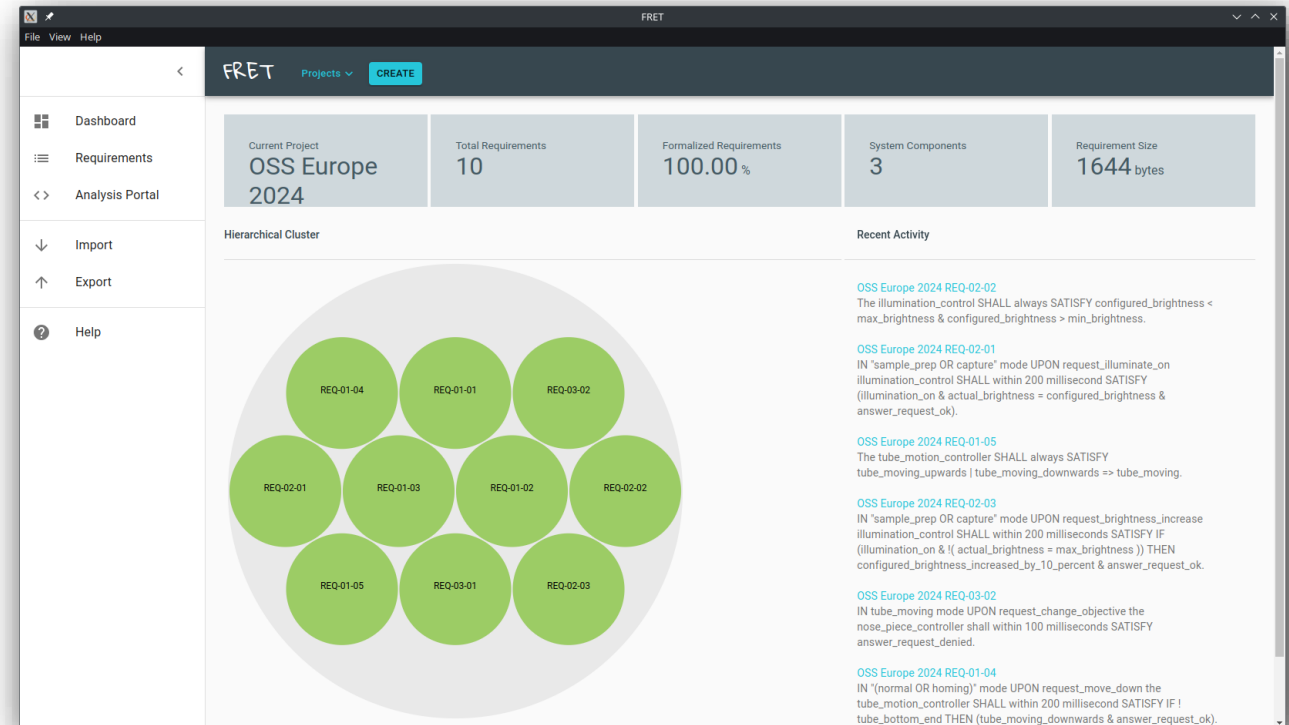
in the excellent tutorial <https://ntrs.nasa.gov/api/citations/20220007610/downloads/NFM22Tutorialv5.pdf>

Tools for FRETish Requirements

The templates and the editor



- FRET tool can be downloaded from <https://github.com/NASA-SW-VnV/fret> (Win/Mac/Linux)
 - Need to build from source
- Simple Project Management
 - JSON-based Import & Export functionality

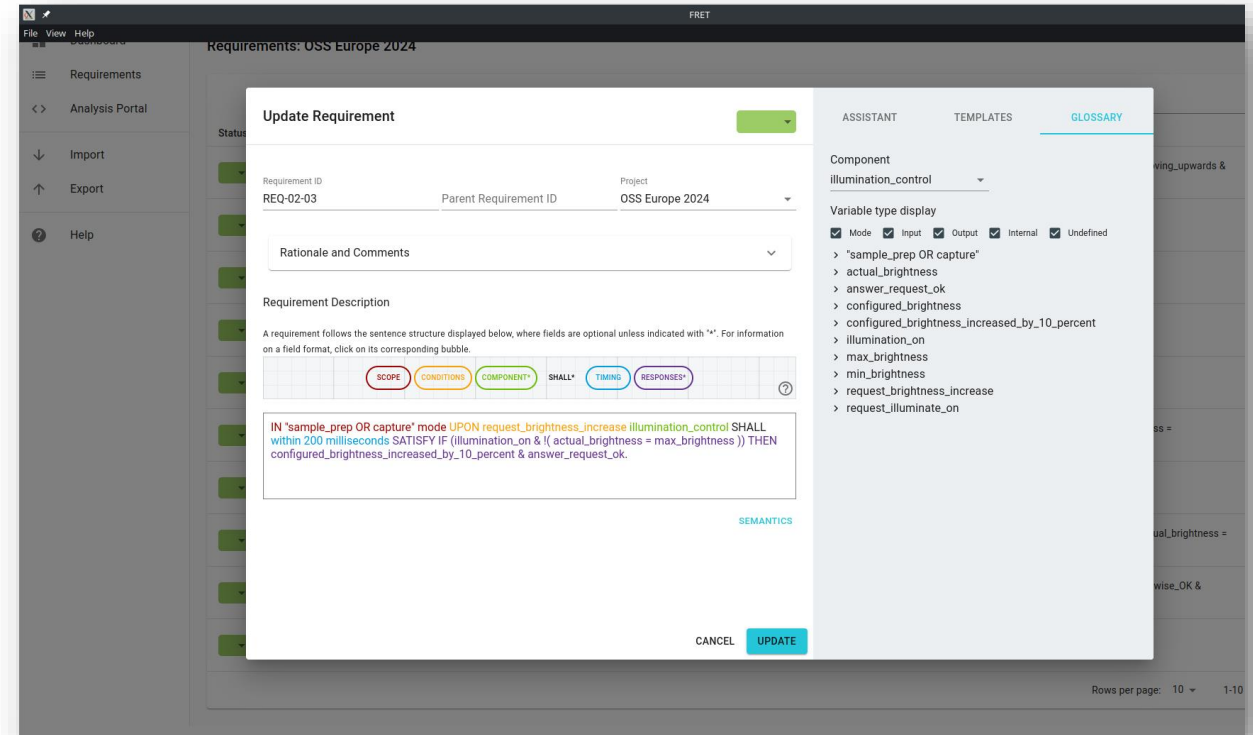


Tools for FRETish Requirements

The templates and the editor



- FRET tool can be downloaded from <https://github.com/NASA-SW-VnV/fret> (Win/Mac/Linux)
 - Need to build from source
- Simple Project Management
 - JSON-based Import & Export functionality
- Syntax Highlighting
- Automatic Glossary
 - Useful to keep terminology consistent across RQTs
- Also captures meta-data
 - ID, Comments, ...
 - Parent/Child Relationships

