Improving feedback loop for automated testing

Presented by NOKIA

10th UCAAT - User Conference on Advanced Automated Testing

15/11/2023

Agenda

- Testing flow evolution
- Testing approach for complex software
- Tools and applied mechanics
- Key takeaways

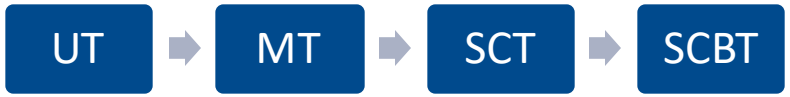
Testing flow evolution – test levels

- Standard approach – split test levels
- More granularity – additional split between system component tests (SCT) and plane integration tests (PIT)

Testing flow evolution

- It is simplified evolution version of flow that is currently used in our team
- Testing flow has been constantly evolving across Nokia products and multiple different tools
- We are responsible for last software-only integration testing

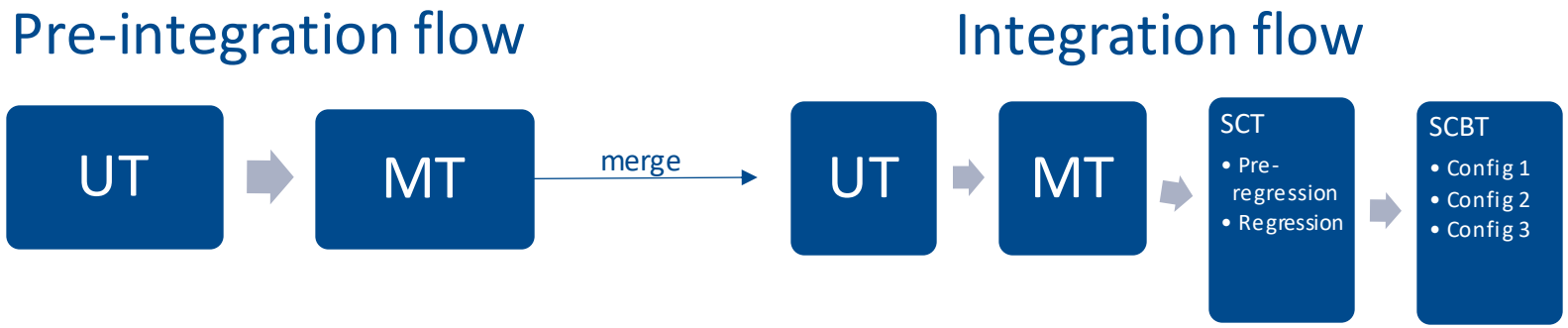
Starting flow



- All steps could be executed on local environment
- There is no pre-integration flow, all changes were merged directly on SVN delivery branches
- Unit Tests (UT) and Module Tests (MT) are managed by development domains, there is single executable
- System Component Tests (SCT) are executed in sequence
- System Component Build Test (SCBT) are first level that uses Hardware (HW), each test is performed manually on selected HW configuration

Testing flow evolution – step 1

- First product (WBTS)
- Testing tool based on Eclipse (Topik) with XML-based Message flow testing
- SVN+Teamcity

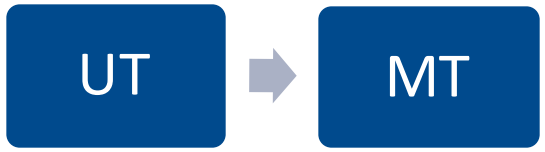


- Automatic execution of UT and MT was added for user branches, so merging is possible
- SCT regression grow quite big, but tests are still executed in sequence, so additional split was done
- Pre-regression were set of basic tests, to detect earlier global failures and it blocked full regression
- SCBT have 2 more configurations, but they still need manual work (each can be done separately)

