Dry-Run Procedures & Sequencer Tutorial

Ralph J. Steinhagen

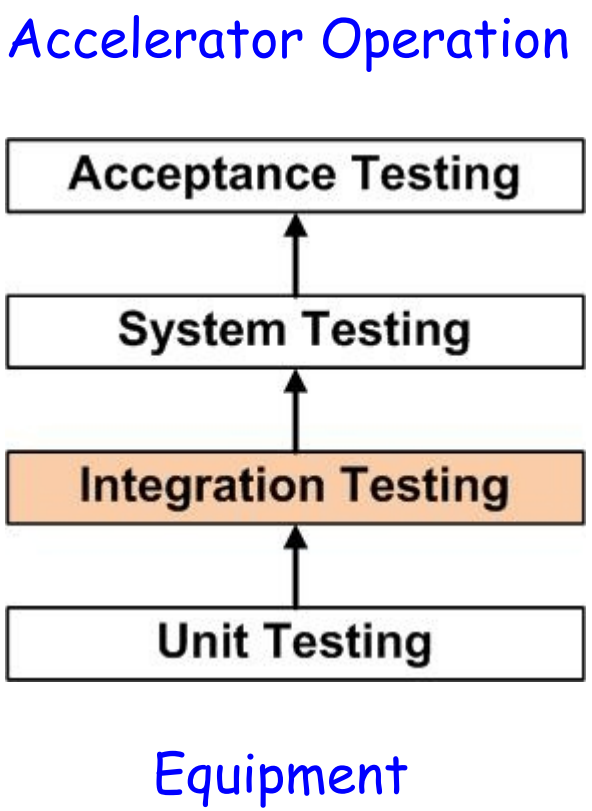
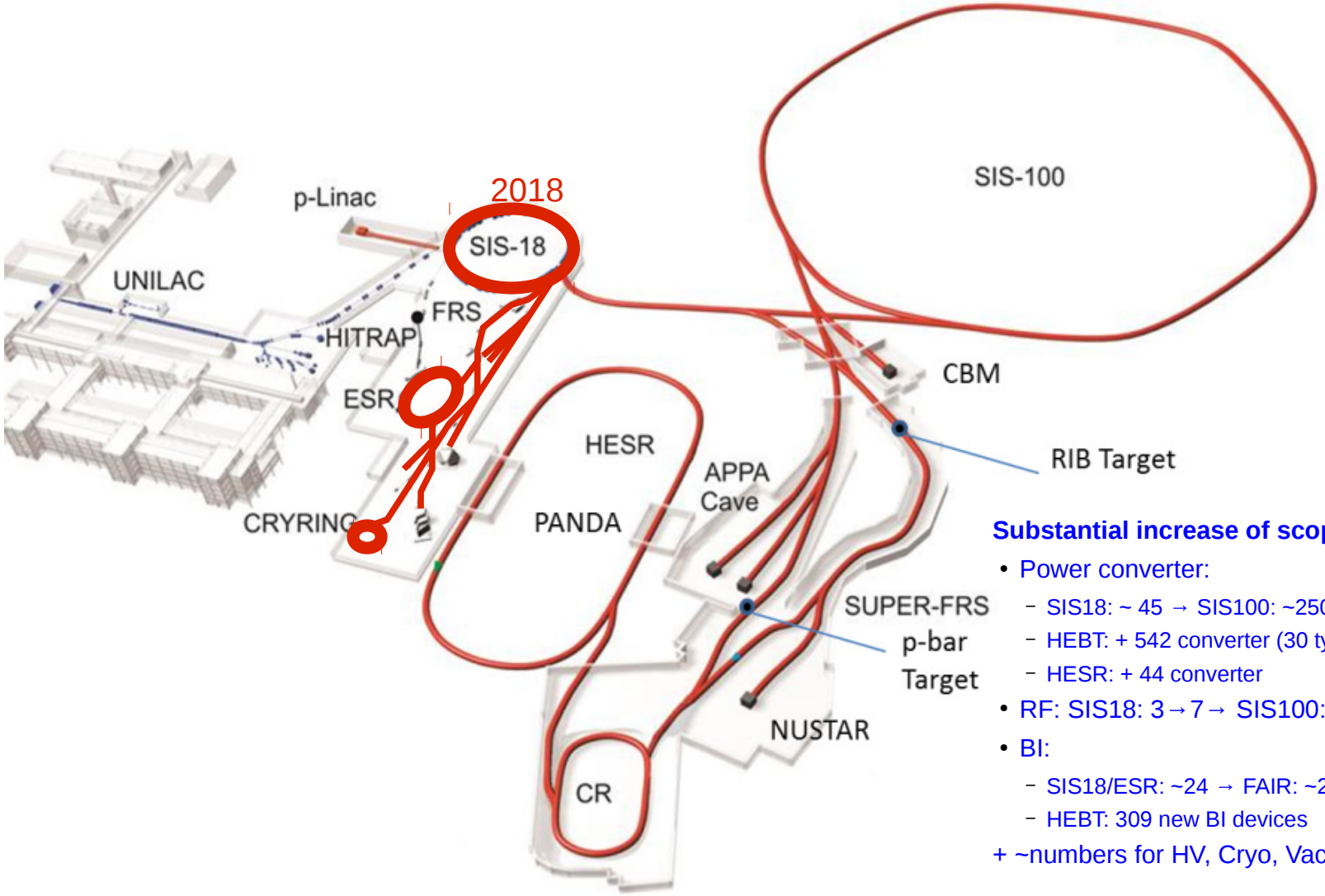


image courtesy: http://softwaretestingfundamentals.com/

Accelerator Controls Retrofitting
GSI → FAIR Transition in 2018



Substantial increase of scope w.r.t. GSI:

- Power converter:
  - SIS18: ~ 45 → SIS100: ~250 converter
  - HEBT: + 542 converter (30 types!)
  - HESR: + 44 converter
- RF: SIS18: 3→7→ SIS100: 38 RF sys.
- BI:
  - SIS18/ESR: ~24 → FAIR: ~250 BPMs
  - HEBT: 309 new BI devices