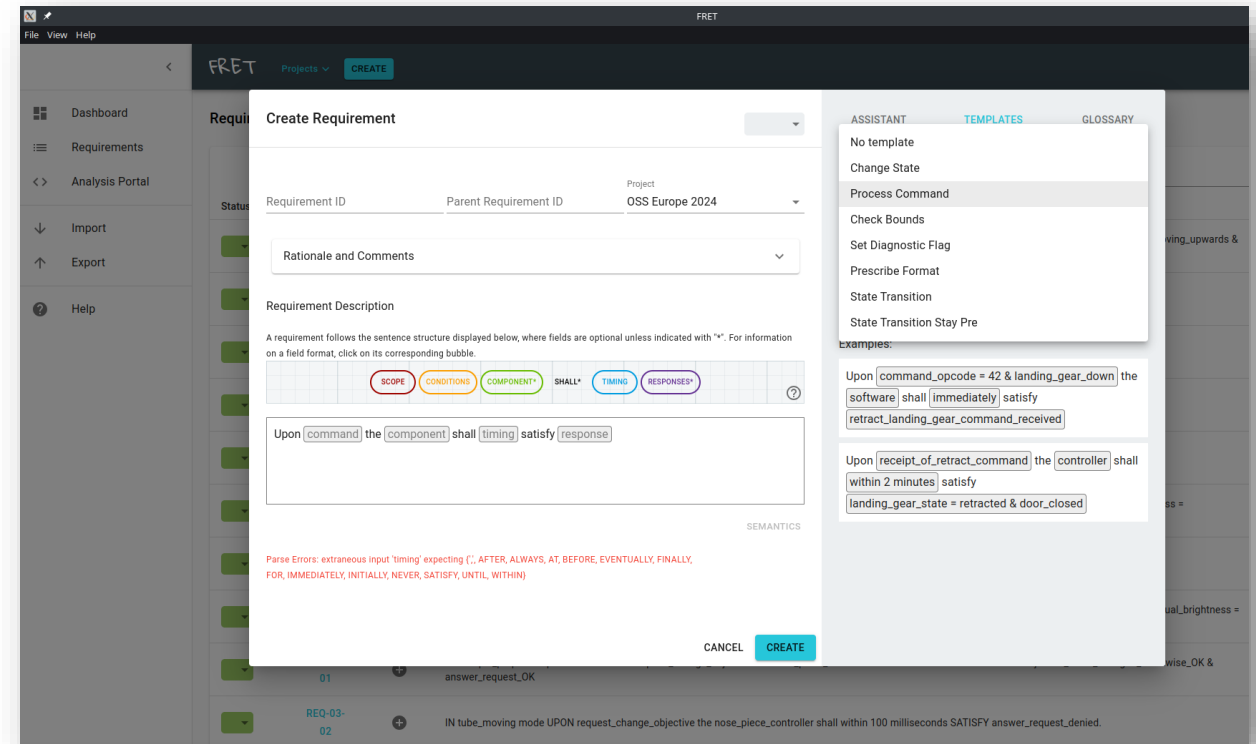


Tools for FRETish Requirements

The templates and the editor



- FRET tool can be downloaded from <https://github.com/NASA-SW-VnV/fret> (Win/Mac/Linux)
 - Need to build from source
- Simple Project Management
 - JSON-based Import & Export functionality
- Syntax Highlighting
- Automatic Glossary
 - Useful to keep terminology consistent across RQTs
- Also captures meta-data
 - ID, Comments, ...
 - Parent/Child Relationships
- Pre-defined templates for common scenarios
 - Very convenient for FRETish starters
- Many more features related to model checking and extracting formal expressions from the requirements
 - Not yet relevant for us