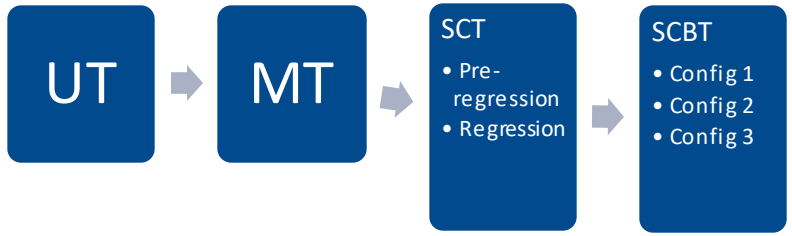
Testing flow evolution – step 2

- New product (Megazone) and new tools (Pegasus, evolution of Topik), gitlab + Jenkins
- Some tests migrated, some written from start

Pre-integration flow



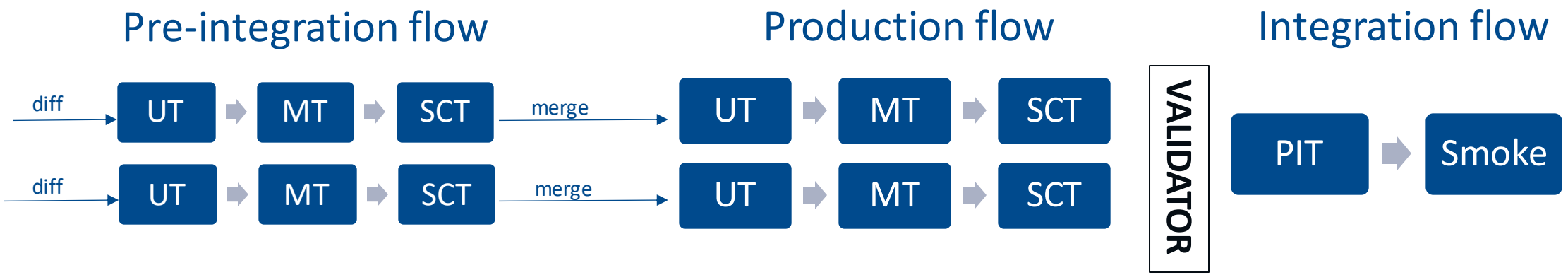
Integration flow



- SCT regression reduced, and it was possible to execute tests in parallel on continuous integration (locally still sequence)
- SCBT still done, but some of HW management operations were automated (like power control, sensors, log collection)

Testing flow evolution – step 3

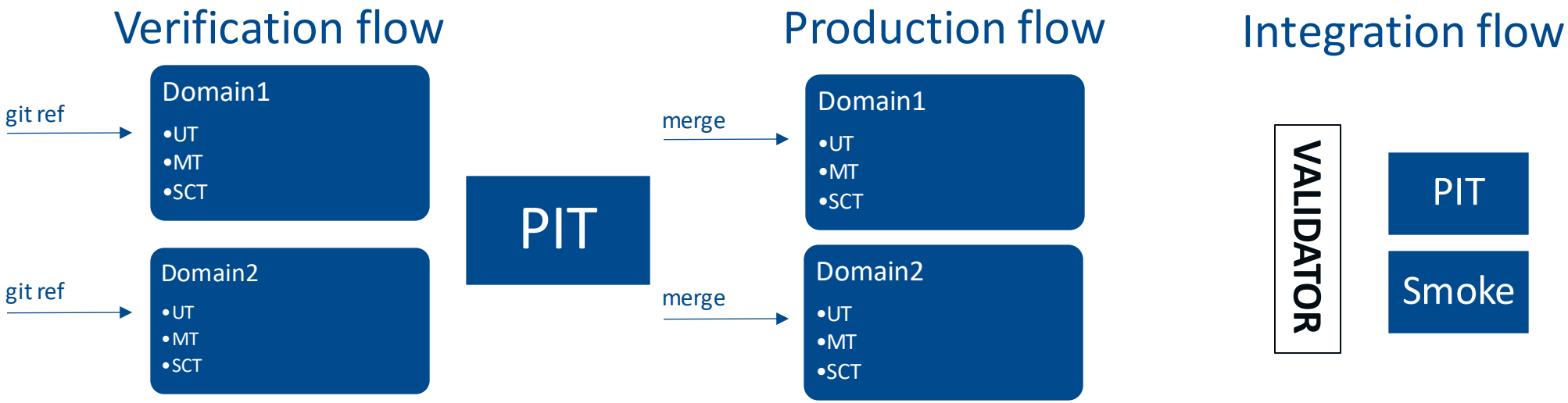
- New product (SRAN) and new tools: K3 with TTCN3, SVN+Jenkins
- Code split – multiple executables added with separate pre-integration flows
- As a result, SCT moved to development responsibility, new level of integration tests added – Plane Integration Testing (PIT)
- Pre-integration flow executed on Nokia tool (ReviewBoard) on uploaded SVN diffs



