

ARM Validation Dashboard

Zygmunt Krynicki
Linaro Infrastructure Team
Canonical

Agenda

- Space race!
- Goals of the project
- Design considerations
- Supported features
- Q&A

What is Linaro about?



<http://endtimepilgrim.org/earthrise.jpg>

beating the **reds** to the moon!

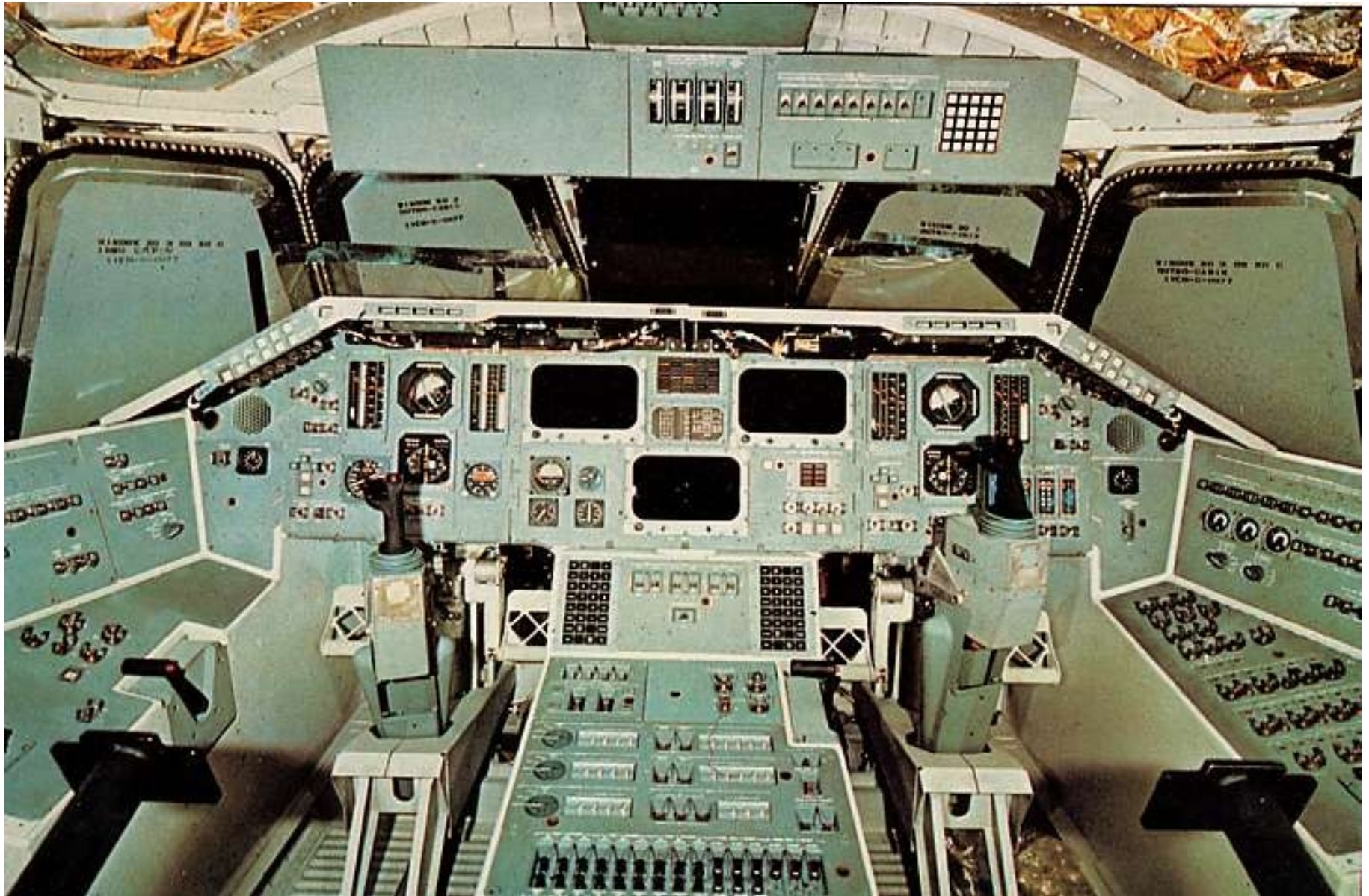
What is the dashboard?



<http://www.flickr.com/photos/arcticpuppy/4076315457/>

...no

This is the space race!



<http://www.buran.fr/bourane-buran/img/STS-dashboard-grand.jpg>

Dashboard helps to deliver
quality-related data to the
engineers



http://commons.wikimedia.org/wiki/File:Engineers_Working_in_the_Launch_Control_Center_Preparing_for_the_Launch_of_Apollo_11.gif

Dashboard is:

- New software created for Linaro
 - Designed with SoC vendor requirements in mind
- Front-end to a much larger system
- Place where data the meets the people
- Tool for making decisions easier
 - Can we ship this?
 - Is it better than last time?