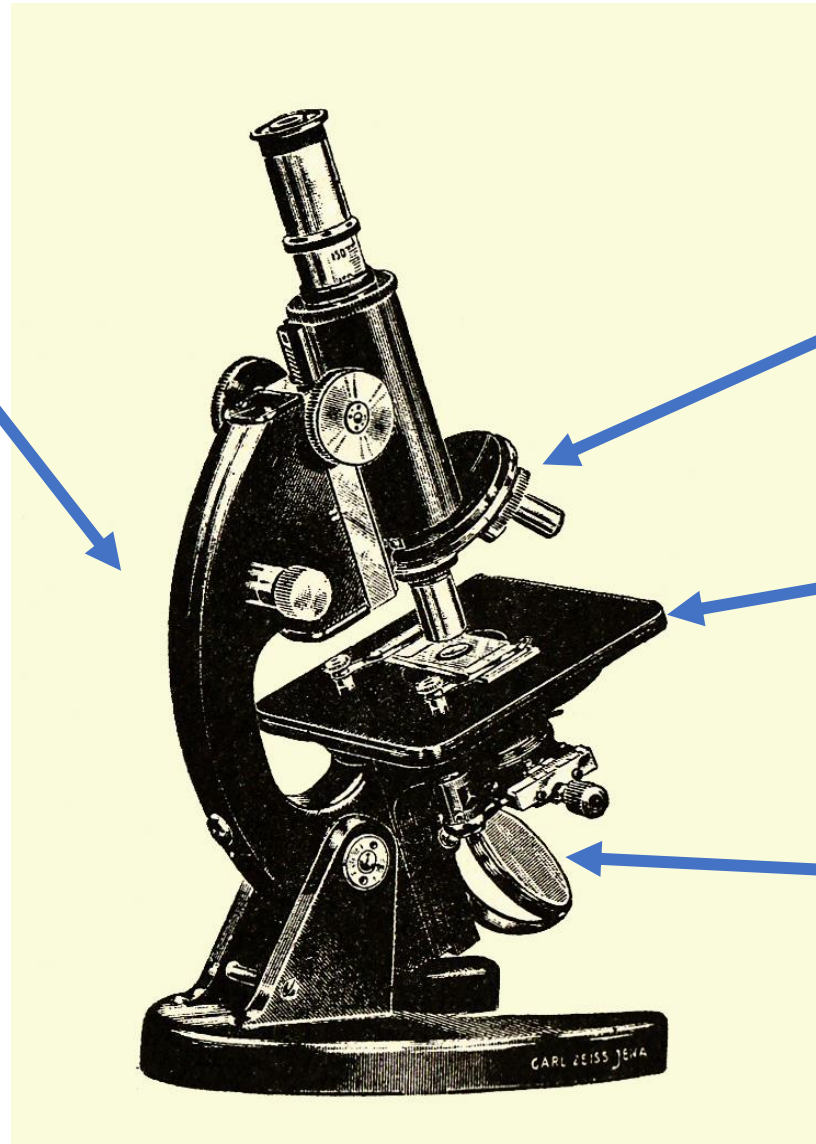
The digitally controlled microscope

An example is worth a thousand theories

Tube w/
Tube Motion Controller

Even microscopes need digital controls these days ;-)

- Precise focus control
- Precise illumination control
- Automated positioning of sample
- Automated change of objective



Nose Piece w/
Nose Piece Controller

Stage w/
Stage Controller

Illuminator w/
Illumination Controller

https://github.com/ZEISS/fretish_robot/tree/main/examples/digital_microscope

The digitally controlled microscope

FRETish Requirements for non-rocket things



Scope

Conditions

Component*

SHALL*

Timing

Response*.

IN "(sample_prep|capture)" mode UPON request_illuminate_on the illumination_controller **SHALL** within 200 milliseconds
SATISFY (illumination_on & actual_brightness = configured_brightness & answer_request_ok).

IN "(sample_prep|capture)" mode UPON request_brightness_increase the illumination_controller **SHALL** within 200 milliseconds
SATISFY IF (illumination_on & !(actual_brightness = max_brightness))
THEN configured_brightness_increased_by_10_percent & answer_request_ok.

The illumination_controller **SHALL** always
SATISFY $\text{configured_brightness} \leq \text{max_brightness}$ & $\text{configured_brightness} \geq \text{min_brightness}$.

The digitally controlled microscope

FRETish Requirements for non-rocket things



Scope

Conditions

Component*

SHALL*

Timing

Response*.

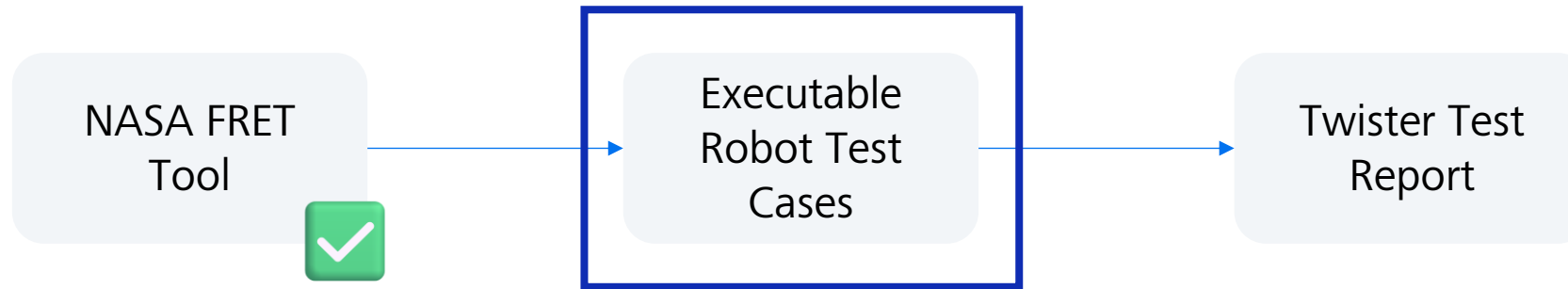
IN "(sample_prep|capture)" mode UPON request_change_objective the nose_piece_controller SHALL within 1 second SATISFY objective_lens_changed_clockwise_ok & answer_request_ok.

IN tube_moving mode UPON request_change_objective the nose_piece_controller SHALL within 100 milliseconds SATISFY answer_request_denied.

IN "(normal|homing)" mode UPON request_move_up the tube_motion_controller SHALL within 200 milliseconds SATISFY IF !tube_upper_end THEN (tube_moving_upwards & answer_request_ok).

Upon request_move_up the tube_motion_controller SHALL always SATISFY IF tube_at_upper_end THEN tube_position_hold.

Our plan for today

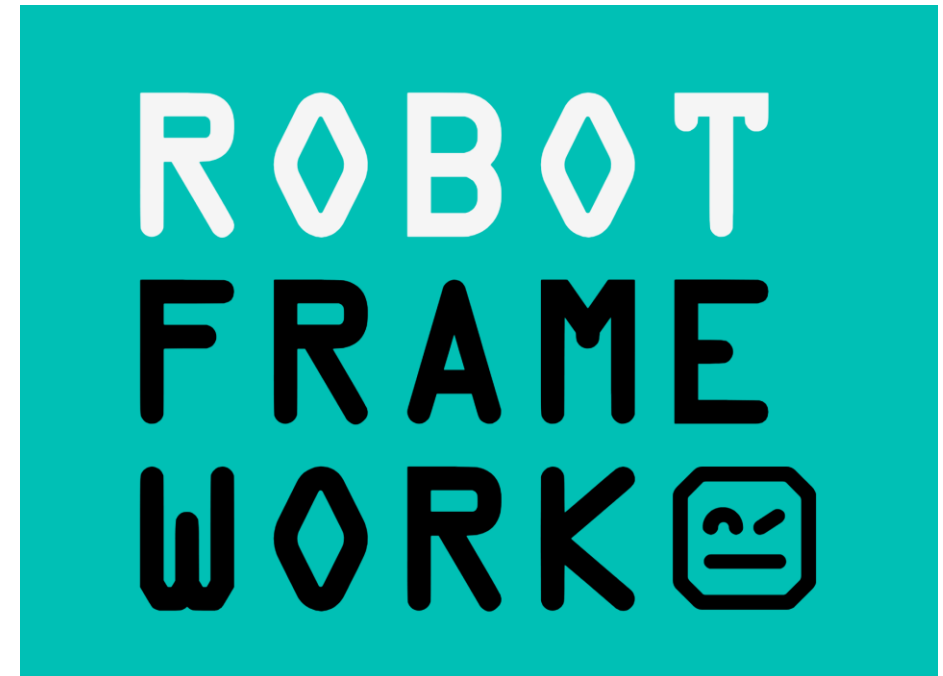


Robot Framework

Human-readable, yet machine-executable



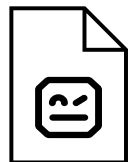
- Open-Source Automation Framework
 - Very mature, development started in 2005
 - Keyword-driven
 - Supports several testing methodologies like Behavior Driven Development (BDD)
 - Also used for Robotic Process Automation (RPA)
 - Very popular for end-to-end testing in web technologies
- Funded by non-profit Robot Framework Foundation
- Supported by Zephyr's Test Runner Twister
- Allows to write human-readable test specifications that can be executed automatically