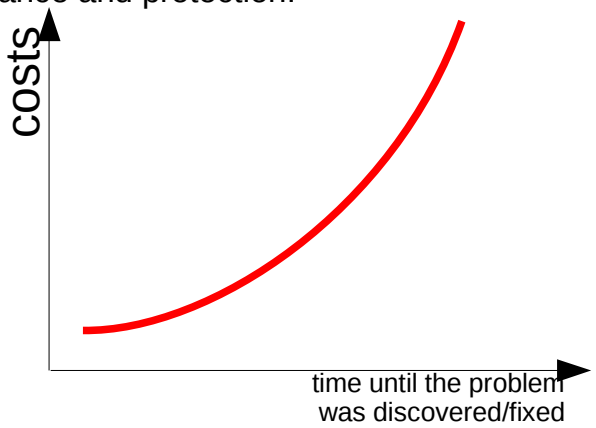
+ ~numbers for HV, Cryo, Vacuum, TI, ...

Poka-Yoke (ポカヨケ) – 'Mistake-Proofing'
Reaction-Time and Cost → “fix” errors early

… minimise common mistakes, procedural errors, etc. affecting machine performance and protection.

→ Fix problems early, when and where they occur:

- Minimises procrastination of errors
- “Safety starts with safe habits”!



- Benefits of early indicators of developing/not-yet-critical faults:
  - Identification/isolation of faults during HWC/DryRuns (e.g. via Sequencer)
    fixes “domino effect” problems at the source not its symptoms:
    - HWC: parallel testing possible → resource control by SPL/equipment group
    - BC: predominantly sequential testing → limited/no resource control ↔ “fire-fighting”
  - Preventative maintenance
    - early indication of drifting parameters (e.g. slow thermal runaway of IGBTs)
    - group common/similar maintenance tasks rather planning for having react (minimises “fire fighting”)
  - Post-Mortem analysis ('as good as new' SIL assurance)
    - insures safety-integrity-level, minimises risk at the source before they escalate
    - minimises human-intervention – OP/expert has to handle errors that have not/cannot be automatically analysed by the post-mortem system

FAIR Commissioning Procedures

- Procedures verify that functional and ‘Detailed Specification’ requirements have been fulfilled → FAT, SAT but also importantly for Beam Operation
  - checks MoU between various stake-holders (AP, BI, CO, RF, ...)
    - *define when, where and how the individual accelerator systems should fit in*
  - identify loop-holes/missing critical functionality
  - identification/isolation of errors & regression testing
    - *ie. minimises that errors fixed in the past are not being reintroduced*
  - ensures that HW is largely/fully commissioned prior to BC/operation
    - *N.B. HWC (testing in ||) vs. BC (testing predominantly sequentially)*
  - (semi-)automation: better to ‘act’ than to ‘react’ → triangle of *‘work to be done’ vs. ‘required project timeline’ vs. ‘available resources’*
  - traceability and documentation (for your peers, SPL/management, operation, …)
    - improve the quality of the project (QA)

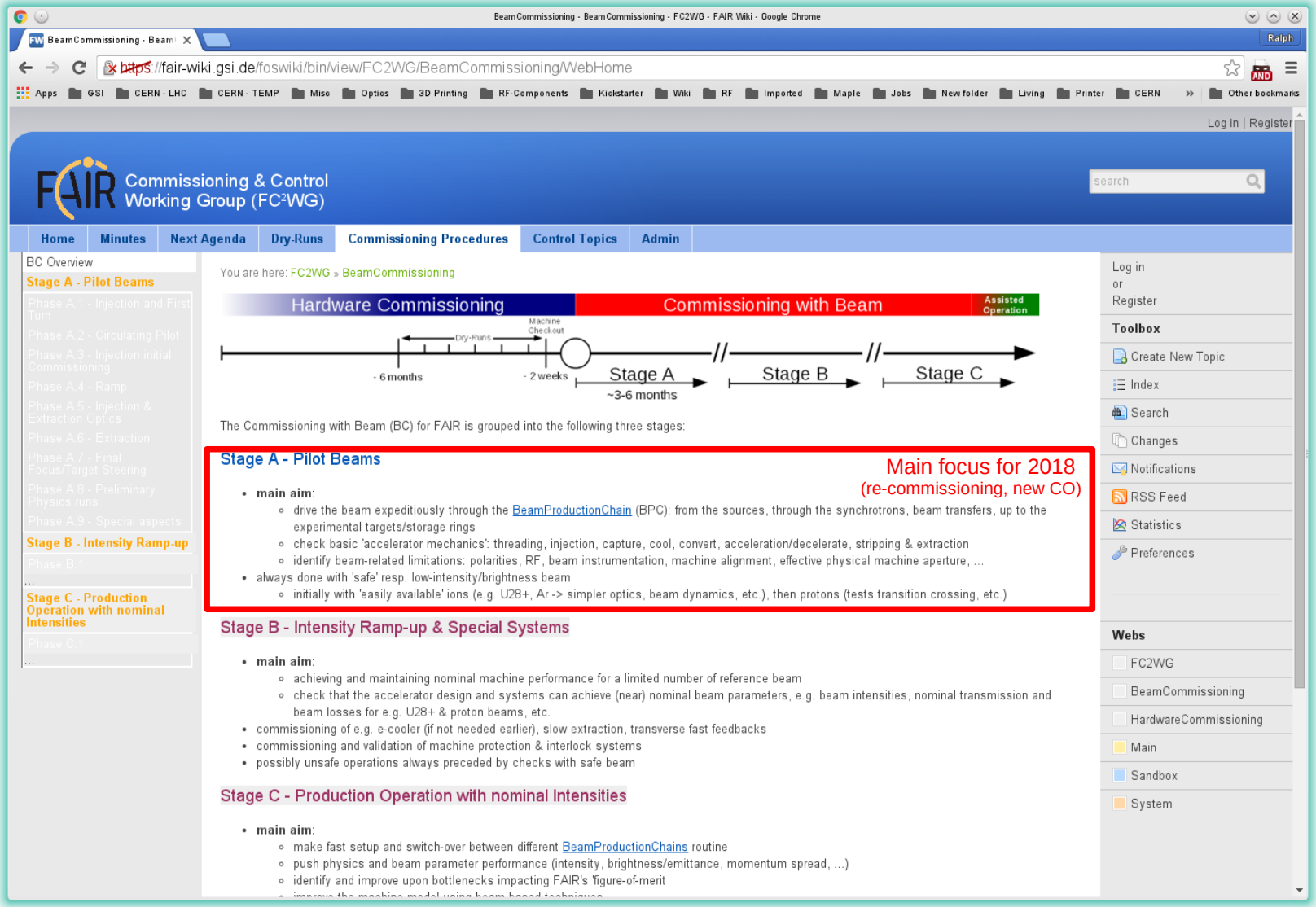
- Test procedures describe the intention of what needs to be achieved
  - specifically not implementation or tool specific ie. implementation can be refactored without the need to change the intention of the test
  - terminology/hierarchy of: unit testing → integration testing → acceptance testing