- Validator added as a collection of SVN revisions of each domain after they pass Production
- SCBT replaced by automated Smoke tests

Testing flow evolution – step 4

- Switch to git/gerrit
- Multiple improvements in PIT



- Tests for each domain started in parallel
- PIT execution added for all integration flows, it triggers only after each domain pipeline pass
- Smoke and PIT started in parallel, but also due to limited resources Smoke started every few builds

Testing flow evolution – PIT improvements

- Introduction of ZUUL for multi-domain PIT execution and auto-merging
- Test tags to reduce executed regression set
- Toggling tests handling
  - Re-run mechanism and pass ratio
  - Unstable tests passer script
- Longer tests with priority
- Load and macro balancer to utilize cloud resources
- External storage for log collection to save Jenkins resources
- Packing common build elements to save download time
- Adding static code analysis tools to speedup review time
- Extracting common part of each TC as separate process triggered every 2 weeks
- Regular "Regression reduction actions" to merge similar tests
- Multiple tools to speed-up failed case analysis

Testing approach for complex software – test synergy

- Test synergy achieved by extensive planning
  - Maximum coverage with minimum resources
- Minimise test level overlap
  - Test coverage split between test levels
  - Early phase estimations and co-planning with testers and developers
- Testplan live review
  - Developers from affected domains take part
  - Tester’s POV versus Developer's POV

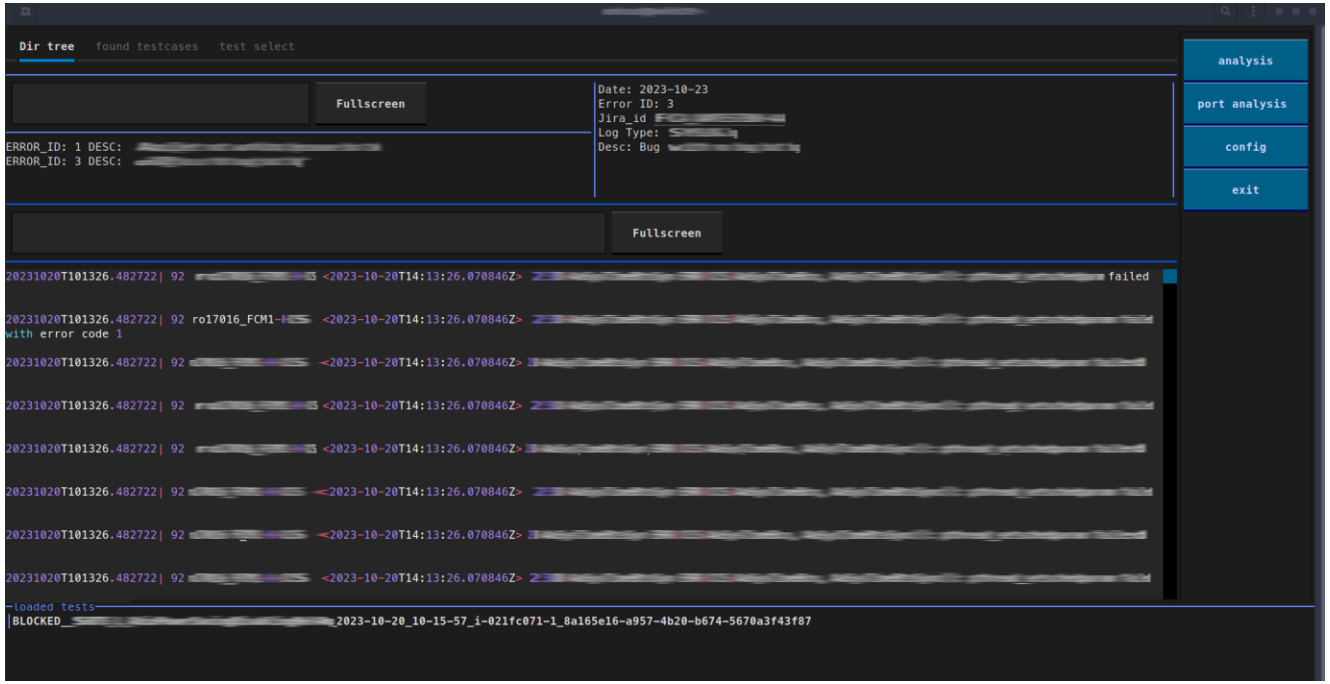
Testing approach for complex software – regression test suite

