bilihome

natural care for newborns with jaundice

Testdag NL - Utrecht

October 31 2025 - Ronald van Doorn

bilihome

Agenda

- Background Bilihome
- Medical Device Technology
- Software lifecycle management
- Embedded testing with MBT

10% of all newborns need Jaundice treatment

The current treatment is **blue light phototherapy**, which breaks down the elevated bilirubin levels

Every 2.3 seconds, a newborn needs jaundice treatment. Annually, 14 million babies face the risk of neurological damage if untreated, with a severe drop in quality of life.

Closer Look on the world map

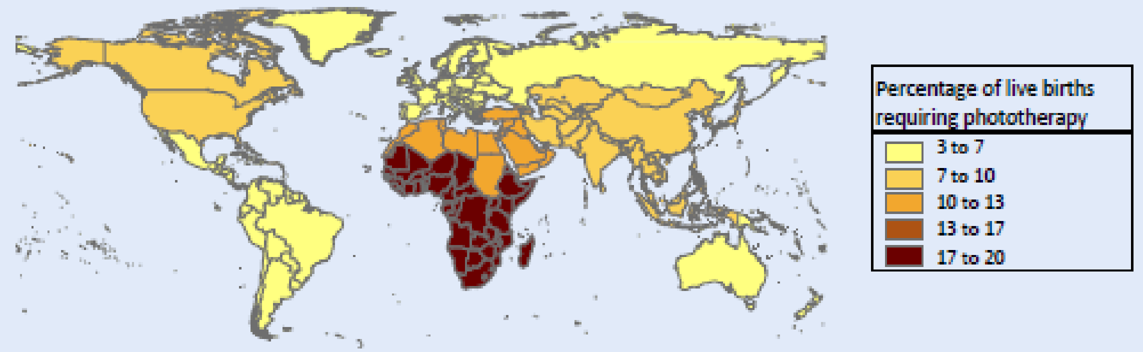


Figure 2. Estimated total annual need for phototherapy treatment by region.

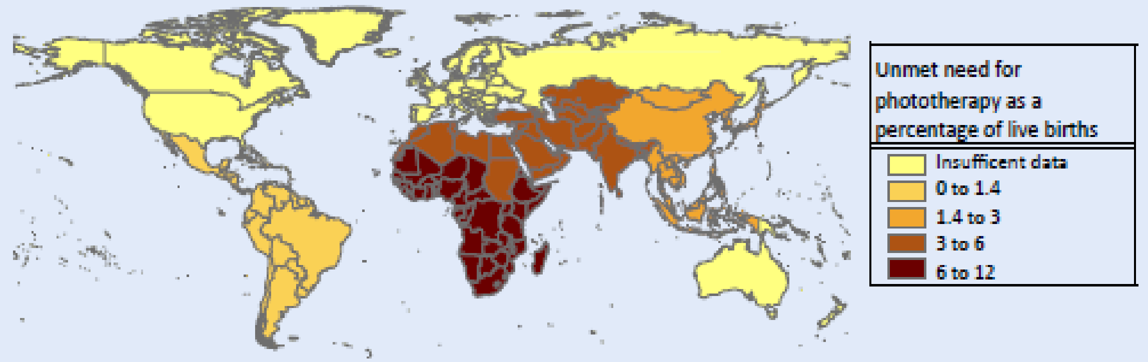


Figure 3. Estimated total unmet need for phototherapy treatment by region.

Globally there is a high prevalence for jaundice treatment

14 million live patients

10.5% treatment prevalence

The cause?

Bilirubin, a waste product of red blood cells, reaching its peak around three days after birth. Without phototherapy treatment, it leads to hyperbilirubinemia, and the risk of neurological damage is high.

Phototherapy

Current situation…

Newborns are undressed, blindfolded, and isolated in incubators or fiberoptic bags. Parents feel helpless and separated.

Beds fill up, and nurses struggle with staff shortages; 30% experience severe fatigue.

Even if babies aren't sick, the healthcare systems spend 3 billion annually due to inefficient jaundice treatments.

bilihome

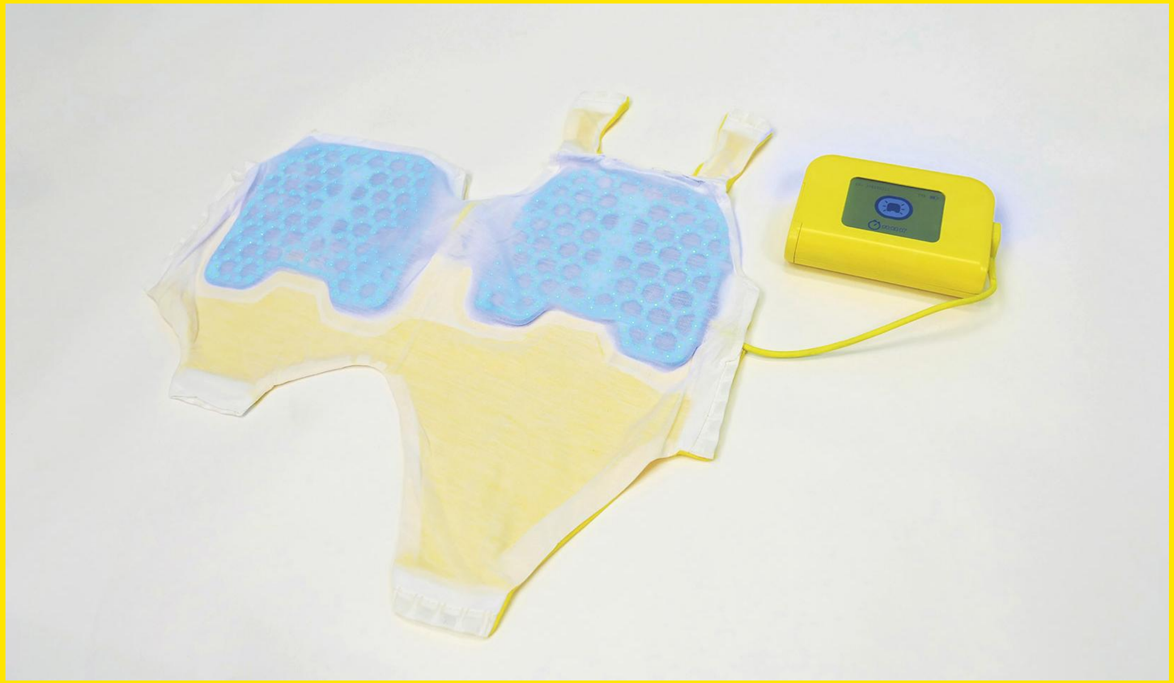
Zero Separation

We envision a world
where newborns are
comforted, naturally in
the arms of their parents
no matter what

Families stay together in a stressless environment

With our patented technology at the core, we have built care paths for Zero-Separation, transforming phototherapy into a wearable device to comfort newborns.