Proposed Procedure Structure

- Description
  - test procedures describe the intention of what needs to be achieved
  - specifically not implementation or tool specific
- Entry Conditions
  - dependencies to other systems/conditions → reference to other test sequences
- Machine Setup
  - conditions to be checked beforehand
- Procedure – list of individual tasks
  - for each detailed testing step:
    - short-hand key for each step (e.g. ‘step A’, naming taken from DS document)
    - short description of what needs to be achieved (ie. a ‘one liner’)
    - acceptance criteria (either from DS or split into warning/error)
- Problems
  - be free to comment on (technical) concerns or potential issues
- Exit conditions
- Open Questions & Action Items
- References & Acronyms
  - where applicable, helps you with further QA documentation

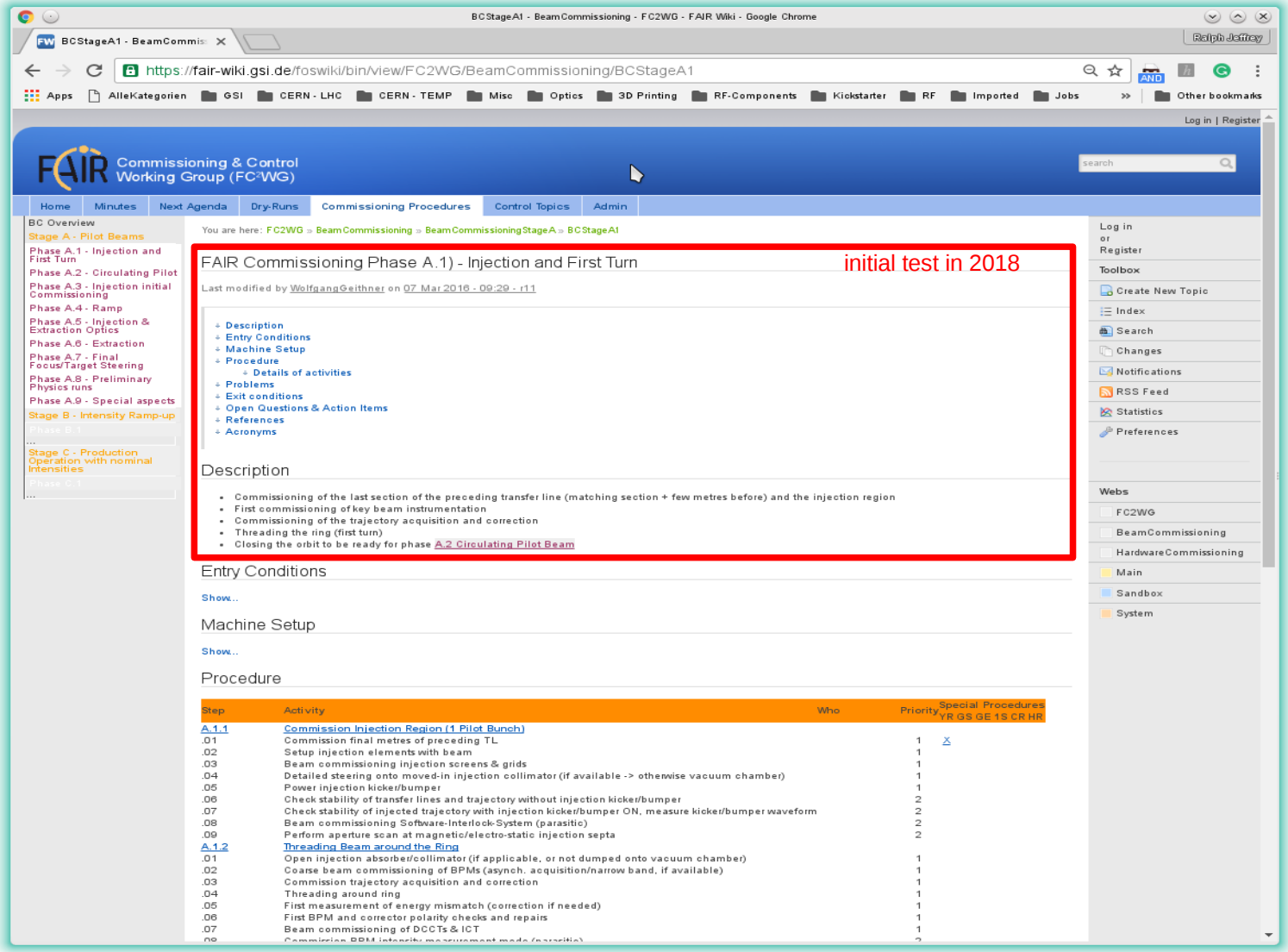
Example: FAIR Commissioning Procedures

https://fair-wiki.gsi.de/FC2WG/BeamCommissioning



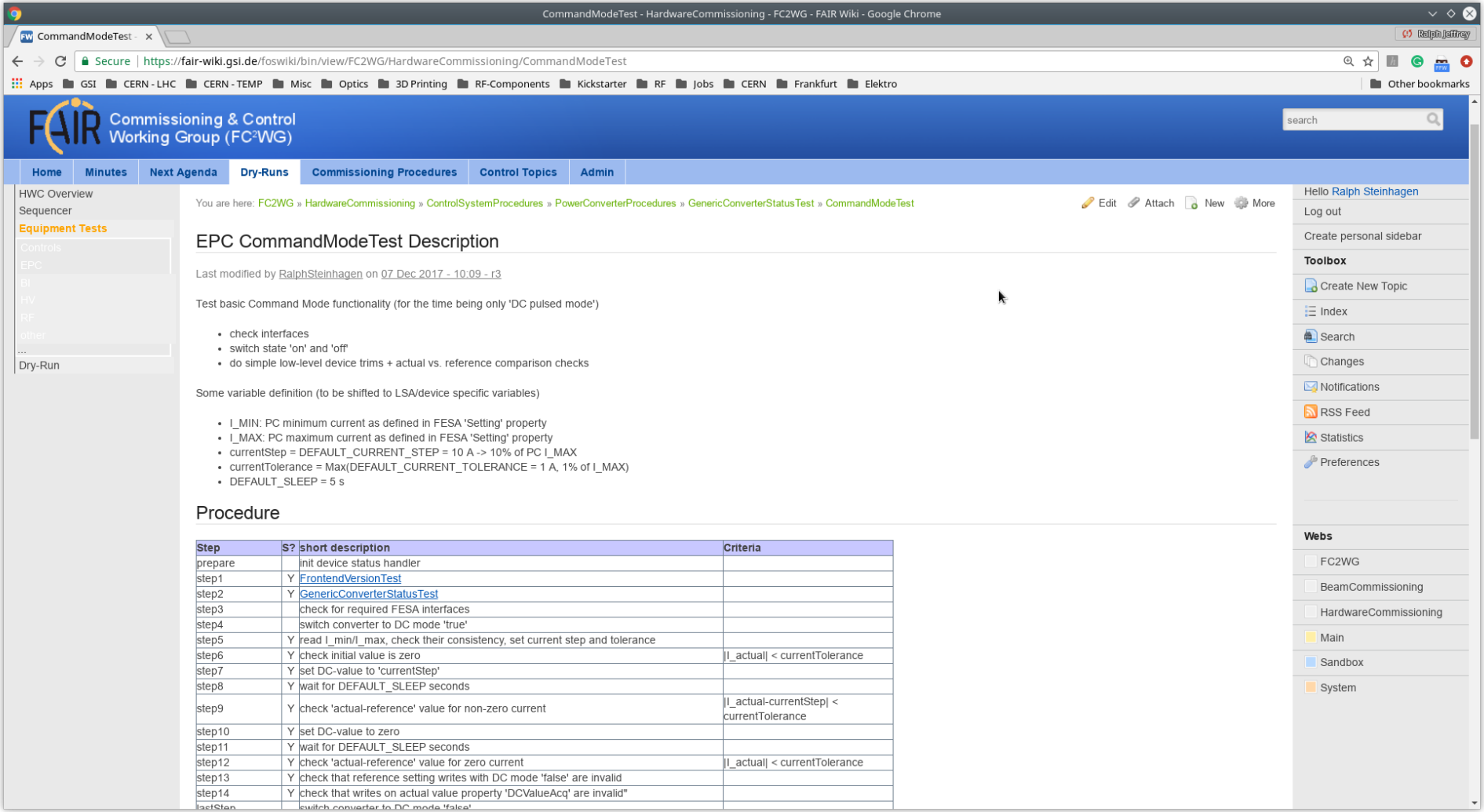
Example: FAIR Commissioning Procedures

https://fair-wiki.gsi.de/FC2WG/BeamCommissioning



Example: FAIR Dry-Run Commissioning

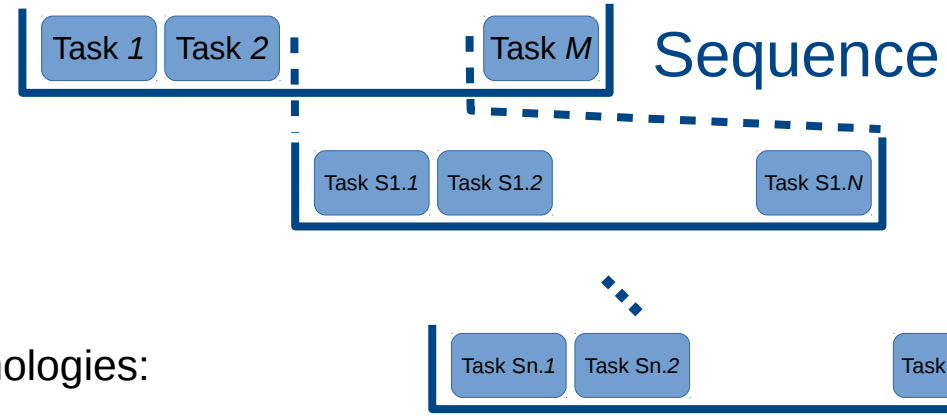
https://fair-wiki.gsi.de/FC2WG/HardwareCommissioning/