- Don’t run unneccessary tests!
  - 2000 automated test scenarios
  - Multiple SW components with complex relations
- Automatic test selection mechanisms
  - Hardware/technology/specification-based
    - Exclude not impacted HW configurations
    - Exclude not impacted parameter configurations
    - Other categories
- Code-coverage-based
  - Live-track code coverage
  - Map each SW component onto set of scenarios

Tools and applied mechanics

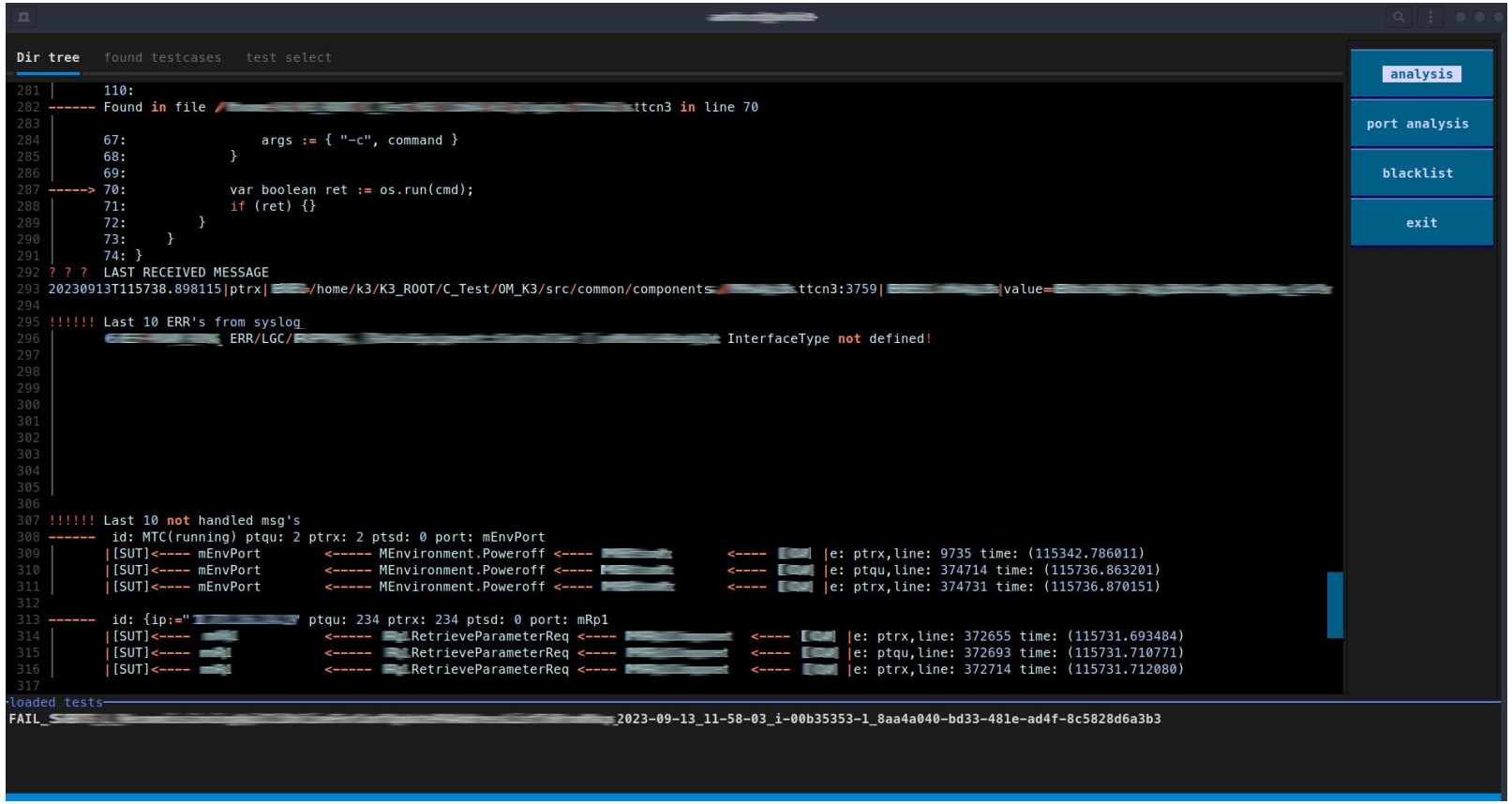
- **Logan** is a tool designed to help analyse K3 logs from tests, based on configuration file defined by testers.

```
root@l:~/...$ logan k3log     path_to_log                    performance messages
Time format: [H]H:MM:SS[.UUUUUU]
Percent Sum Average Median nCalls <Min,Max> Name
26.4154% 0:34:23.308649 0:02:08.956790 0:02:08.979582 16 <0:02:08.650618,0:02:09.159795> Message 1
11.8096% 0:15:22.444281 0:01:42.493809 0:01:30.011393 9 <0:00:04.022531,0:02:49.879414> Message 2
11.5585% 0:15:02.832183 0:02:08.976026 0:02:00.172095 7 <0:00:00.039938,0:05:00.798319> ...
6.9492% 0:09:02.803739 0:01:48.560747 0:01:45.373469 5 <0:00:01.259754,0:03:45.423288> ...
6.9328% 0:09:01.520992 0:02:15.380248 0:02:15.377737 4 <0:00:45.347751,0:03:45.417767> Message n
```