<https://robotframework.org/>

Robot Framework

Let's look at a simple example



TestSuite.robot

```
*** Test Cases ***
```

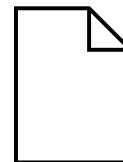
```
Login with Password
```

```
    Connect to Server
```

```
    Login User          ironman      1234567890
```

```
    Verify Valid Login  Tony Stark
```

```
    Close Server Connection
```



keywords.resource

```
*** Keywords ***
```

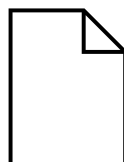
```
Login User
```

```
[Arguments]    ${login}    ${password}
```

```
Set Login Name    ${login}
```

```
Set Password      ${password}
```

```
Execute Login
```

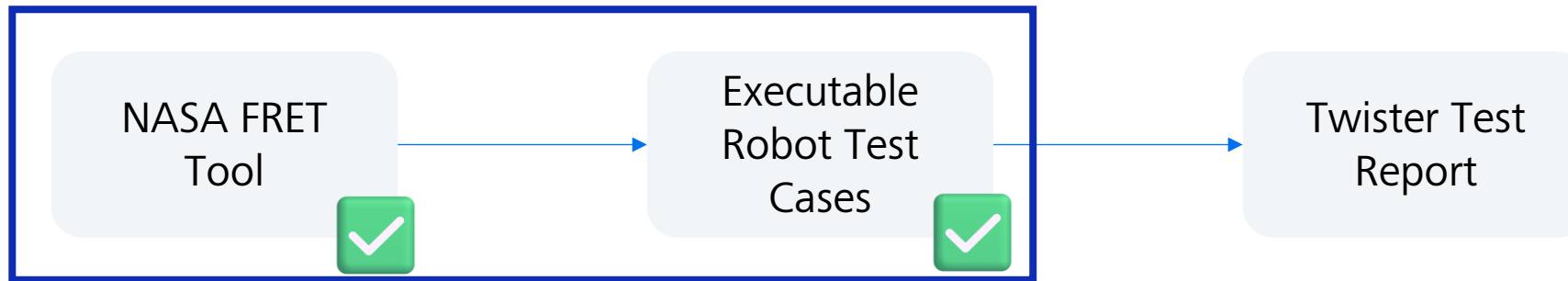


CustomLibrary.py

```
def set_login_name(self, login):  
    '''Sets the users login name and stores it for authentication.'''  
    self.login = login  
    info(f'User login set to: {login}')
```

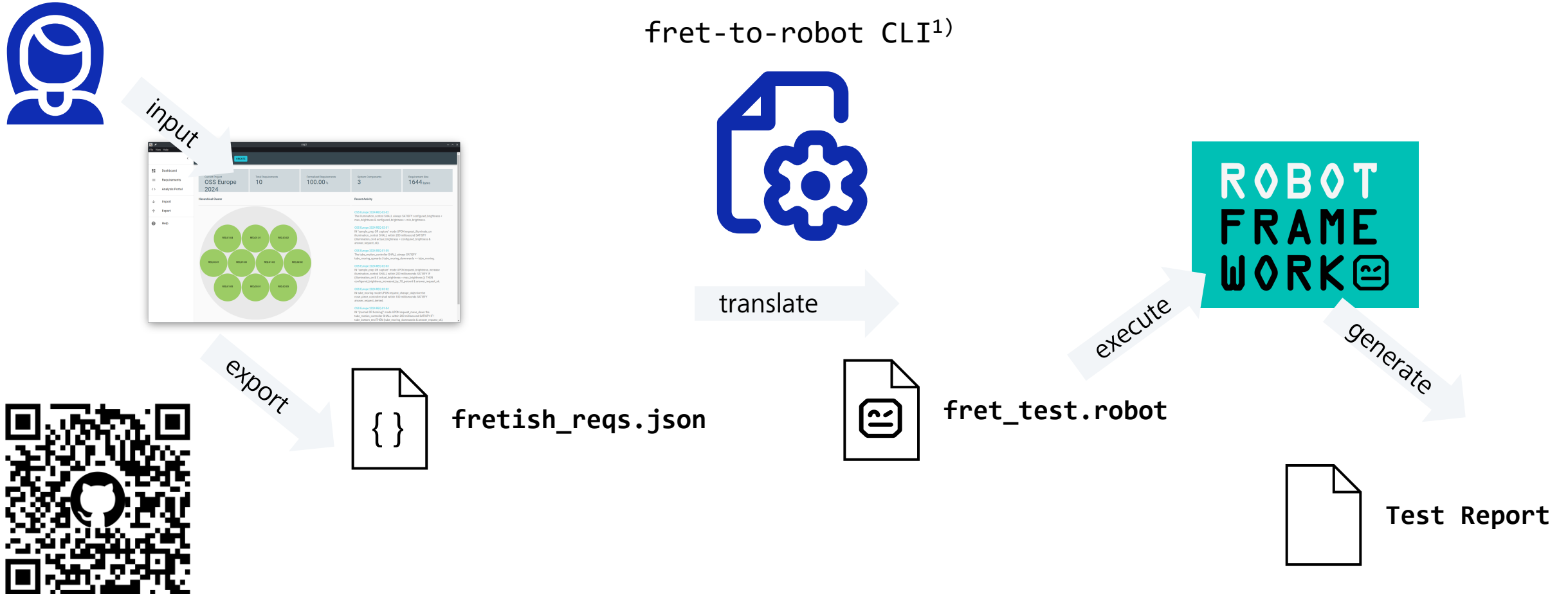
<https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#creating-keywords>