Sequences & Tasks I/III – Implementation

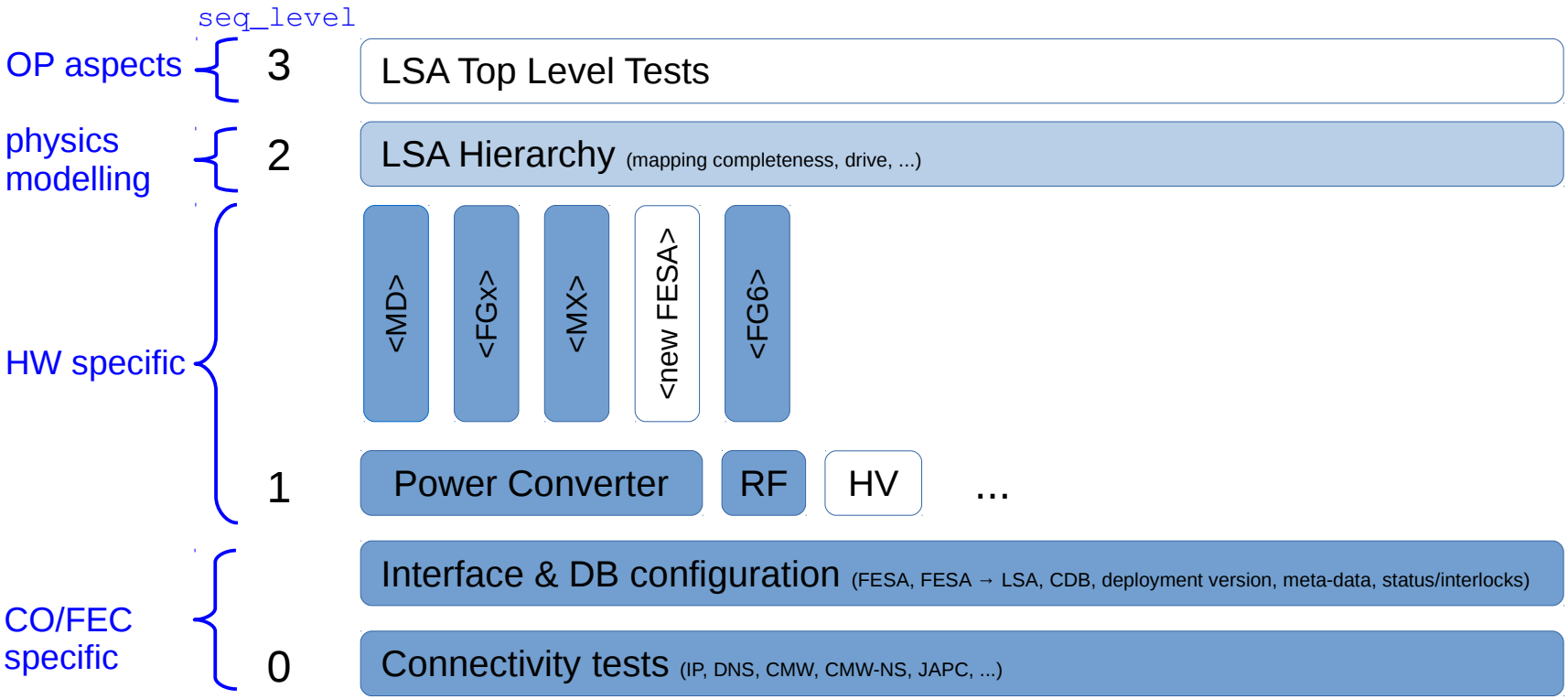
- ‘Task’ = device class specific atomic test, e.g.
  - connectivity test, power ‘on’, power ‘off’, ...
  - actual vs. reference comparison, ...
- ‘Tasks’ can be assembled to ‘Sequences’ …

  ...which may also contain further sub-sequences:



- CO backbone technologies:
  - FAIR Archiving Systems → Documentation
  - LSA-based Settings Management → Reference & Data Supply
  - System- and Site-wide Digitisation of Analog Signals→ ‘actual vs. reference’ process monitoring

Some Preliminary Test Hierarchy



- Some logstash meta-data keys (see: https://logstash.acc.gsi.de/):
  - Existing tags: `program`: 'sequencer', `user_name`, `pid`, ...
  - `seq_device`: e.g. device name, LSA property name, global function
  - `seq_level: <0 … 4>, seq_task`: <task/class name>, `seq_sequence`: <collection of tasks> (???)
  - `seq_testID`: unique identifier for given sequencer run (↔ multi-user, parallelism)
  - seq_test_start: <time-stamp>

Sequences & Tasks II/III – Implementation

- What is provided by the sequencer frame-work:

```
abstract class AbstractSequenceImpl extends AbstractTaskImpl implements Sequence {
     void exec() {
          execPreUserCode(); // init communication to Archiving, LSA, Sequencer Service etc.
          execUserCode();
          execPostUserCode(); // reporting, etc.
     }
}
```

- Level 1 tests (provided by the CO/equip. Group/machine experts):

```
class FrontEndGenericConnectivityTest extends AbstractSequenceImpl implements Sequence {
     // [..]
     void execUserCode() {
          addTask(new Task("prepare", "init device status handler", this, () -> {
               deviceConnection = new GenericStatusTest(deviceName);
          }));

          addTask(new SkippableTask("step1", "check simple IP connectivity", this, () -> {
               if (deviceConnection.isHostReachable(CONNECTIVITY_TEST_IP) == false) {
                         throw new RawSequenceException("testCMW3get() - failed");
               }
          }));
          // [..]
     }
     // [..]
}
```