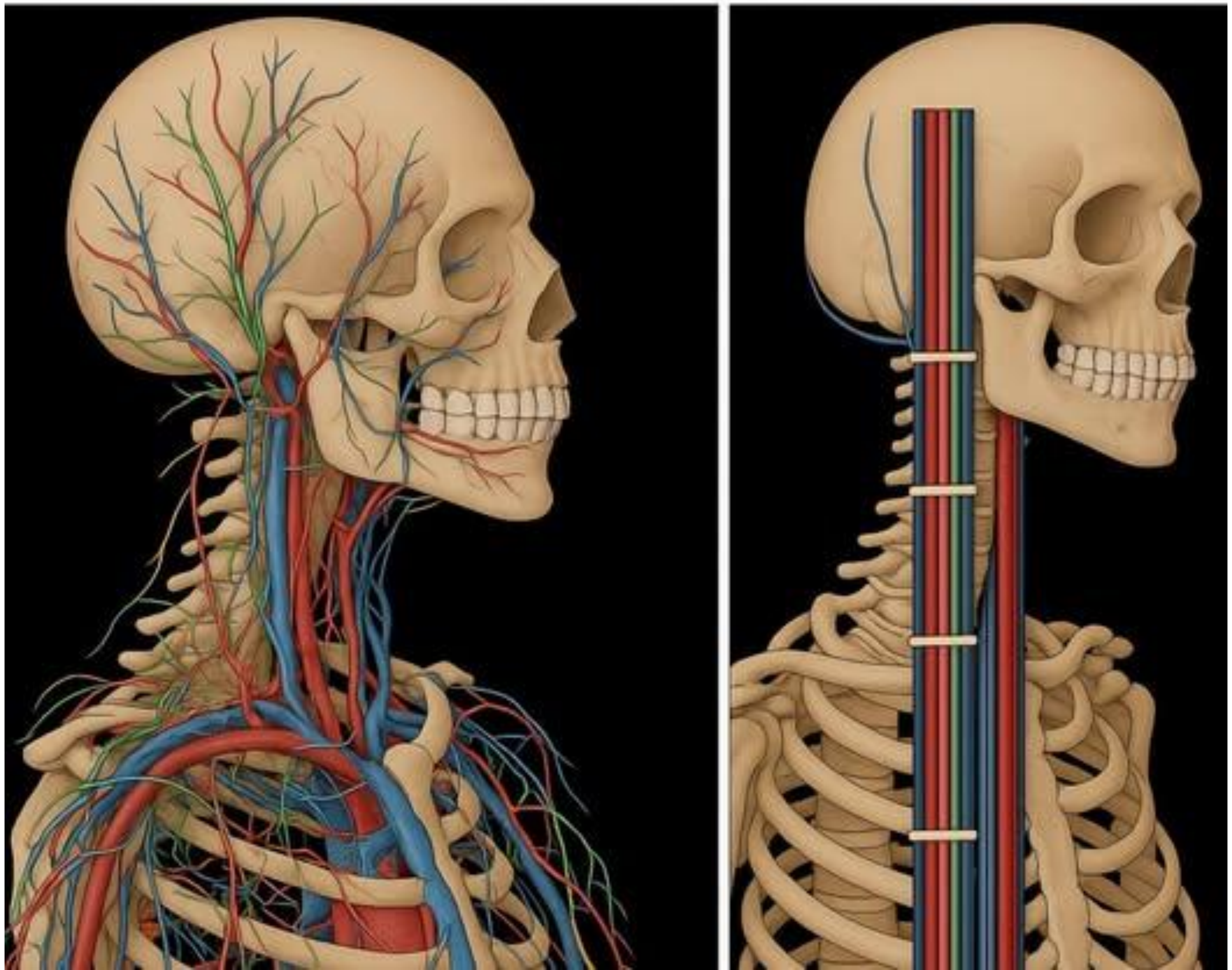
Bilihome's Patented Technology for Zero Separation



Bilihome offers the lightest and smallest wearable phototherapy, the only true mobile solution.

- It is a specially designed organic romper.
- The flexible open mesh pads radiate blue light onto the skin, without shielding the eyes.
- The remote care app guides parents through the treatment.

When an engineer goes to medical school



Medical device technology

Medical device technology is an innovative and regulated market.

- Requires compliance with all kinds of standards depending on the field of application
- Requires regulatory approvals (CE, FDA, etc.).
- Quality shall be top priority.

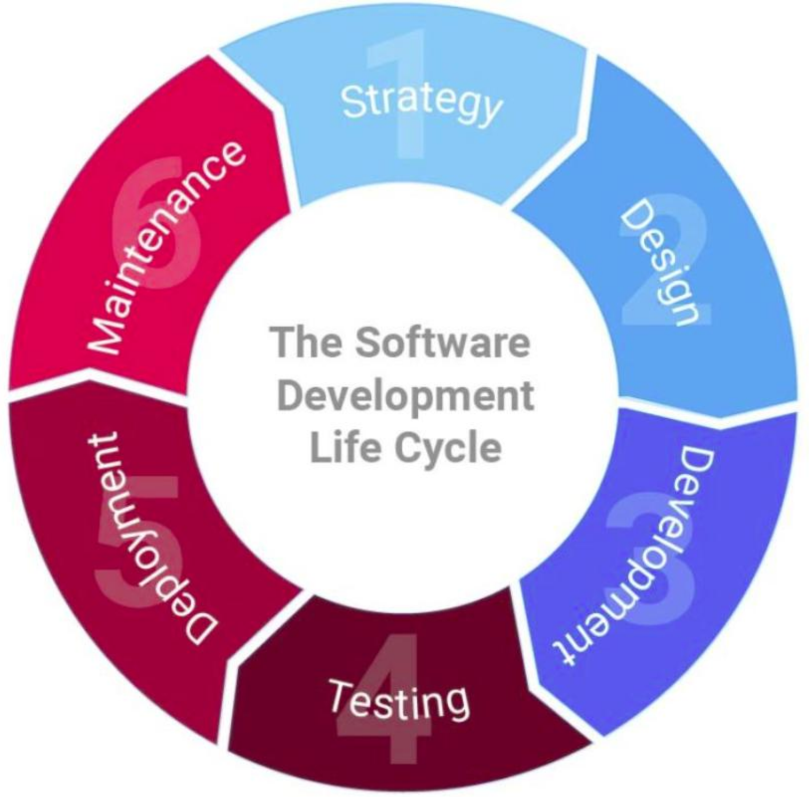
Medical device technology

For the Jauni Wearable Phototherapy, we had to show compliance with:

- EU Medical Device Regulation (MDR)
- ISO 13485 - QMS
- IEC 60601-1 - Main part on basic safety and essential performance of MD
- IEC 60601-1-2 - EMC
- IEC 60601-1-6 - Usability
- IEC 60601-1-11 - Homecare
- IEC 60601-2-50 – Phototherapy equipment
- IEC 62471 – Photobiological safety
- IEC 14971 - Risk management
- ISTA - Transport testing
- **IEC 62304 - Software lifecycle**

Software lifecycle management

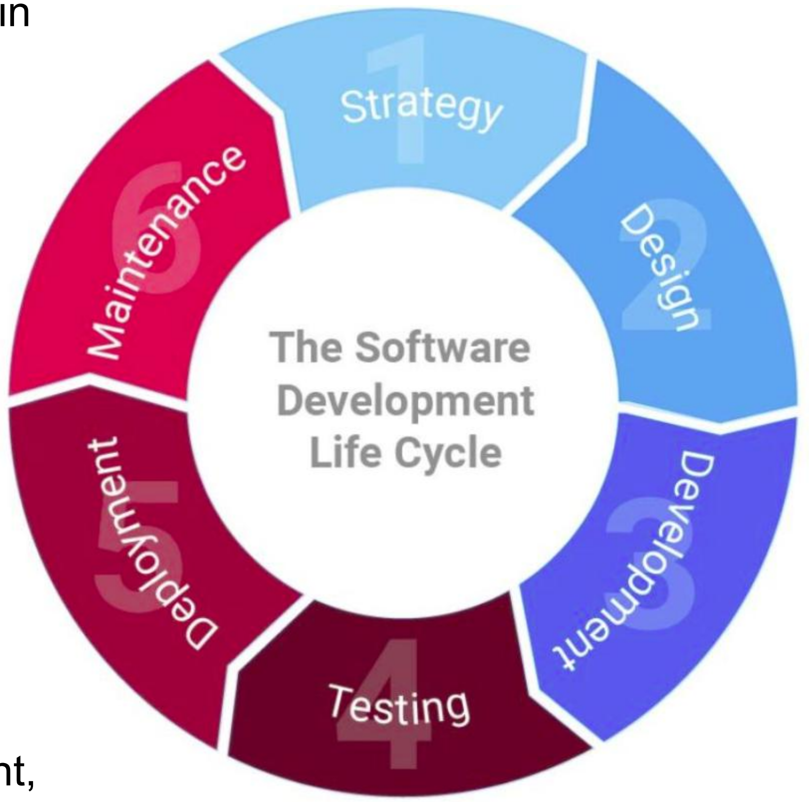
- Standards like **ISO 26262** (Automotive), **DO-178C** (Aviation), and **IEC 62304** (Healthcare) are essential for ensuring safety, reliability, and compliance in safety-critical software development.
- They provide structured frameworks that shape QA processes and **aim to reduce risks**, ultimately protecting lives (and businesses).
- Implementation might be challenging.
- The rewards are enhanced safety, regulatory compliance, and stakeholder trust.
- At the end it delivers safe(r), high-quality software.



