



# Automated Reliability Testing via hardware interfaces

Bits & Chips 2011

Embedded Systemen

Bryan Bakker

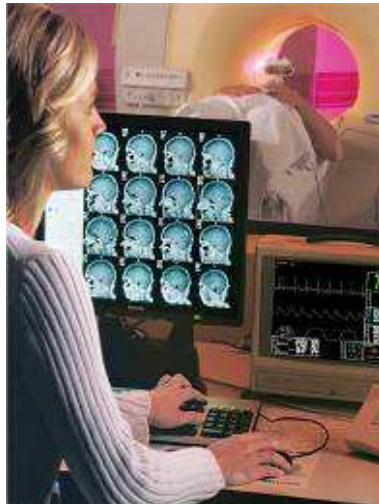


November 2011

- Sioux
- Intro – The need for action
- Increment 1 – First success
- Buy-In from management
- Increment 2 – A language for the testers
- Increment 3 – Logfile interpretation
- ROI and Crow-AMSAA
- Results – key success factors



- Test Expert
- Certifications: ISTQB, TMap, Prince2
- Member of ISTQB Expert Level on Test Automation
- Accredited tutor of ISTQB Foundation
- Domains: medical systems, professional security systems, semi-industry, electron microscopy
- Specialties: test automation, integration testing, design for testability, reliability testing



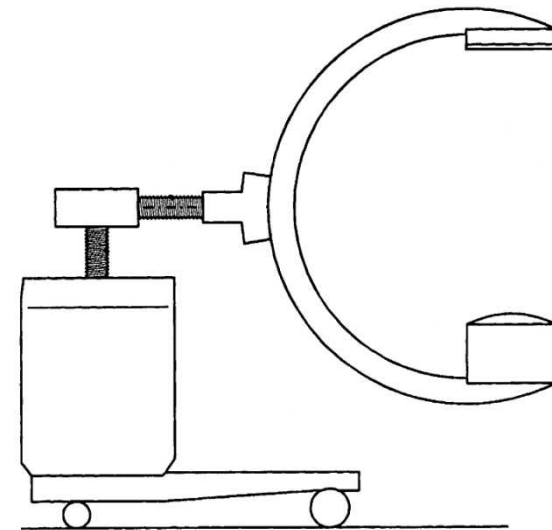
## Medical Surgery Device:

- X-ray exposure + acquisition during surgery activities
- Real-time image chain
- Mobile device (frequently off/on)
- Quality and testing considered important in organization

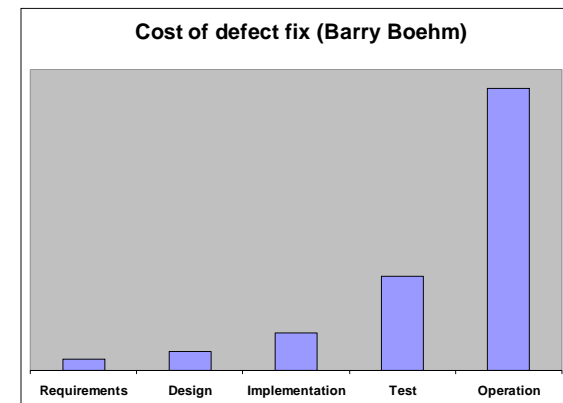
## Reliability was an issue:



- “Frequent” startup failures
- Aborted acquisitions
- Always safe... but not reliable!

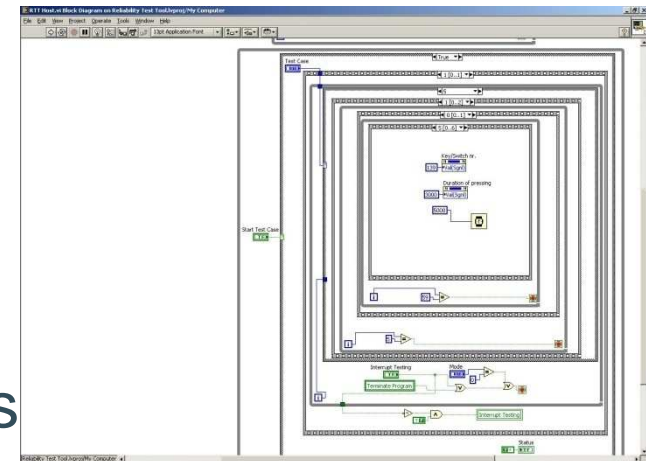


- Reliability issues are nasty to analyse, solve and test
- Fixing defects in field (remember Boehm)
- Impact on other projects (development + test resources)
- High service costs
- Troublesome system test (up to 15 cycles!!)