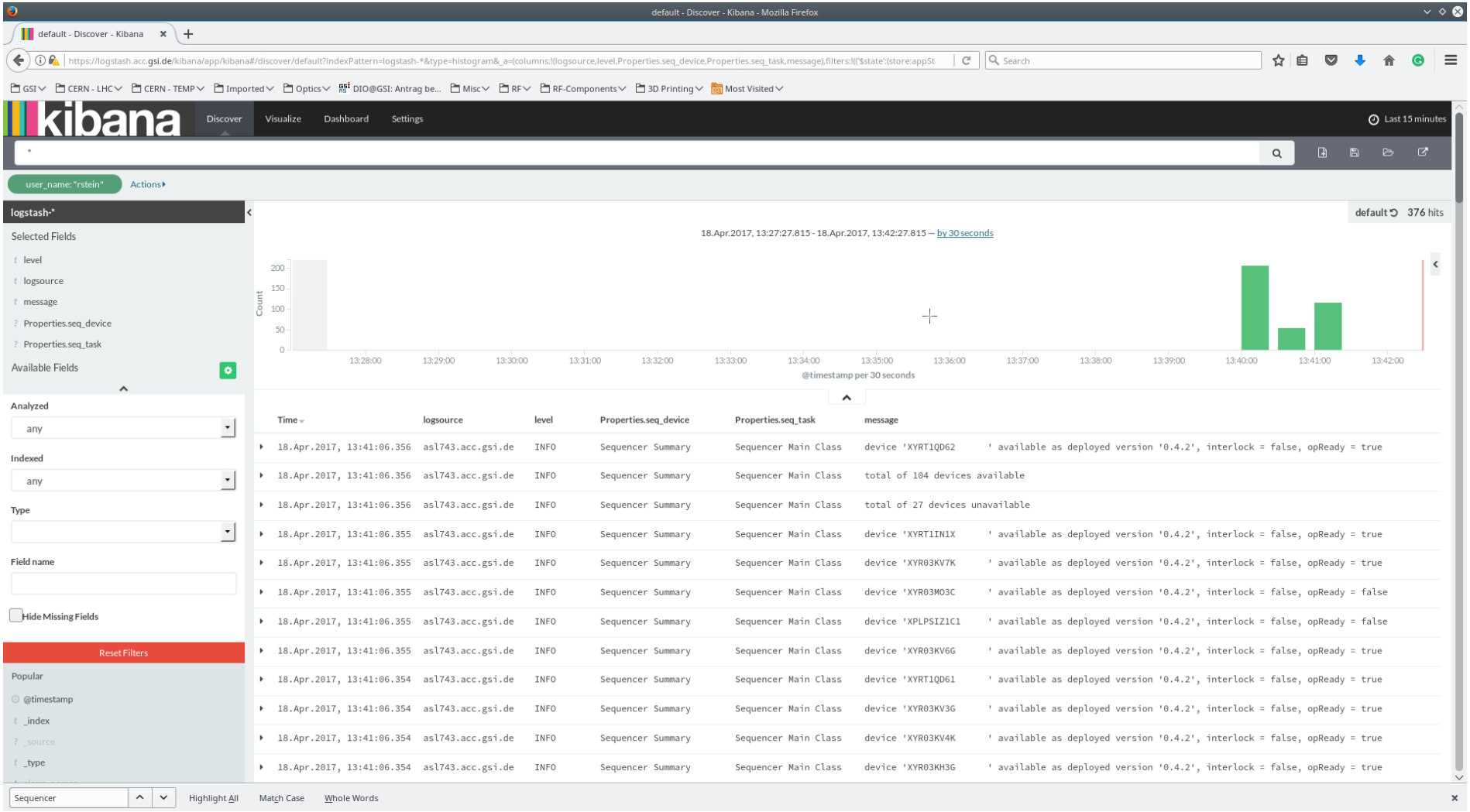
Sequences & Tasks III/III – Implementation
– what the user needs to implement

```
class HwcTest1 extends AbstractSequenceImpl implements Sequence {
       void execUserCode() {
            addTask(new FrontEndGenericConnectivityTest(getParameter()));

            addTask(new Task("step1", "description #1", seqReference, () -> {
                 task1();  // user/device-specific non-skippable atomic test operation 1
            }));

            addTask(new SkippableTask("step2", "description #2", seqReference, () -> {
                 task2();  // user/device-specific skippable atomic test operation 2
            }));
            [..]
       }

      void task1() {
            // test SAT-A sub-procedure x.1, see specification... item ...
            // [..], may throw Exception (or derivatives)
      }

      void task2() {
            // test SAT-A sub-procedure x.2, see specification... item ...
            // [..], may throw Exception (or derivatives)
      }
}
```

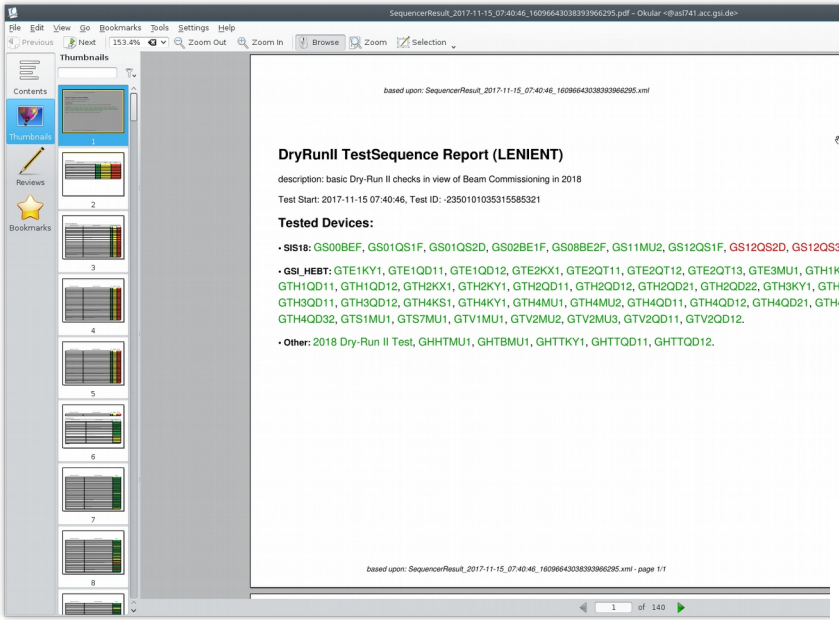
Some examples to get a flavour of the targeted code style and flavour:
https://www-acc.gsi.de/svn/applications/app-codesnippets/
https://www-acc.gsi.de/svn/applications/sequencer/

Sequencer Documentation Example I/III

https://logstash.acc.gsi.de/



Sequencer Documentation Example II/III
automatic PDF generation (→ SAT)



Tested Sequences:

Sequence Name	OK	Skipped	FAILED
BasicLSATest	1	0	0
CommandModeTest	45	0	0
DryRunII_TestSequence	53	0	0
FecConnectivityTest	52	0	0
FecStandardTest	52	0	0
FecStatusTest	1	0	0
FecVersionTest	7	0	0
GenericConverterStatusTest	23	0	0
MiniMaspTest	1	0	0

Detailed Test Sequences:

Sequence:Task Name:	detailed description	OK	Skipped	FAILED
DryRunII_TestSequence:DryRunII_TestSequence	basic Dry-Run II checks in view of Beam Commissioning in 2018	1	0	0
MiniMaspTest:prepare	init device status handler	1	0	0
BasicLSATest:prepare	init device status handler	1	0	0
DryRunII_TestSequence:prepare	prepare global dry-run sequence	1	0	0
DryRunII_TestSequence:BasicLSATest	checks basic LSA functionalities and interfaces	1	0	0
DryRunII_TestSequence:MiniMaspTest	checks basic MASP functionalities and interfaces	1	0	0
MiniMaspTest:check1	test availability and consistency of all_status property	1	0	0
BasicLSATest:check1	check if 'SIS18' 'B ho' is defined and retrievable	1	0	0
BasicLSATest:check2	check if 'CRYRING' 'B ho' is defined and retrievable	1	0	0
MiniMaspTest:check2	test consistency of MASP POWER_ON with device status	0	1	0
MiniMaspTest:check3	test consistency of MASP INTERLCK with device status	0	1	0
MiniMaspTest:check4	test consistency of MASP MOD_RDY with device status	0	1	0

Sequence:Task Name:	detailed description	Result:
- step3	check availability/conformity deployed versions	FINISHED
- step4	check LSA vs deployed versions	FINISHED
FecStandardTest:FecStatusTest	simple FESA status property test	FINISHED
FecStatusTest:prepare	init device status handler	FINISHED
- check1	test availability/consistency of FESA Status property	FINISHED
- check2	report on pending device warning and error messages	FINISHED
- check3	check whether device is in DC mode	FINISHED
- check4	check whether Status->Status is 'OK'	FINISHED
- check5	check device power state	SKIPPED_FAULTY
- check6	check device being in remote state	SKIPPED_FAULTY
- check7	check device being in (SW) interlock	FINISHED
- check8	check device being in OPready mode	FINISHED
- check9	check whether device modules are ready	SKIPPED_FAULTY

GS12QS2D:

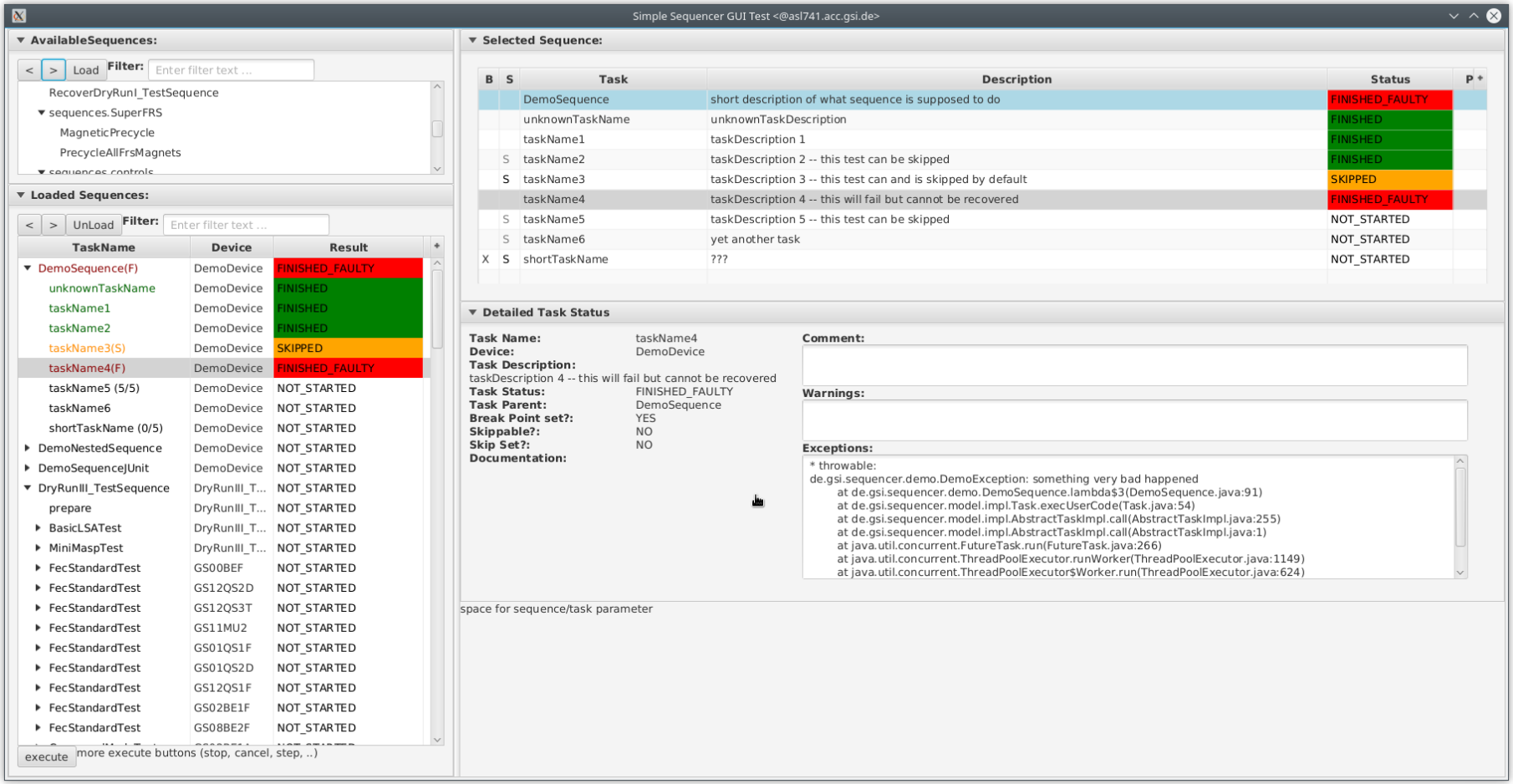
Sequence:Task Name:	detailed description	Result:
DryRunII_TestSequence:FecStandardTest	standard FEC test: connectivity -> version vs. LSA -> status	FINISHED_FAULTY
FecStandardTest:FecConnectivityTest	low-level front-end communication and status tests	FINISHED_FAULTY
FecConnectivityTest:prepare	init device status handler	FINISHED
- step1	check simple IP connectivity	FINISHED
- step2	check availability of RDA nameserver	FINISHED
- step3	check low-level CMW3 'get'	FINISHED_FAULTY