Software lifecycle management

IEC 62304 - Software lifecycle management:

- Emphasizing traceability, documentation, and risk-based processes to maintain regulatory compliance to ensure patient (and operator) safety
- Classify software by safety impact (Class A, B, or C) and mandate controls accordingly (IEC 62304:2026 defines only two classes: Low and High)
- Multiple classifications within a single device are allowed; for example, the same device may have Class B alarm systems and Class C subsystems for core life-supporting functions
- It requires:
  - Planning (software development plan)
  - Design and development (safety and security designs)
  - Risk management (risk identification, FMEA, risk control measures)
  - Verification and validation (test reporting)
  - Maintenance activities (all of the above)
- The higher the classification, the stricter the requirements for risk management, verification, and documentation
- Whenever possible, implement hardware risk control measures 😉



Software lifecycle management

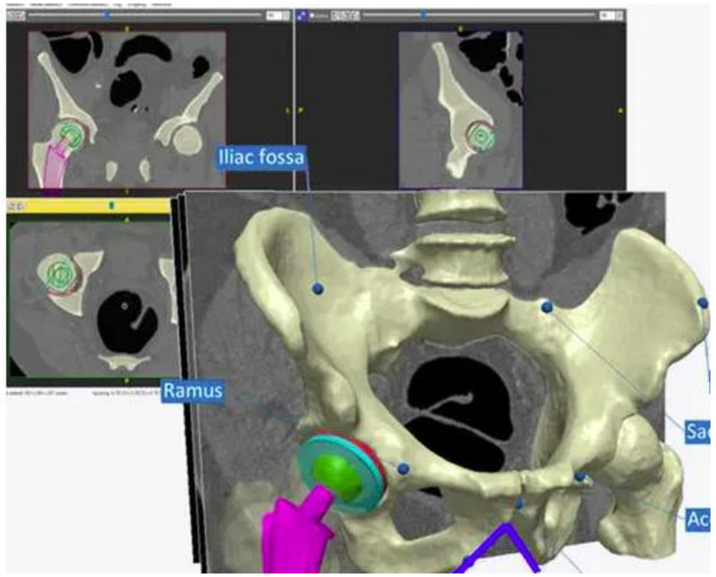
Class A

No injury or damage to health is possible

No risk to the patient's health!

Fitness tracker

Diagnostic viewer (no interpretations)



Software lifecycle management

Class B

Non-serious injury is possible

Reversible or temporary harm, but not life-threatening!

Blood pressure measurement

Lung function diagnostics

Infusion pump

Software lifecycle management

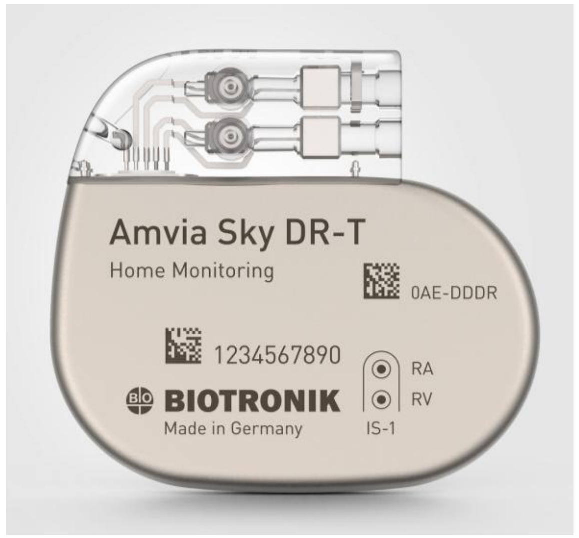
Class C

Death or serious injury is possible

Non-reversible injury or even death!

Anesthesia device

Pacemaker



Software lifecycle management

Jauni Wearable Phototherapy classifications:

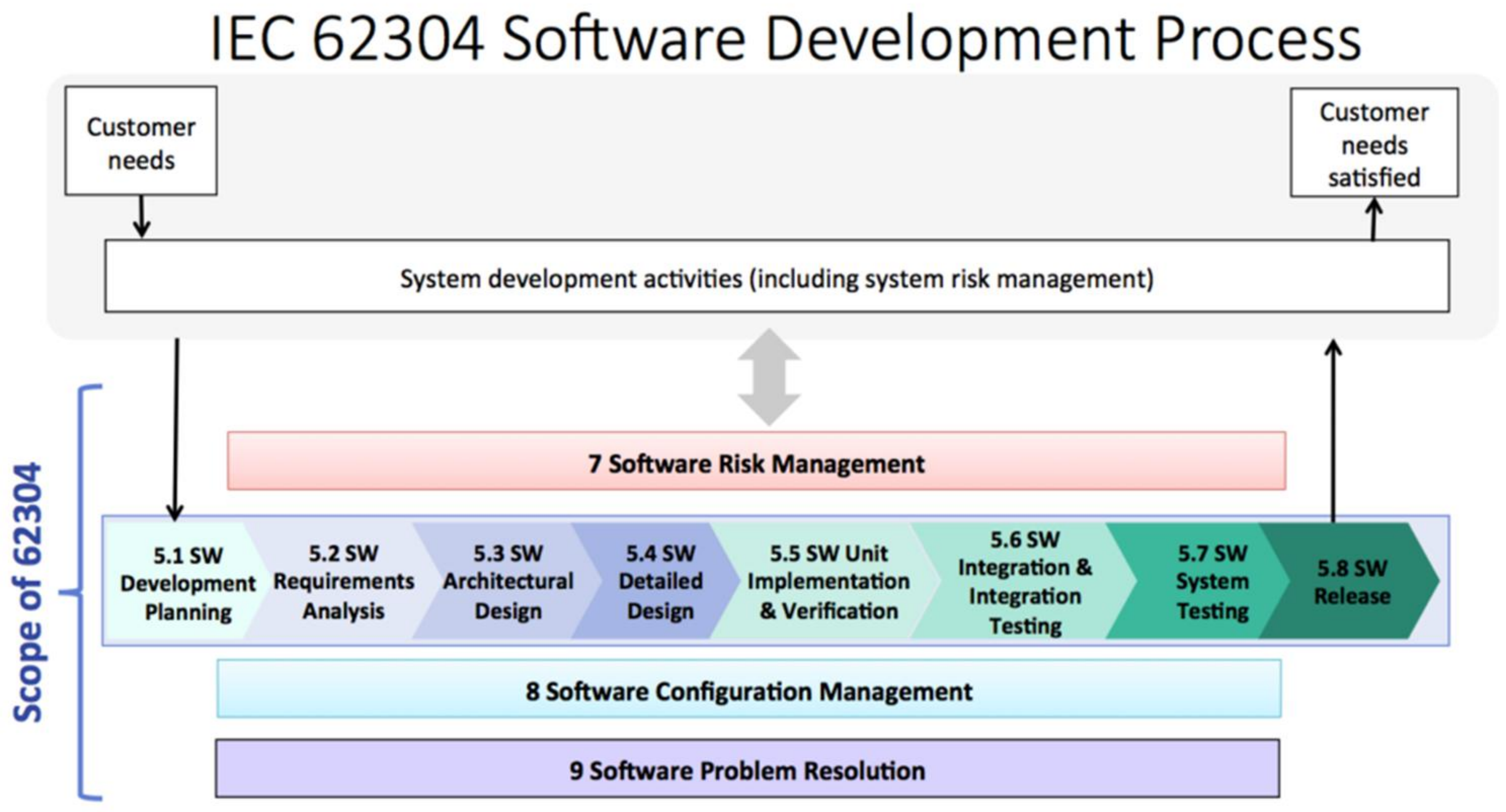
- MDR: **Class IIa**
- Software safety: **Class A**
  - Not immediately life-threatening even if the software fails
  - Multiple functional hardware safety measures implemented