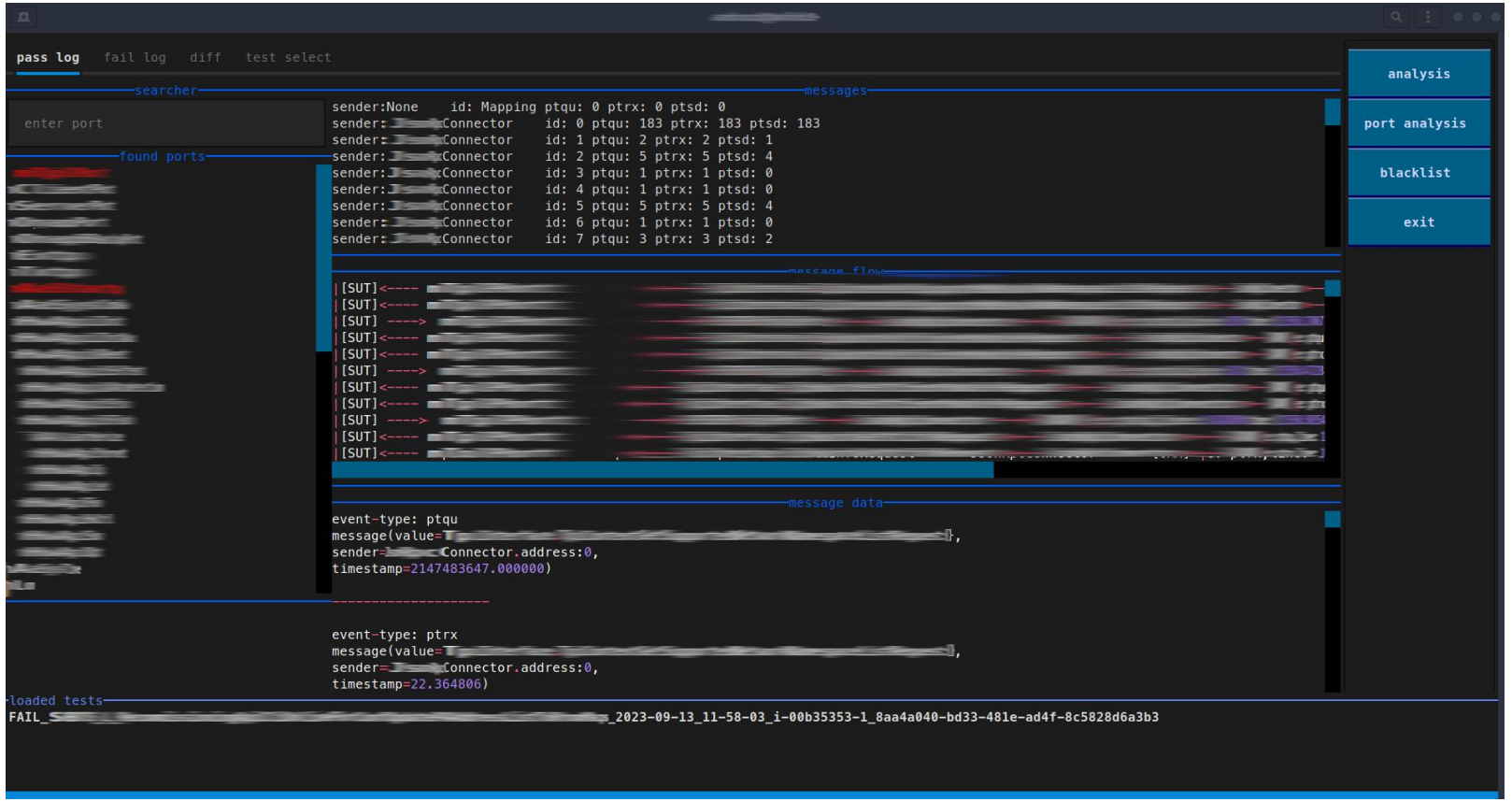
Tools and applied mechanics

- **Logan** is a tool designed to help analyse K3 logs from tests, based on configuration file defined by testers.