Our plan for today



From FRET to Robot

On the shoulders of giants



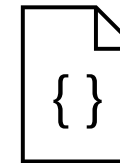
(1) https://github.com/ZEISS/fretish_robot

Generated Robot testcase

Shown for simple example requirement



```
{  
  "scope_mode": "\"(sample_prep|capture)\",",  
  "regular_condition_unexp_pt": "request_illuminate_on",  
  "component_name": "illumination_control",  
  "timingTextRange": [ 86, 107 ],  
  "post_condition_unexp_ft": "((illumination_on & (actual_brightness = configured_brightness)) & answer_request_ok)",  
}
```



fretish_reqs.json

```
$>fret-to-robot fretish_reqs.json --out fret_test.robot
```



*** Test Cases ***

TEST_REQ-02-01-1

```
[Tags]      REQID=REQ-02-01      SCOPE=sample_prep      TRIGGER=request_illuminate_on
```

```
In sample_prep mode
```

```
Upon      request_illuminate_on
```

```
Within    200 millisecond      Satisfy      (($illumination_on and
```

```
($actual_brightness == $configured_brightness)) and $answer_request_ok)
```



fret_test.robot

Keyword implementation

Teaching FRETish to a robot



Upon request_illuminate_on



FRETish keywords

- Keywords in FRETish syntax to express FRET semantics
- Examples: Upon, Within X (milli)seconds
- Implementation:
 - Done in library `fretish_robot.FRETlib`
 - Reduction to built-in keywords

FRETLib.py

```
def upon(self, event_name):  
    """Runs the `event_name` keyword.  
    Like `Run Keyword` but for FRET read"""  
    self.built_in.run_keyword(event_name)
```

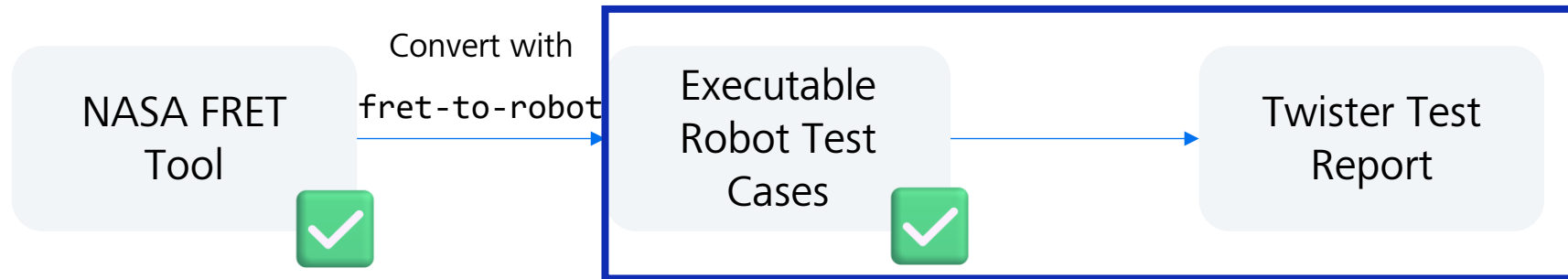
Functional keywords

- Keywords that implement functional behavior
- Example: `request_illuminate_on`
- Implementation:
 - Specific to application logic → additional custom library

CustomLib.py

```
def request_illuminate_on(self):  
    """Turns illumination on.  
    Done by sending command via shell fixture"""  
    self.shell.exec_command("illuminate set on")
```

Our plan for today



Zephyr, Twister and Test Harnesses

All batteries included, yes, or no?

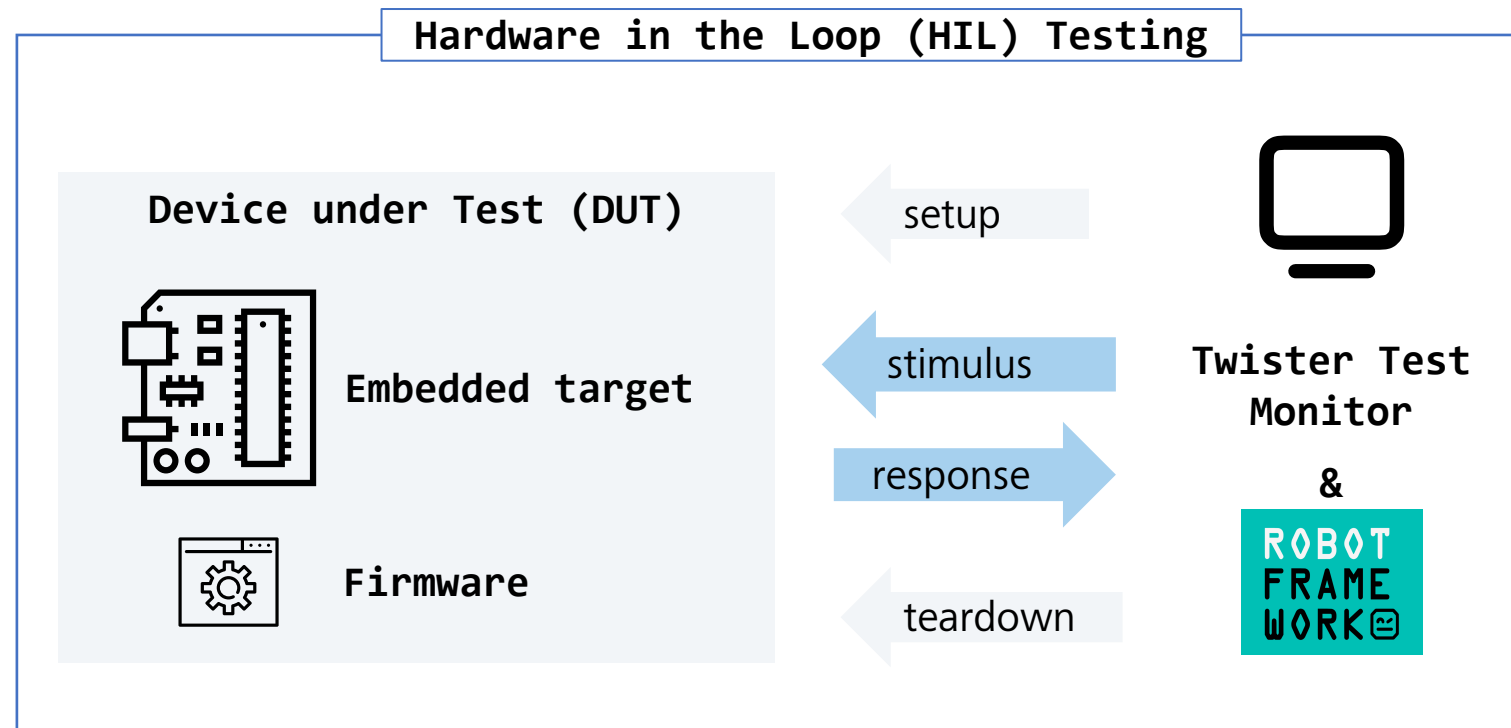


- To this point FRETish & Robot independent of application domain
- Our domain is embedded computing, and our Firmware runs on Zephyr :-)
- Better still, Zephyr has built-in support for HIL testing called **Twister**

Our Question then became:

How can we

- make Twister run our Robot files and
- give us back the Robot test results?