Software lifecycle management

What should be tested according **Class A**:

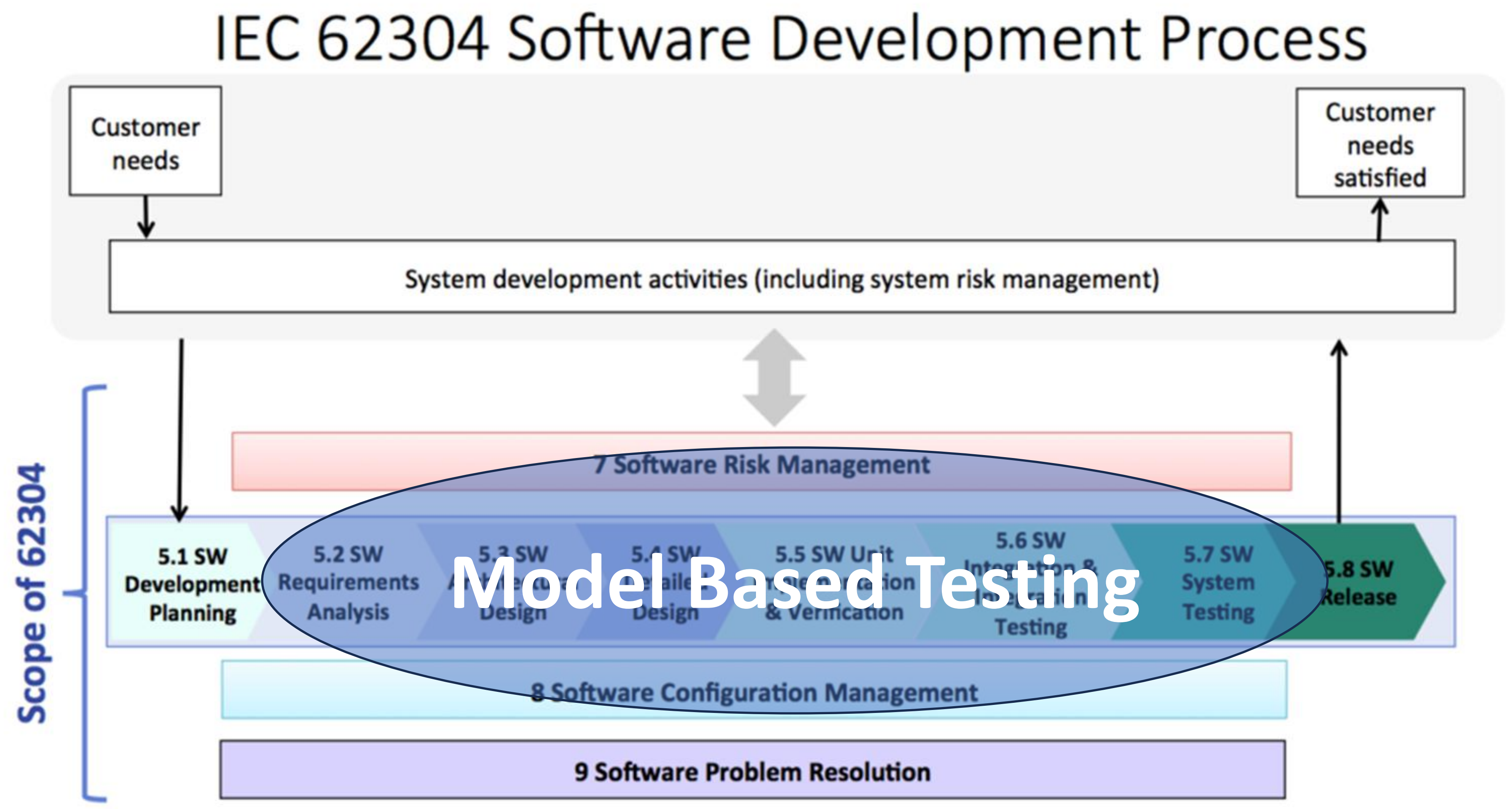
- Any software risks / risk control measures. In our case the detection of all self-test safety mechanisms.
- System-level testing to verify product requirements.
- Integration testing between subsystems is not mandatory, but we applied Model Based Testing techniques.
- Module or unit testing is not required.