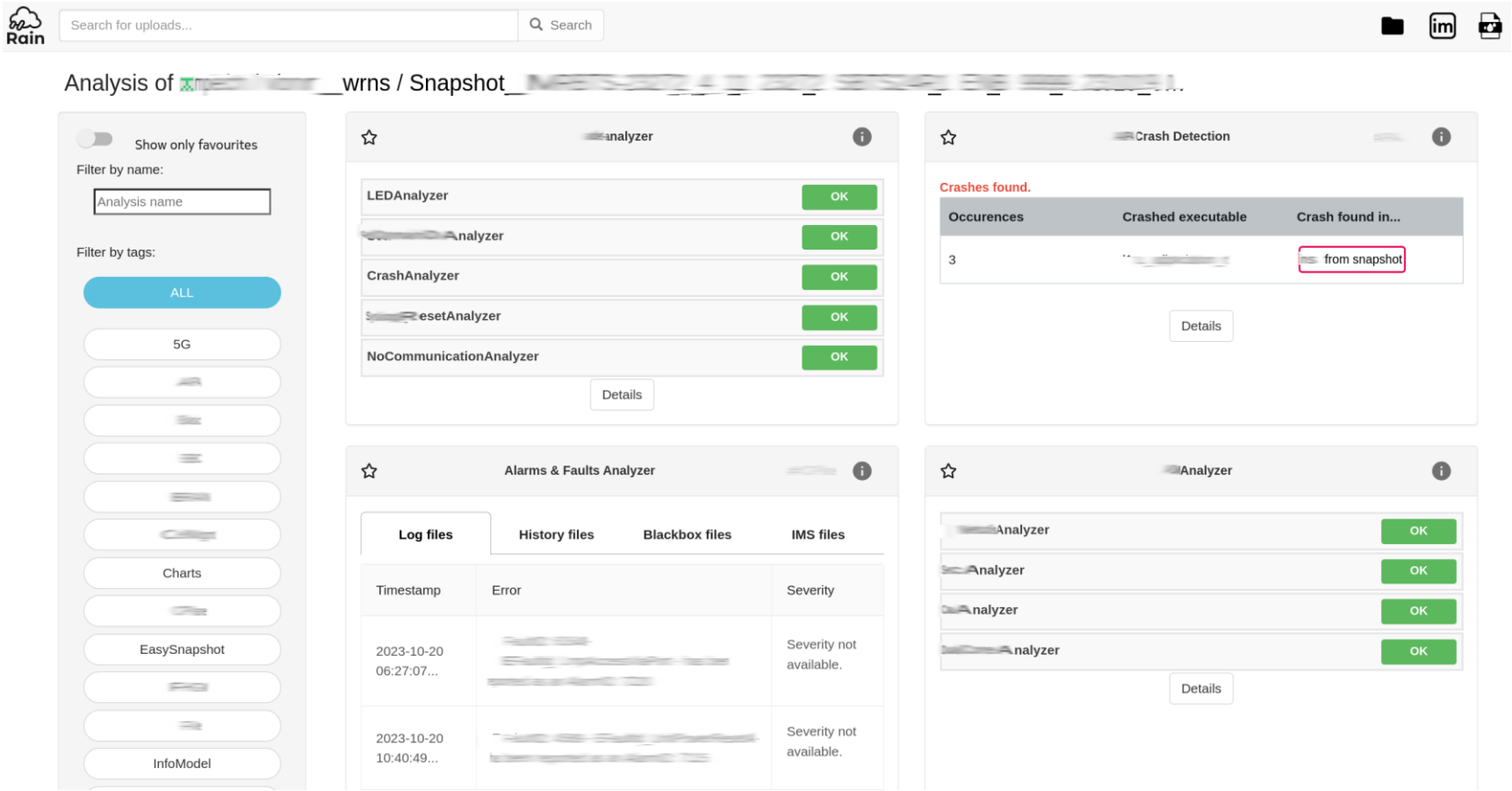
Tools and applied mechanics

- **Logan** is a tool designed to help analyse K3 logs from tests, based on configuration file defined by testers.



Tools and applied mechanics

- **Rain** is a log collaboration and analysis platform. Research on Rain evolves toward BRain which is Machine Learning solution focusing on removing noisy data, cluster log events, finding patterns and detection of anomalies.



Tools and applied mechanics

- **Rain** is a log collaboration and analysis platform. Research on Rain evolves toward BRain which is Machine Learning solution focusing on removing noisy data, cluster log events, finding patterns and detection of anomalies.

Tools and applied mechanics

- **RETEST** tool is automated Jenkins job with parametrized template which allows to execute group of tests on selected range of Software builds. That provides fast feedback which SW build introduced instability and should be reverted.