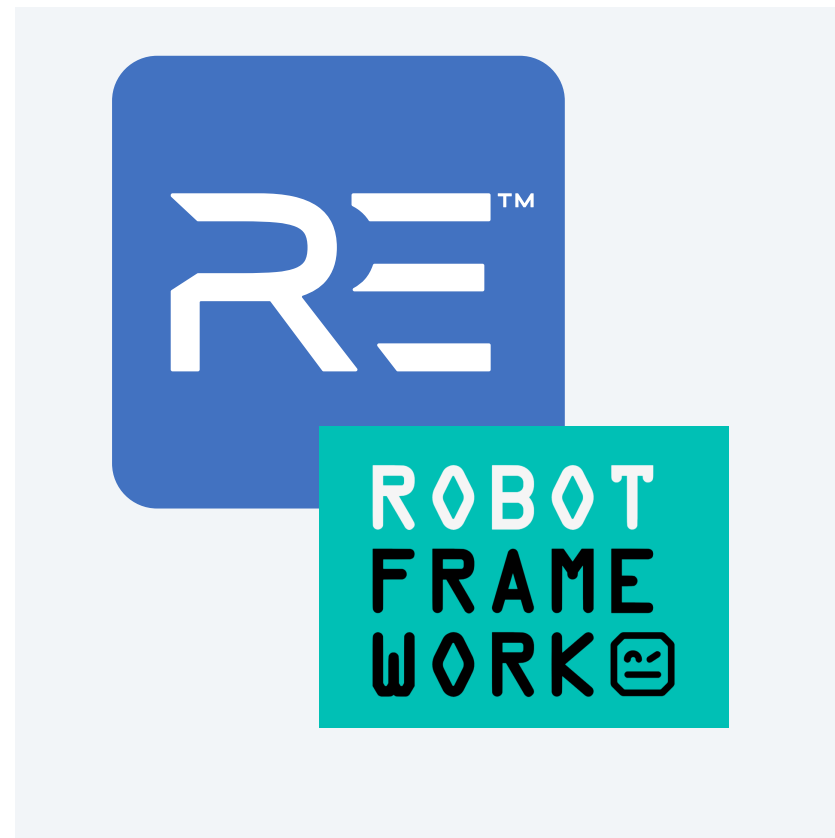


Current State of Robot integration into Twister

The good, the not-so-good, the missing



- Twister supports different test frameworks called harnesses
 - Console, Ztest, GoogleTest but also pytest and **Robot**
- Twister also supports different test monitors called handlers
 - Simulation (native), QEMU, Device
- However, not all harnesses can be used with all handlers
 - Robot harness tightly coupled to Renode simulator
 - Renode simulator configured only for a small list of (in-tree) boards
- Robot Framework integration provides special keywords
 - Start Emulation, Send Key to Uart, Wait For Outgoing Packet
- But currently usable with Renode simulator only



Introducing the robotframework Twister harness

Our changes and why they were necessary

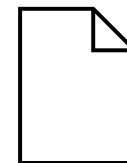


Goals of the robotframework harness:

- Run Robot Test Suites
- Execute on `native_sim`, QEMU and **real hardware**
- Provide Zephyr-specific Robot keywords
 - Run Device: Flash application and run
 - Run Command: Send commands via any transport (UART, MQTT, CAN)

Our strategy:

- Leverage existing pytest integration and code from (in-tree) `pytest_twister_harness` plugin
 - Invoke the robot CLI the same way as it is done for pytest
 - Figure out a way to pass relevant information from `twister` to robot
 - Implement required keywords using `XYZAdapter` classes from `pytest_twister_harness`
- Code available on https://github.com/ZEISS/zephyr/tree/zeiss/fretish_robot