Software lifecycle management

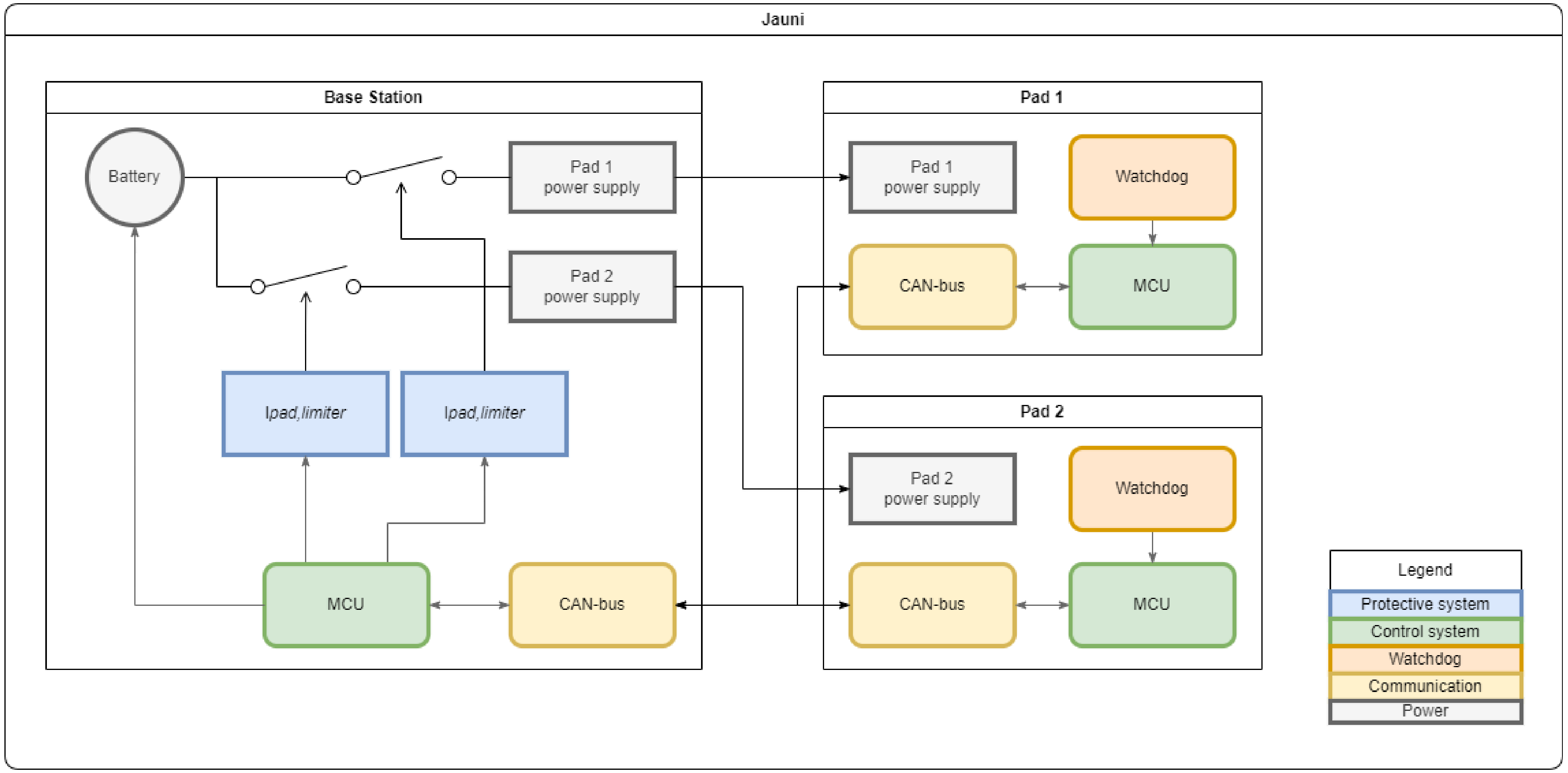


IEC 62304 Software Development Process
Customer needs
Customer needs satisfied
System development activities (including system risk management)
Scope of 62304
7 Software Risk Management
5.1 SW Development Planning
5.2 SW Requirements Analysis
5.3 SW Architectural Design
5.4 SW Detailed Design
5.5 SW Unit Implementation & Verification
5.6 SW Integration & Integration Testing
5.7 SW System Testing
5.8 SW Release
8 Software Configuration Management
9 Software Problem Resolution

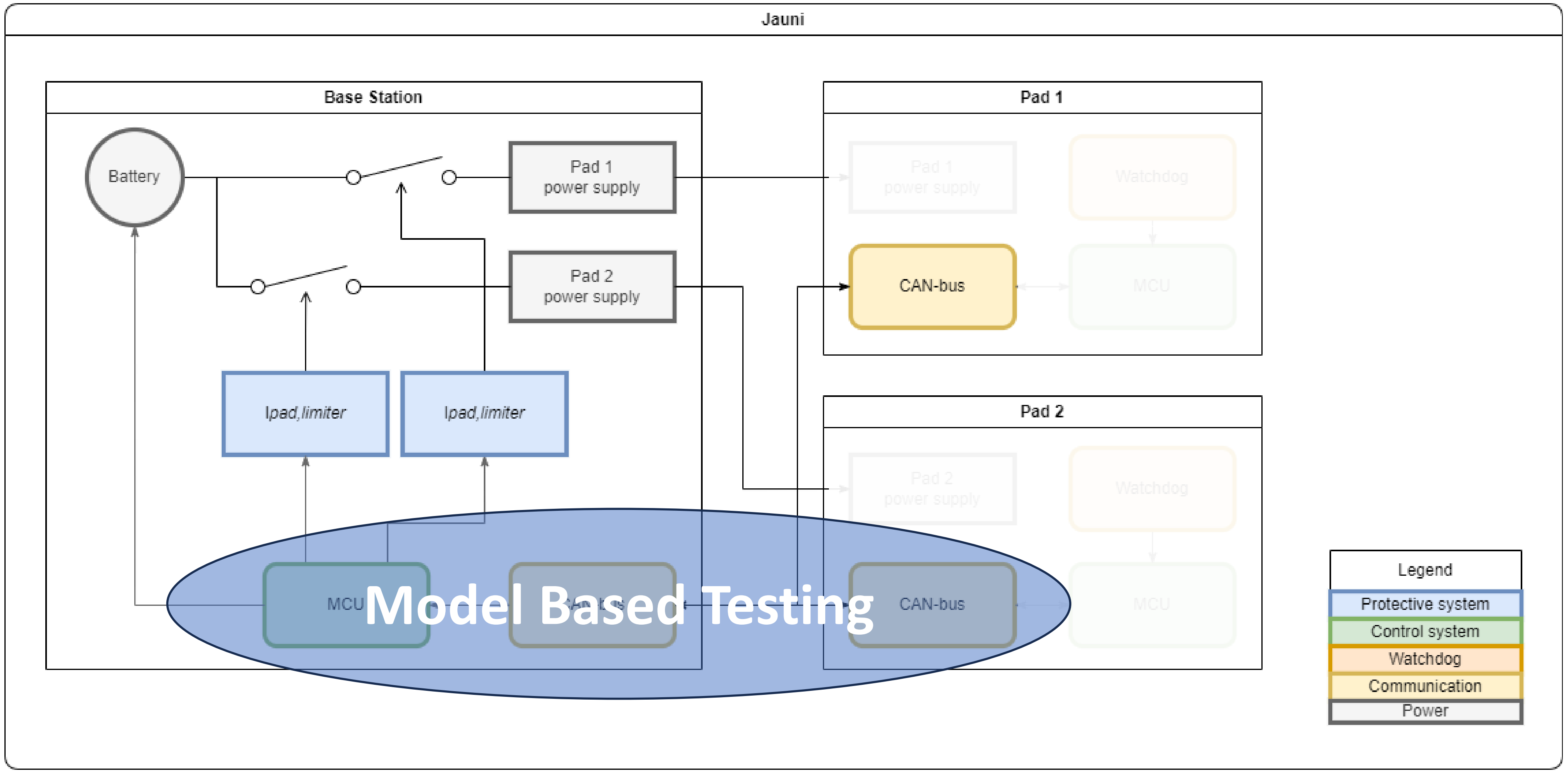
Software lifecycle management


IEC 62304 Software Development Process
Customer needs
Customer needs satisfied
System development activities (including system risk management)
7 Software Risk Management
5.1 SW Development Planning
5.2 SW Requirements Analysis
5.3 SW Architectural Design
5.4 SW Detailed Design
5.5 SW Unit Implementation & Verification
5.6 SW Integration & Integration Testing
5.7 SW System Testing
5.8 SW Release
8 Software Configuration Management
9 Software Problem Resolution
Scope of 62304
Model Based Testing

Architecture and design



Architecture and design



Software architecture and design

Most relevant for testing:

- Self-test and sanity check
- Heartbeats
- Treatment activation, deactivation, pausing, and resuming
- Hardware faults
- Happy and erroneous behavior

Less important:

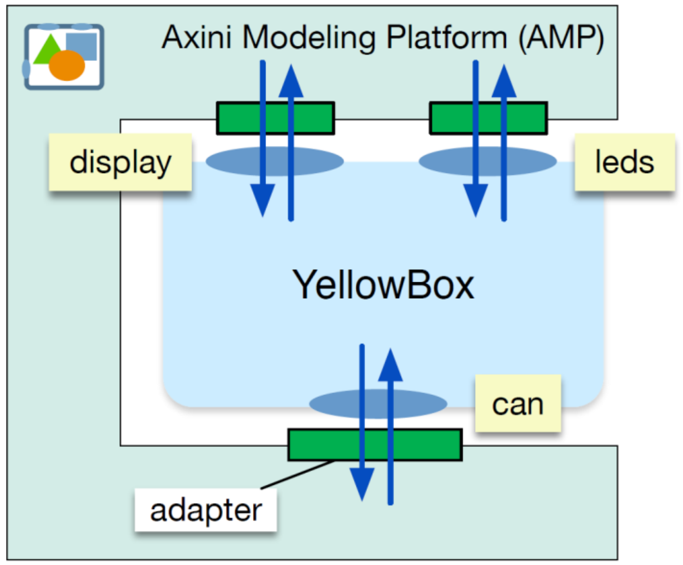
- Downloading log files
- Wifi

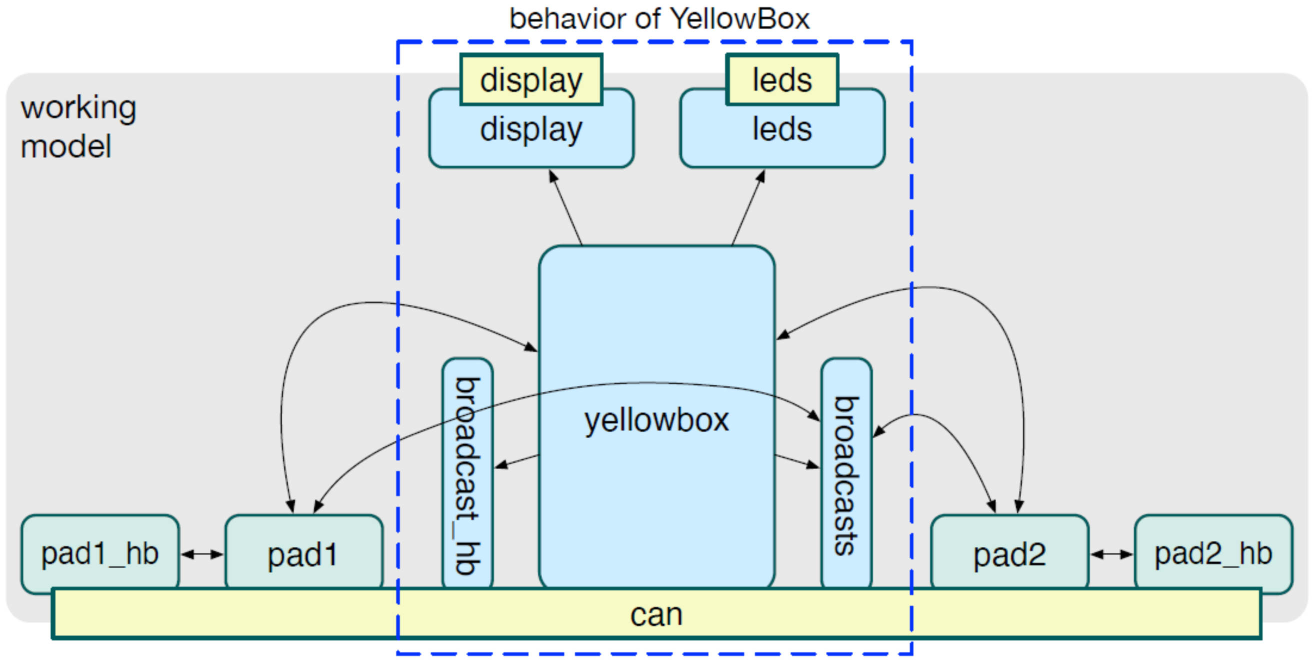
Software verification

Verification methods and levels:

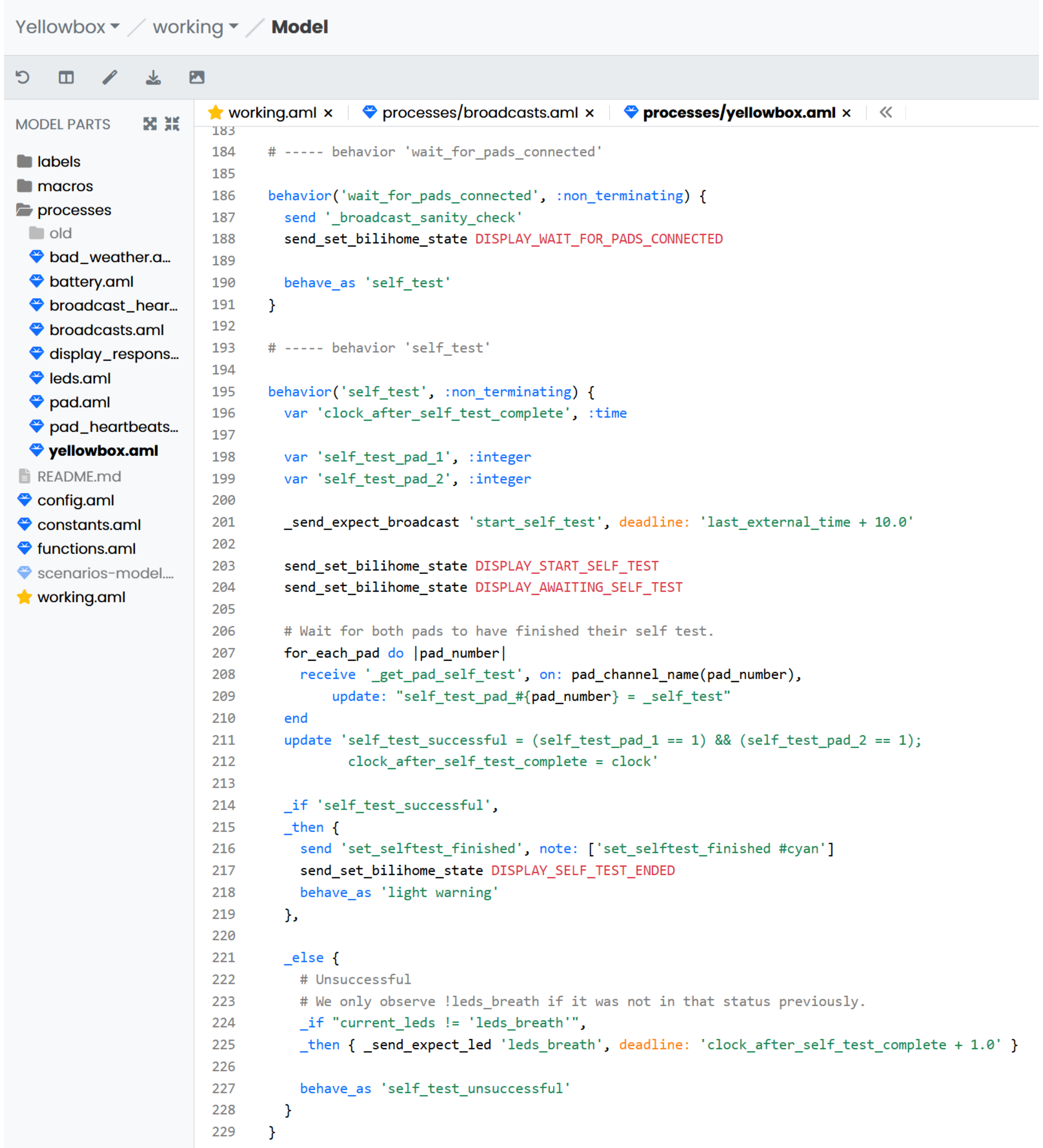
- No hardware in the loop!
- Individual testing on all of the functional safety aspects (non-repetitively)
- Integration tests between Yellow Box and Pads by means of MBT
- System level testing to verify the overall product safety. Final test executed prior to a new SW release