What do we want to deliver?

Goals

- Integrate QA data from various sources
 - Image builds
 - Automatic testing and benchmarking
 - Bugs reports
- Build the required infrastructure components
- Present this data in helpful ways
 - Graphical
 - Easy to understand
 - Help to answer common questions

Start with something small

- This is my first spec
 - From my first UDS
- It's better to start with something small
- Small is usually simple
- Besides
 - It's clear what you want...
 - ...after you start using what you have

Features

- Handle testing and benchmarking
 - Primary feature this cycle
- Show useful charts, tables and other views
- Secondary features
 - Tracking package changes
 - Tracking bug tracker activity
 - Enhance views with that meta-data

Contents of project dashboard page

- Summary of failures per test collection
 - Aggregate qualitative results (PASS in 214/260 tests)
 - Distinct results for specific tests (FAIL to boot)
- Summary of benchmark changes
 - Deviations from baseline and other major changes
- Key, pre-selected, benchmark results
- Bonus features
 - Packages, bugs, etc

How do we get all the data?

The general idea

- Get a test suite/test collection
- Run it on a device
- Upload logs into the dashboard
 - Tell dashboard how to parse the logs in advance
- Have dashboard read the log files
- See the results

Log files are the key

- Each test run ends as a bunch of log files
- Output of the actual tests
 - Text output
- A software profile log
 - Packages, origin
- A hardware profile log
 - CPU, memory and devices
- All logs are uploaded to the dashboard

Logs tell the truth

- Historical source for all data presented by the system
 - Like wikisource for wikipedia
- Analysed on the server
 - We don't trust the client computing anything
 - This approach is Just Better™ (faster, safer, repeatable)
- Logs turn to samples
 - Samples carry references to the log file
 - You can always check where they came from

Samples...

- Contain single qualitative measurement
 - Pass vs Fail/Crash/Skip/Timeout
- Or quantitative measurement
 - 323115123 of ...
 - ...com.phoronix.aio-test.sequential-read.thread1of1
- Optionally store references to emitter and time stamp
 - To connect samples from one run (time stamp)
 - To connect samples from various runs (emitter identity)

...and emitters

- Source/identity allowing to group samples
- Add maintenance burden
 - We have to identify the emitter along with the sample
 - We need to invent a name space for the emitters
 - Add register it in the system
- Required for benchmarks
 - Otherwise results make no sense
- Optional for tests

How do we show the data?

Archetypes of sample views

- Aggregate qualitative summary
- Discrete qualitative
- Qualitative without time stamp
- Qualitative with time stamp

Aggregate qualitative summary

- Aggregate view of multiple samples from qualitative tests
 - Shows amount of pass/fail results
- Can compare results to samples in other test runs
 - How did this do yesterday?
 - ~~How did this do on other hardware?~~
- Requires little effort to produce and maintain
- Suitable for tests with large number of test cases

Discrete qualitative

- View for samples from specific qualitative emitter
- Can compare results to samples in other tests
 - Did this particular test fail yesterday?
- Suitable for specific questions
 - Does it install/boot/turn off?
 - Did it exceed the memory quota?
 - Did it fail? (regression tests)

Qualitative without timestamp

- Single measurement without associated timestamp
 - Average disk read speed during test
 - Time to boot to desktop
- Needs emitter identity to associate with other samples
- Displayed as graph:
 - Separate data series for different environments
 - Values across measurements from different images

Qualitative with timestamps

- Multiple associated measurements with timestamps
- Sampled with certain frequency
 - Disk read speed/s
 - Video frames/s
 - System/program memory consumption/s
- Displayed as graph
 - Single series for each run
- Could be simplified to average value if it makes sense

Archetypes influence the implementation

- Core scenarios for graphing samples
- Determine database query
 - Filtering
 - Aggregation
- Determine the data model
 - Fields
 - Joins

Applications for view archetypes

- Results from current image vs older images
- Results across different image flavours
 - Core, server, netbook, etc
- Results across different value of hardware property
 - Physical Memory
 - CPU Speed
 - Storage device type

Other considerations

Secondary things to deliver

- Build manifest harvesting
 - Packages, versions
 - Could allow more interesting queries
- Build log harvesting
 - That's just LexBuilder
- Bug tracker integration
 - Find bugs targeting milestone
 - Launchpad integration

Bonus features

- Be compatible with

<https://dev.launchpad.net/LEP/ResultsTracker>

- Be useful to the regular Ubuntu teams

Questions?

Thanks

<http://wiki.ubuntu.com/Specs/M/ARMValidationDashboard>