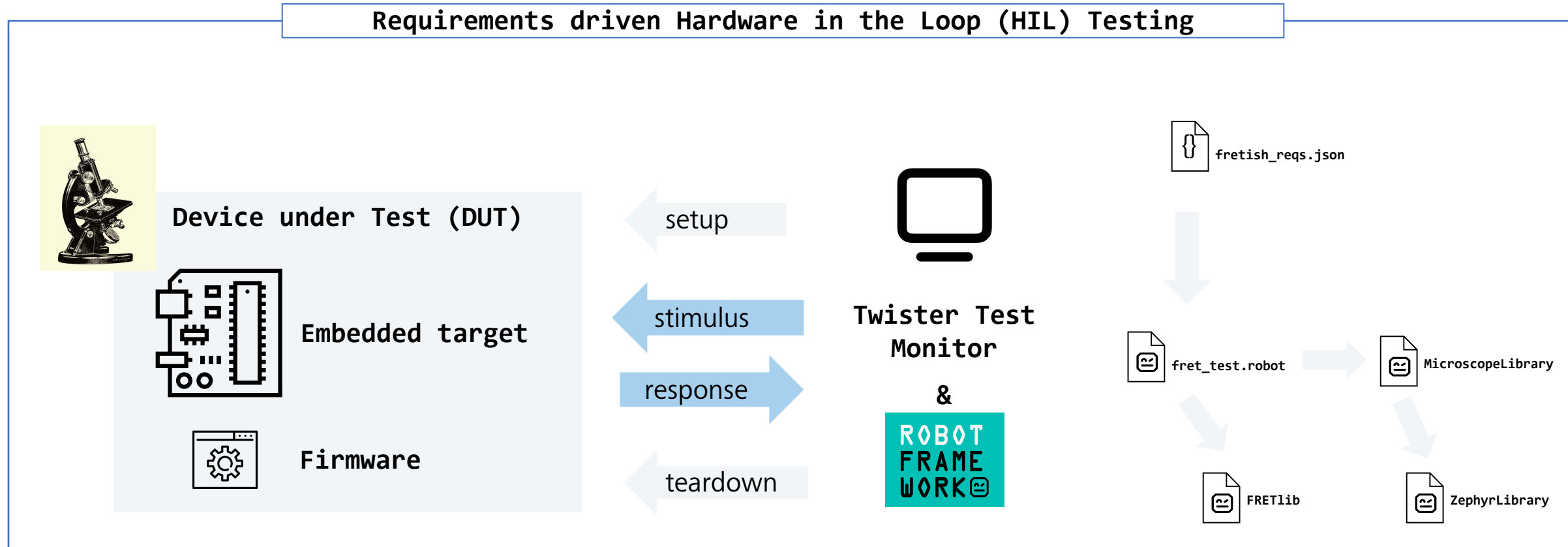


testcase.yaml

```
tests:
  sample.robot.shell_1:
    harness: robotframework
    platform_allow:
      - native_sim
      - mimxrt1020_evk
    tags:
      - test_framework
      - robot
```

Our current setup

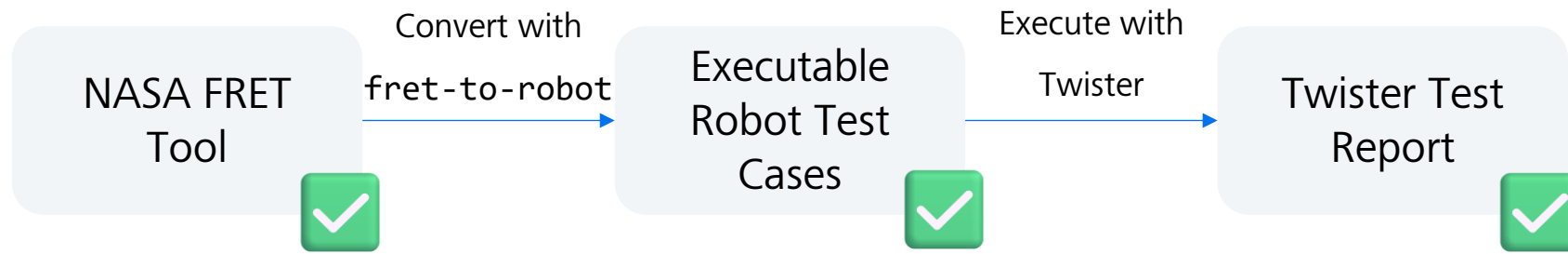
All together now ...



- Reminder: Robot tests contain general FRETish as well as domain-specific keywords
- Domain specific keywords still need to be implemented by hand, yet making use of generic TwisterLibrary for interacting with DUT

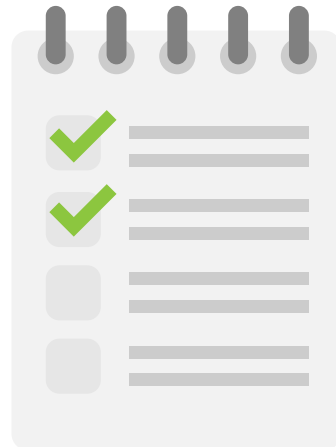
Our plan for today

All done!



What we achieved so far

If it doesn't scale it ain't worth a penny



Our pilot project captured
743 FRETish requirements



Presented tooling derived
594 test cases

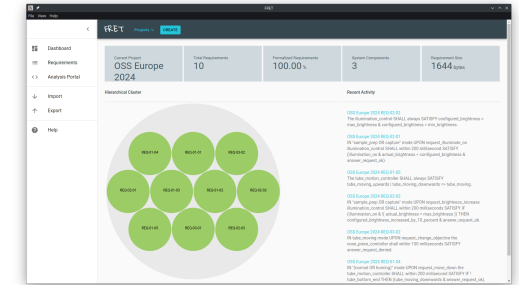
Benefits of FRETish Requirements & Test Automation

So why are we doing this?



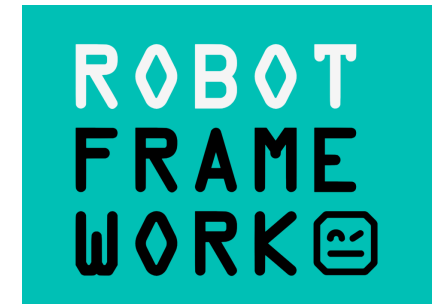
FRETish Requirements

- Clear and unambiguous semantics
- Machine-parsable
- Additional use cases like model checking, simulation, ...



Robot Framework

- Tests expressed in human-readable form and understandable for non-SW folks
- Derivable from FRETish requirements (traceability included)
- Obviously, not limited to automatically generated tests
- Keyword Libraries for re-use and separation of concerns



Zephyr

- Market-leading RTOS library and embedded firmware framework
- Built-in HIL-support
- Extensible and tunable to specific needs



Future Improvements for Robot Twister Integration

Let's work together!



Make general usage of Robot available upstream

- Allow robot scripts without renode · **Issue #64825** · zephyrproject-rtos/zephyr (github.com)¹⁾
- Integrate Robot Framework without Renode into twister by MP-StefanKraus · **Pull Request #67607** · zephyrproject-rtos/zephyr (github.com)²⁾

Not mergeable as is since community should also address technical debts in twister codebase

- Pytest harness works quite different from other harnesses
- Redundant implementations between Twister handlers and `pytest_twister_harness` adapters – which to choose?
- Provide consistent extension API for handlers/harnesses

(1) <https://github.com/zephyrproject-rtos/zephyr/issues/64825>

(2) <https://github.com/zephyrproject-rtos/zephyr/pull/67607>

The End

Come talk to us



Christian Schlotter
Software Architect
Carl Zeiss Meditec AG

Security Committee, Zephyr Project



Dr. Tobias Kästner
Solution Architect Medical IoT
Inovex GmbH

Safety Working Group, Zephyr Project
Maintainer Bridle Project, Tiac-Systems



Stefan Kraus
Senior Software Engineer
UL Solutions

Working for a safer world



Seeing beyond

The digitally controlled microscope

FRETish Requirements for non-rocket things



IN "(sample_prep|capture)" mode UPON request_change_objective the nose_piece_controller SHALL within 1 second SATISFY objective_lens_changed_clockwise_ok & answer_request_ok.