Model based testing





MBT



Yellowbox / working / **Model**

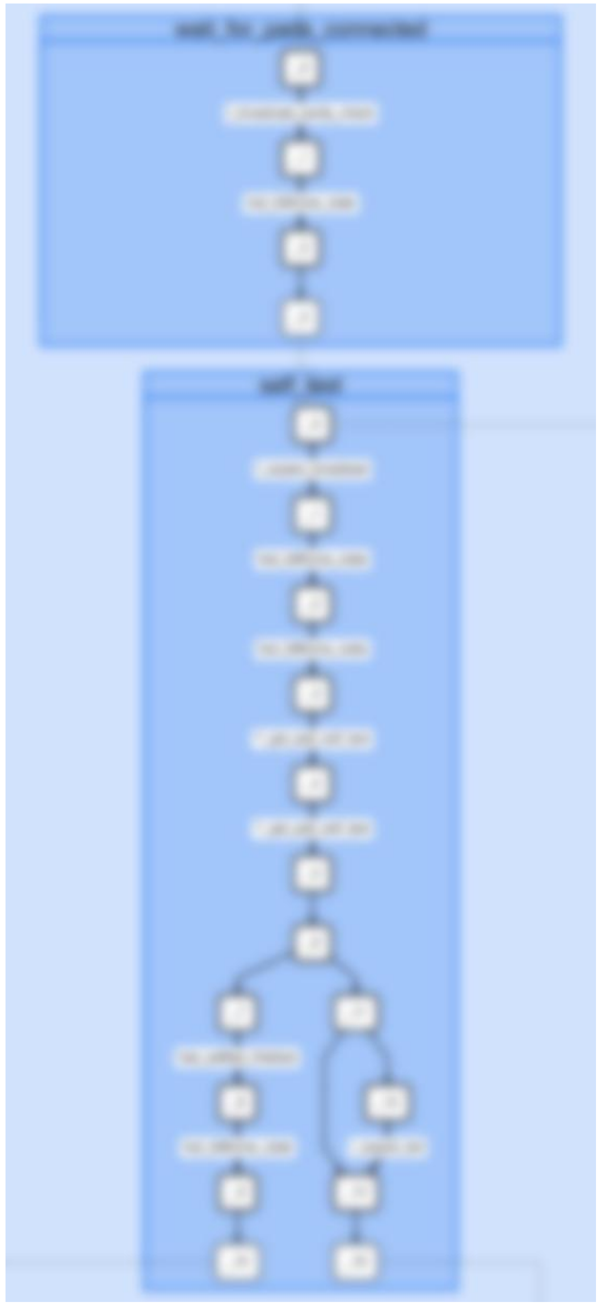
MODEL PARTS

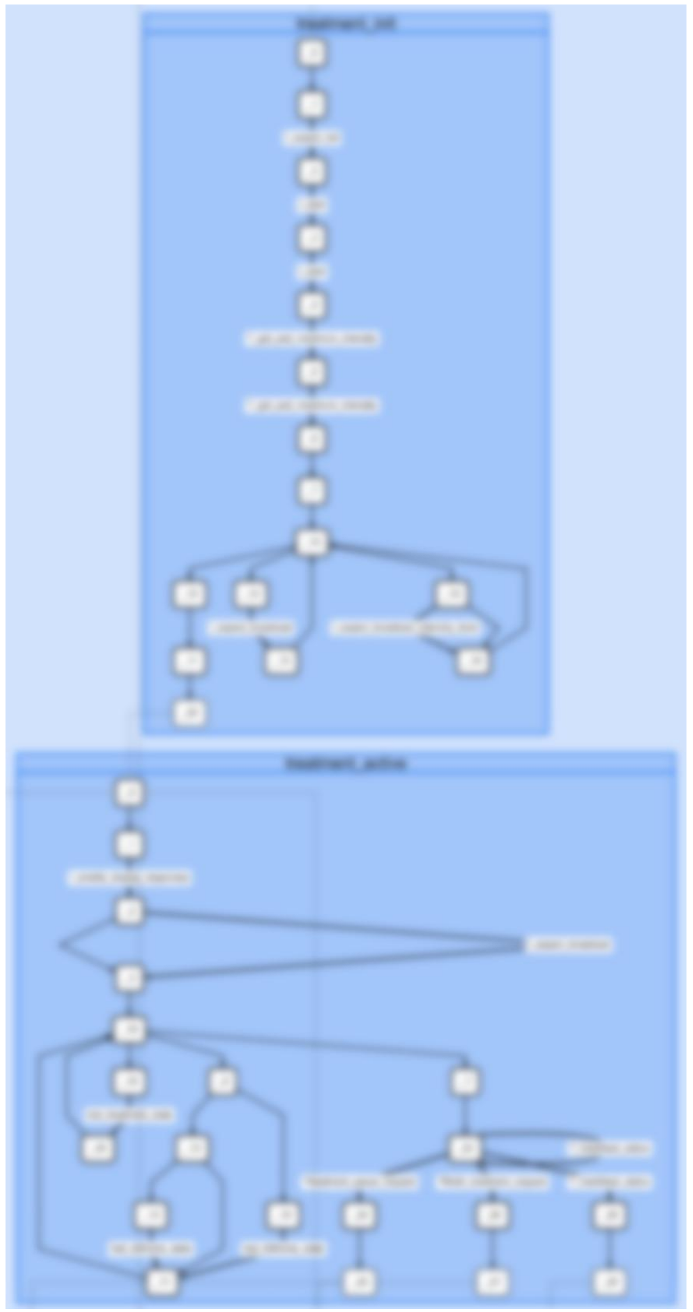
- labels
- macros
- processes
  - old
  - bad_weather.a...
  - battery.aml
  - broadcast_hear...
  - broadcasts.aml
  - display_respons...
  - leds.aml
  - pad.aml
  - pad_heartbeats...
  - **yellowbox.aml**
- README.md
- config.aml
- constants.aml
- functions.aml
- scenarios-model...
- working.aml

working.aml × | processes/broadcasts.aml × | **processes/yellowbox.aml** ×

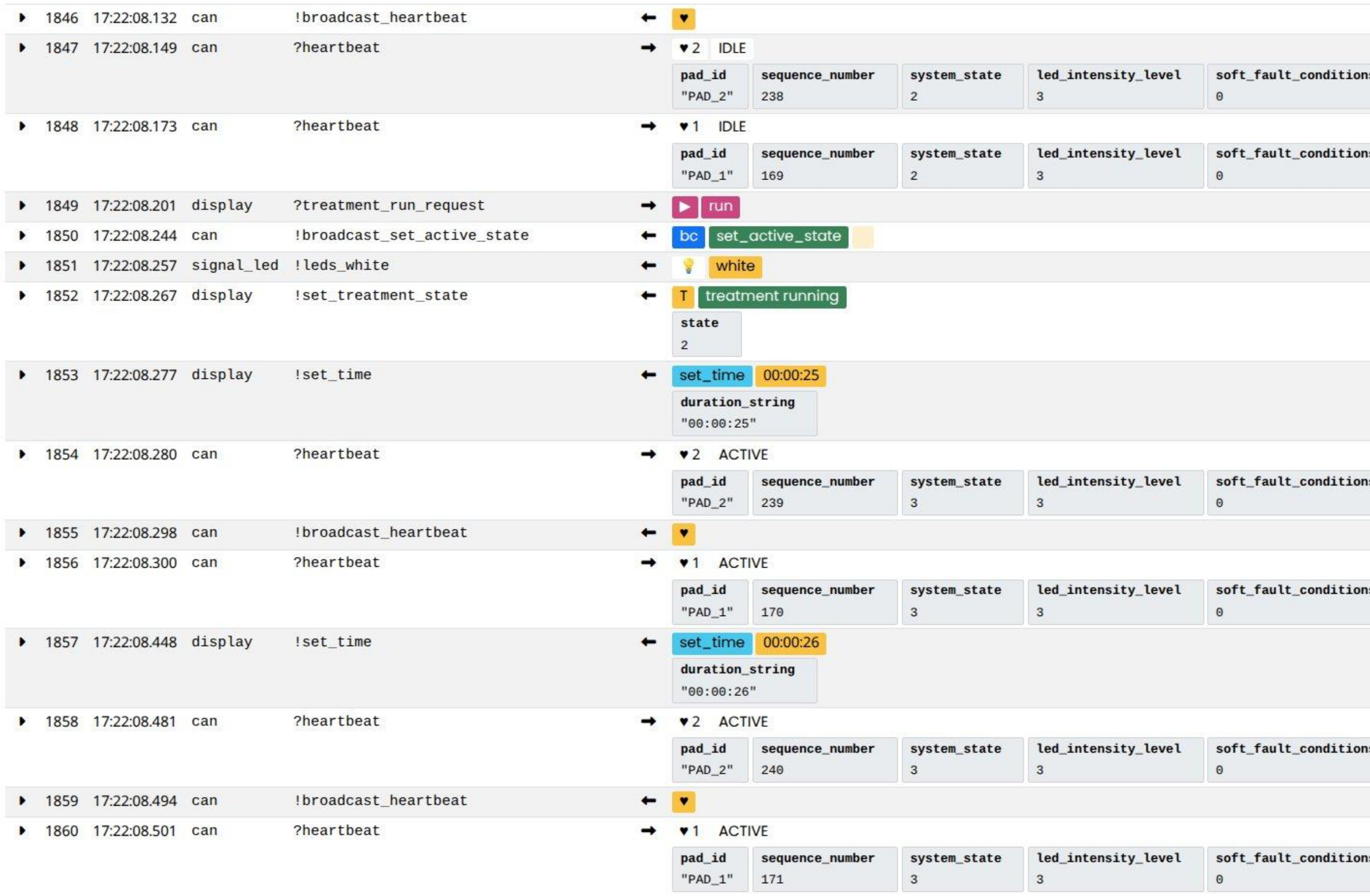
```
# ----- behavior 'wait_for_pads_connected'

behavior('wait_for_pads_connected', :non_terminating) {
  send '_broadcast_sanity_check'
  send_set_bilihome_state DISPLAY_WAIT_FOR_PADS_CONNECTED

  behave_as 'self_test'
}

# ----- behavior 'self_test'

behavior('self_test', :non_terminating) {
  var 'clock_after_self_test_complete', :time

  var 'self_test_pad_1', :integer
  var 'self_test_pad_2', :integer

  _send_expect_broadcast 'start_self_test', deadline: 'last_external_time + 10.0'

  send_set_bilihome_state DISPLAY_START_SELF_TEST
  send_set_bilihome_state DISPLAY_AWAITING_SELF_TEST

  # Wait for both pads to have finished their self test.
  for_each_pad do |pad_number|
    receive '_get_pad_self_test', on: pad_channel_name(pad_number),
        update: "self_test_pad_#{pad_number} = _self_test"
  end
  update 'self_test_successful = (self_test_pad_1 == 1) && (self_test_pad_2 == 1);
          clock_after_self_test_complete = clock'

  _if 'self_test_successful',
  _then {
    send 'set_selftest_finished', note: ['set_selftest_finished #cyan']
    send_set_bilihome_state DISPLAY_SELF_TEST_ENDED
    behave_as 'light warning'
  },

  _else {
    # Unsuccessful
    # We only observe !leds_breath if it was not in that status previously.
    _if "current_leds != 'leds_breath'",
    _then { _send_expect_led 'leds_breath', deadline: 'clock_after_self_test_complete + 1.0' }

    behave_as 'self_test_unsuccessful'
  }
}
```