GS12QS3T:

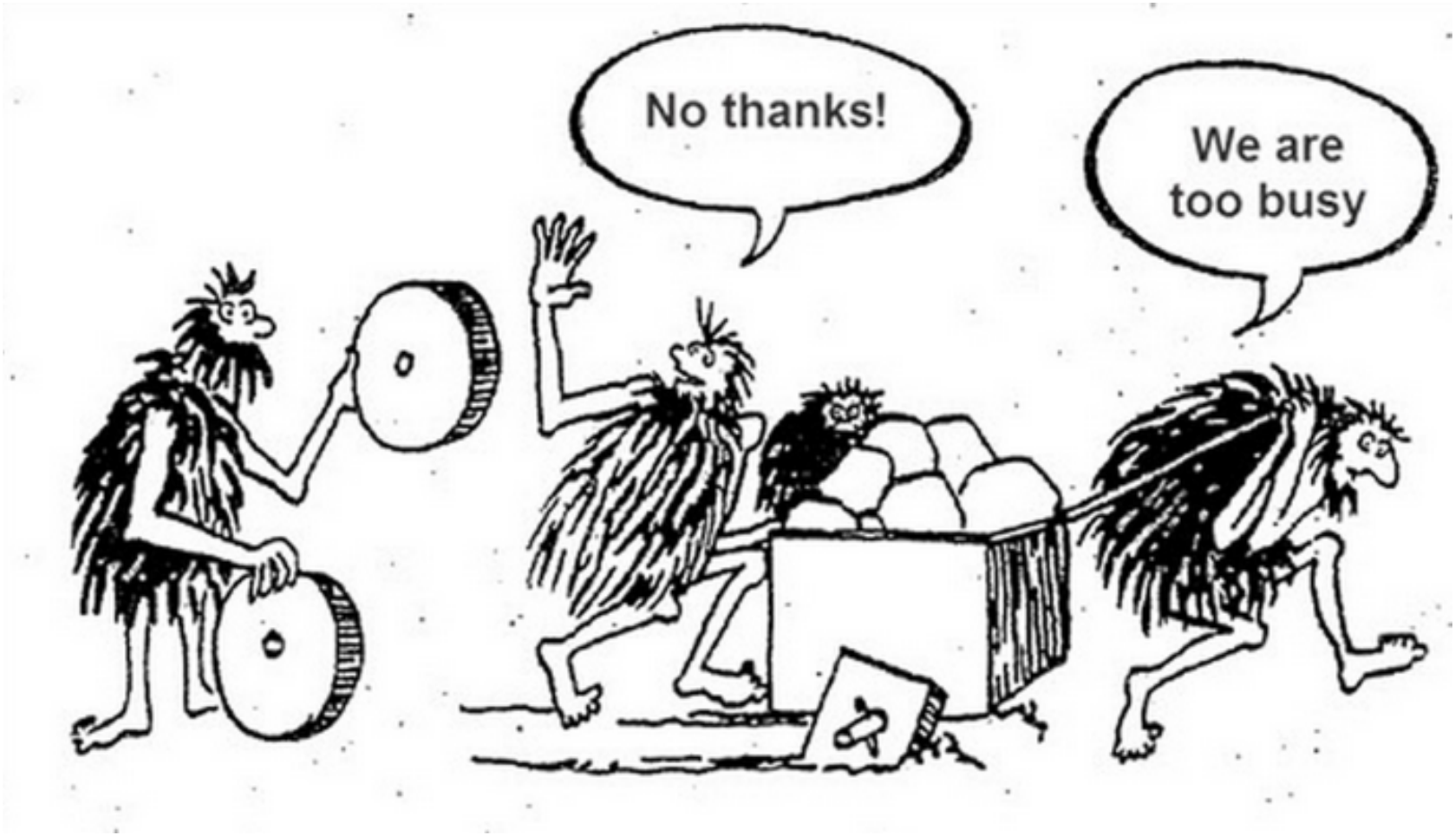
Sequencer Documentation Example III/III via Archiving System

- N.B. no nice GUI (yet) but report similar to previous...

FAIR Sequencer GUI Prototype



Reinventing the Wheel ...



FAQ & Frequently Used Interfaces I/II

- … check-out the demos first
- … check-out the existing sequences/tasks
- KISS: – keep it simple and safe & document
  - think about what needs to achieved first
  - keep the code readable (→ for your colleagues & yourself)
  - stick to Java (~ C++) coding styles
    - PMD and ‘Find Bugs’ Eclipse plugins are your friends!
  - stick to CO coding-style guide-lines!
- plan-ahead: Java Experts (<5%) vs. Non-Java Experts (the other 95% at GSI/FAIR)
  - 1. document sequence → code
  - 2. code → document sequence

FAQ & Frequently Used Interfaces II/II

- Common Task Interfaces:

```
 // example of the most commonly used/useful setter methods

final TaskUserInterface dummyCode = (taskReference) -> {

      // my dummy user-level code code

};

final CallableTask myTask = addTask(new Task()).setTaskName("shortTaskName").setTaskDescription("???")
      .setSequence(this).setSkippableState(true).setSkipTaskState(true).setTaskBreakPoint(true)

      .setTaskRepitionNumber(5).addReference("task info reference, e.g. URL")

      .addUpdateListener((updateEvent) -> {

      // call me if you want to be notified about task execution state update

      // (usually only used within Sequencer service)

      }).setUserCode(dummyCode).setErrorRecoveryUserCode(dummyCode).setPreUserCode(dummyCode)

      .setPostUserCode(dummyCode);

      myTask.isSkippable(); // getters mirror usually setters etc.

[..]
```