IN tube_moving mode UPON request_change_objective the nose_piece_controller SHALL within 100 milliseconds SATISFY answer_request_denied.

Scope

Conditions

Component*

SHALL*

Timing

Response*.

The digitally controlled microscope

FRETish Requirements for non-rocket things



UPON (tube_moving_upwards & tube_at_upper_end) the tube_motion_controller
SHALL at the next timepoint SATISFY tube_position_hold.

UPON (tube_moving_downwards & tube_at_bottom_end) the tube_motion_controller
SHALL at the next timepoint SATISFY tube_position_hold.

IN "(normal|homing)" mode UPON request_move_up the tube_motion_controller **SHALL** within 200 milliseconds
SATISFY IF !tube_upper_end THEN (tube_moving_upwards & answer_request_ok).

IN "(normal|homing)" mode UPON request_move_down the tube_motion_controller **SHALL** within 200 milliseconds
SATISFY IF !tube_bottom_end THEN (tube_moving_downwards & answer_request_ok).

The tube_motion_controller **SHALL** always
SATISFY IF (tube_moving_upwards | tube_moving_downwards) THEN tube_moving.

Scope

Conditions

Component*

SHALL*

Timing

Response*.

FRET exported requirements

FRET tool exports to simple JSON



IN "(sample_prep|capture)" mode

UPON request_illuminate_on

the illumination_controller

SHALL

within 200 milliseconds

SATISFY (

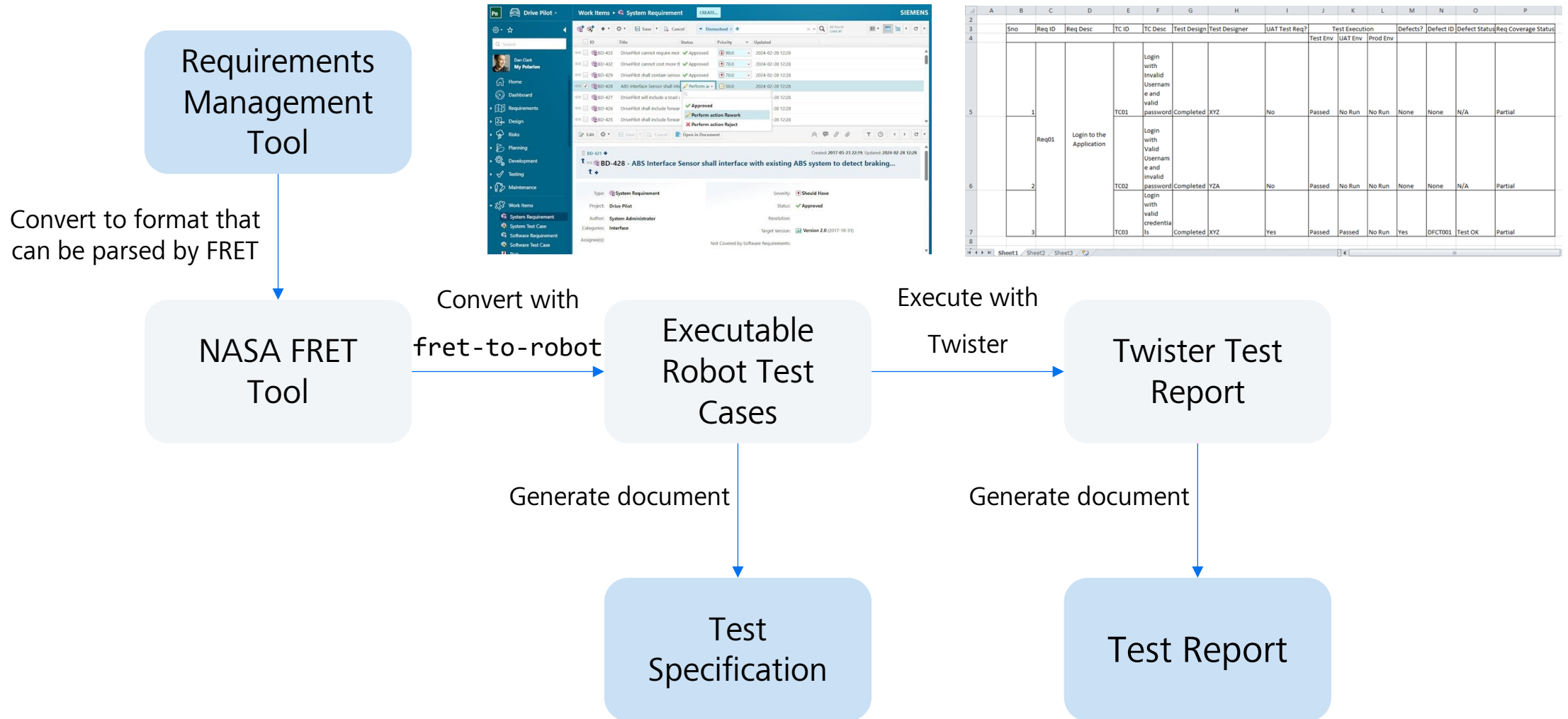
illumination_on &
actual_brightness = configured_brightness &
answer_request_ok

)

```
{  
  "reqid": "REQ-02-01",  
  "fulltext": "IN \"(sample_prep|capture)\" mode UPON  
request_illuminate_on illumination_control SHALL within 200  
millisecond SATISFY (illumination_on & actual_brightness =  
configured_brightness & answer_request_ok)",  
  "semantics": {  
    "scope_mode": "\"(sample_prep|capture)\"",  
    "regular_condition_unexp_pt": "request_illuminate_on",  
    "component_name": "illumination_control",  
    "timingTextRange": [ 86, 107 ],  
    "post_condition_unexp_ft": "((illumination_on &  
(actual_brightness = configured_brightness)) & answer_request_ok)",  
    "variables": [ "request_illuminate_on", ...]  
  }  
}
```

What we did not cover

There is only so much you can say in 30 mins



Current State of Robot integration into Twister



- Upstream Zephyr supports running Robot Test Suites in Twister via `robot` harness
- Harness employs `renode-test` to run a Robot Test Suite in Renode
 - Starts Renode in the background
 - Configures it to allow Robot Framework to connect to Renode
- Robot Framework integration in Renode provides special keywords
 - `Start Emulation`
 - `Send Key to Uart`
 - `Wait For Outgoing Packet`
 - ...