MBT


wait_for_pads_connected
self_test


treatment_init
treatment_active

MBT



```
▸ 1846  17:22:08.132  can         !broadcast_heartbeat      ←  ♥
▸ 1847  17:22:08.149  can         ?heartbeat                →  ♥2  IDLE
                                                               pad_id    sequence_number  system_state  led_intensity_level  soft_fault_condition
                                                               "PAD_2"   238              2             3                    0
▸ 1848  17:22:08.173  can         ?heartbeat                →  ♥1  IDLE
                                                               pad_id    sequence_number  system_state  led_intensity_level  soft_fault_condition
                                                               "PAD_1"   169              2             3                    0
▸ 1849  17:22:08.201  display     ?treatment_run_request    →  ▶ run
▸ 1850  17:22:08.244  can         !broadcast_set_active_state ← bc set_active_state
▸ 1851  17:22:08.257  signal_led  !leds_white               ←  💡 white
▸ 1852  17:22:08.267  display     !set_treatment_state      ←  T treatment running
                                                               state
                                                               2
▸ 1853  17:22:08.277  display     !set_time                 ←  set_time 00:00:25
                                                               duration_string
                                                               "00:00:25"
▸ 1854  17:22:08.280  can         ?heartbeat                →  ♥2  ACTIVE
                                                               pad_id    sequence_number  system_state  led_intensity_level  soft_fault_condition
                                                               "PAD_2"   239              3             3                    0
▸ 1855  17:22:08.298  can         !broadcast_heartbeat      ←  ♥
▸ 1856  17:22:08.300  can         ?heartbeat                →  ♥1  ACTIVE
                                                               pad_id    sequence_number  system_state  led_intensity_level  soft_fault_condition
                                                               "PAD_1"   170              3             3                    0
▸ 1857  17:22:08.448  display     !set_time                 ←  set_time 00:00:26
                                                               duration_string
                                                               "00:00:26"
▸ 1858  17:22:08.481  can         ?heartbeat                →  ♥2  ACTIVE
                                                               pad_id    sequence_number  system_state  led_intensity_level  soft_fault_condition
                                                               "PAD_2"   240              3             3                    0
▸ 1859  17:22:08.494  can         !broadcast_heartbeat      ←  ♥
▸ 1860  17:22:08.501  can         ?heartbeat                →  ♥1  ACTIVE
                                                               pad_id    sequence_number  system_state  led_intensity_level  soft_fault_condition
                                                               "PAD_1"   171              3             3                    0
```
]

Model based testing

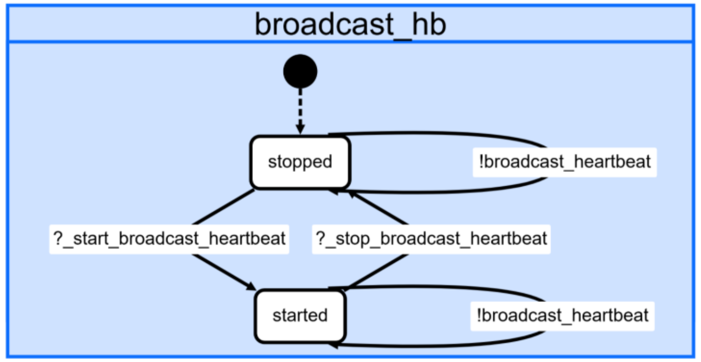
Why model based testing for Bilihome?

- Documentation has a stuffy appearance, so why not create lots of fun while designing your application?
- Testing starts right away, and any improvement can be tested (First Time Right).
- Provides early feedback on the design of your software.
- **Fail Hard - Fail Fast - Fail Often**
- Improves team cooperation and includes architects, software designers, coding and testers in a single approach.
- What if your test results can be generated?

Model based testing

Why model based testing for Bilihome?

- Tests are in line with the latest software designs.
- Documentation is in line with the software designs.
- Testing implemented at any interfaceable level.
- Direct and quick feedback.
- Allows testing the happy flow and bad weather scenarios.
- Tests your artifacts.
- The more complex the software, the better MBT will become
- Integrates at all levels, system, integration and unit level testing.



Model based testing

Our ‘lessons learned’

- ✓ Testing showed us several anomalies in our released software, without showing up in previous test (scripts).
- ✓ Added automated corner case testing (AI-initiated).
- ✓ Automated testing.
- ✓ Enhanced our product safety and usability.
- ✗ Started way too late, so had to do the modelling and testing retrospectively.
- ✓ Modelling starts with your requirements definition and creates a testable model throughout the full software development and maintenance lifecycle.
- ✓✗ Not specifically for UI testing.

Model based testing

- Take testing and test models into account asap, but at least during the design phase.
- MBT can be applied to any level within the software as long as it can be interfaced. This thereby ensures clear software boundaries within your designs.
- MBT software testing is acceptable to the Notified Body.
- No need to integrate hardware, allowing easy initiation of a test case simulating broken hardware.
- Incorporate continuous testing in your development process in early stages and integrate it with your CI/CD process.

bilihome

axini

Ronald van Doorn
r.vandoorn@bilihome.org

Machiel van der Bijl
machiel.van.der.bijl@axini.com