Pipeline PITs retest CLASSICAL

This build requires parameters:

USER_GIT_REF

master

TESTCASE_REGEX

TESTCASE_MULTIPLIER

Enable_STEP_1

Check to not skip '1__' step

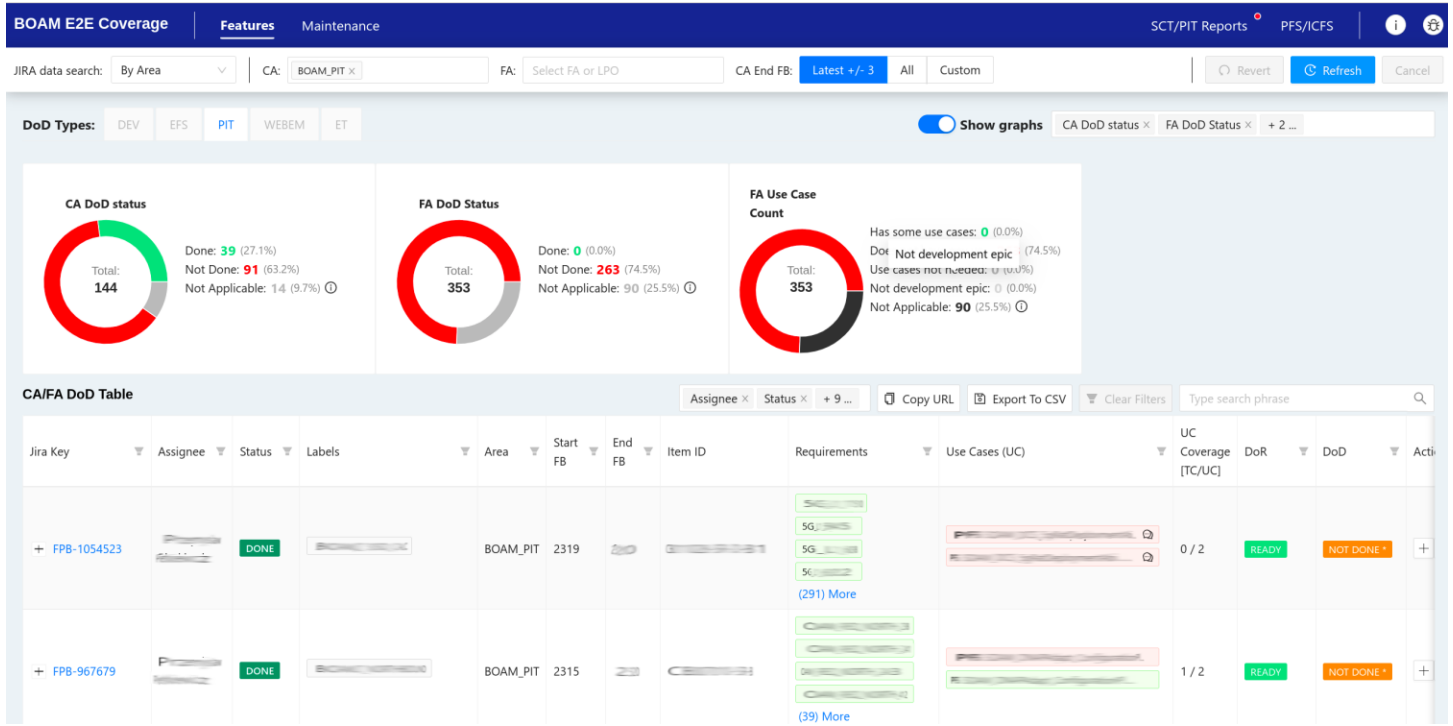
VALIDATOR_URL_OR_OAM_TAG__1

C_TEST_HASH__1

KNIVES__1

Tools and applied mechanics

- **DoD** tool is End-to-End Coverage Tool (DoD checker) monitors requirement-based software development and testing for all product functions in the JIRA backlog.



Key takeaways

- Documentation first!
- When you allow bug to be deliver, you will face with instabilities.
- Less is more.
- It's all at your fingertips.
- Double, double checking.

Any further questions?

Tomasz Kowalczyk
tomasz.kowalczyk@nokia.com

Bartosz Hajduk
bartosz.hajduk@nokia.com

Mariusz Lont
mariusz.lont@nokia.com

Piotr Czermak
piotr.czermak@nokia.com