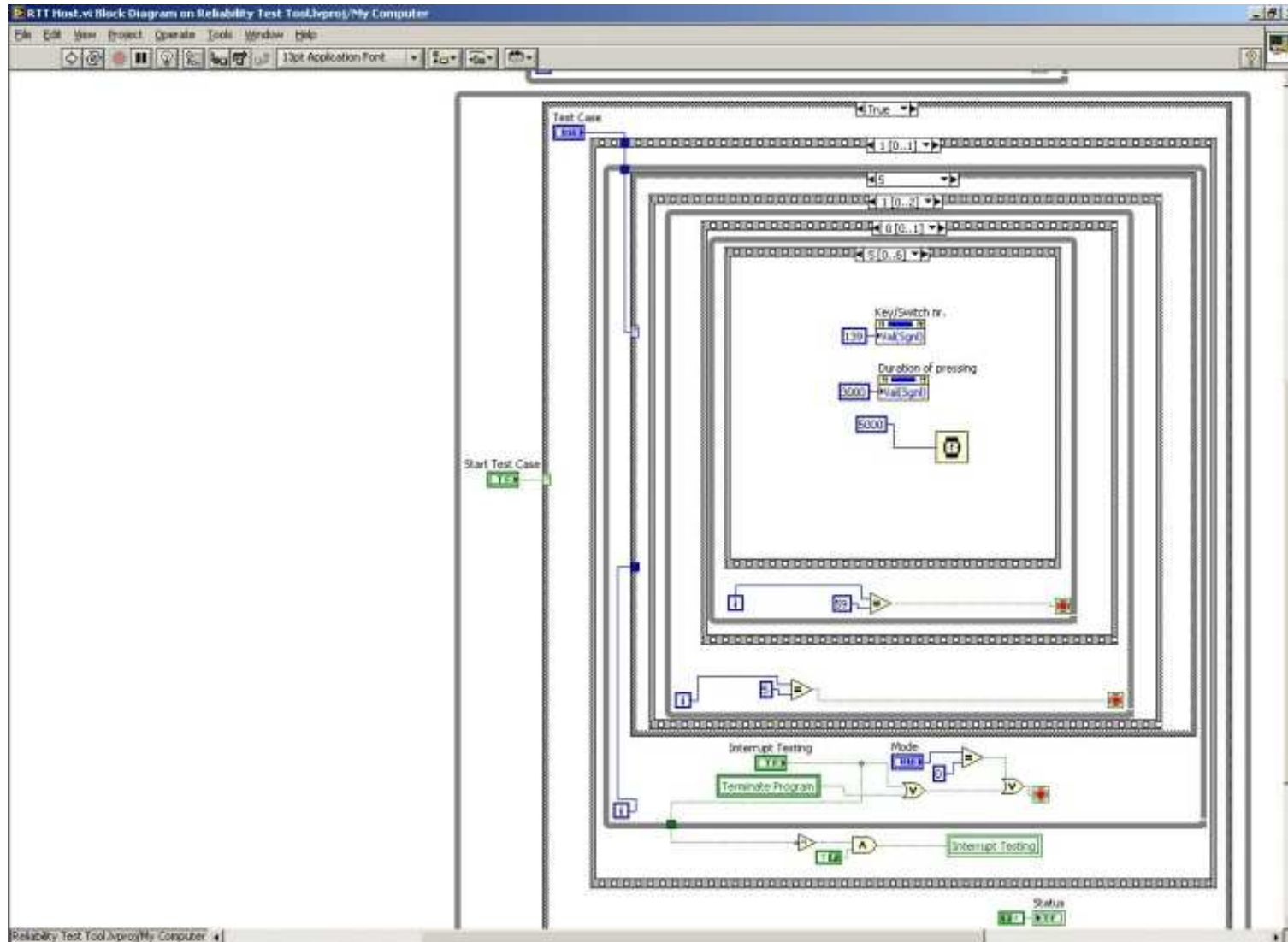


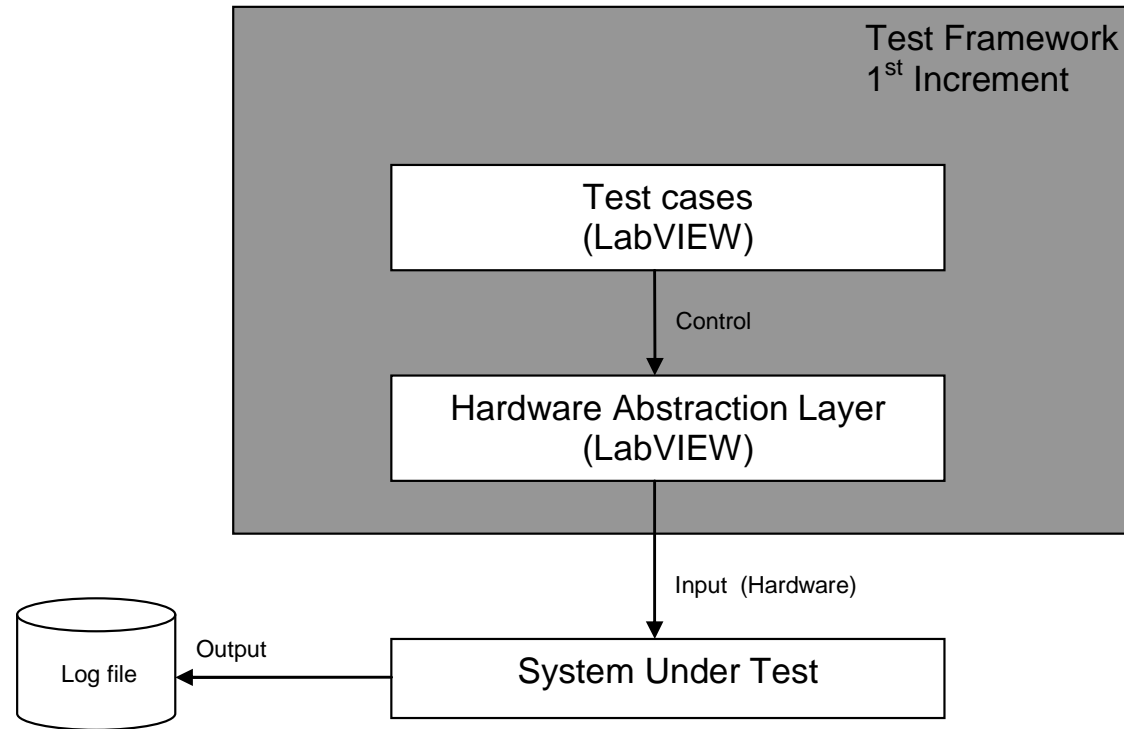
- Start with automated reliability tests (simple, short)
- Quick and dirty at first
- No SW resources available
  - To help with automation
  - Implementing test interfaces
- Goal: show (quick) that reliability issues can be reproduced
- Expectation: ... then attention and funding would increase

- Hardware interfaces used to invoke actions on SUT
  - Buttons on different keyboards
  - Handswitches
  - Footswitches
  - Different power-switches
- LabVIEW generates hardware signals
- Test cases defined in LabVIEW
- Only logfiles stored, no other verification performed
- No software changes needed for this approach



# Increment 1 – First success





- Simple, but quick first results
- Multiple reliability issues found
- Work to do for the developers

- Several defects found were already known:
  - Customer issues
  - Not reproducible -> no solution
  - Now: developers could work on them
  - And fix could be tested as well
- Several presentations given explaining the approach
- And... get clear what we are looking for!
- Primary functions should work reliable

# Definition of Reliability Hit

1	2	3
<b>Primary failure</b> <b>Problem in PF</b>	<b>Secondary failure</b> <b>Problem outside PF, but</b> <b>impact on PF</b>	<b>Tertiary failure</b> <b>No direct impact</b> <b>on PF</b>
PF fails Reset during PF	PF cannot be started Reset outside PF	Functional failures

**Reliability Hits**

Also important but not for reliability

PF = Primary Function

Primary function: e.g. startup, acquisition

Non-primary function: e.g. printing, post-viewing

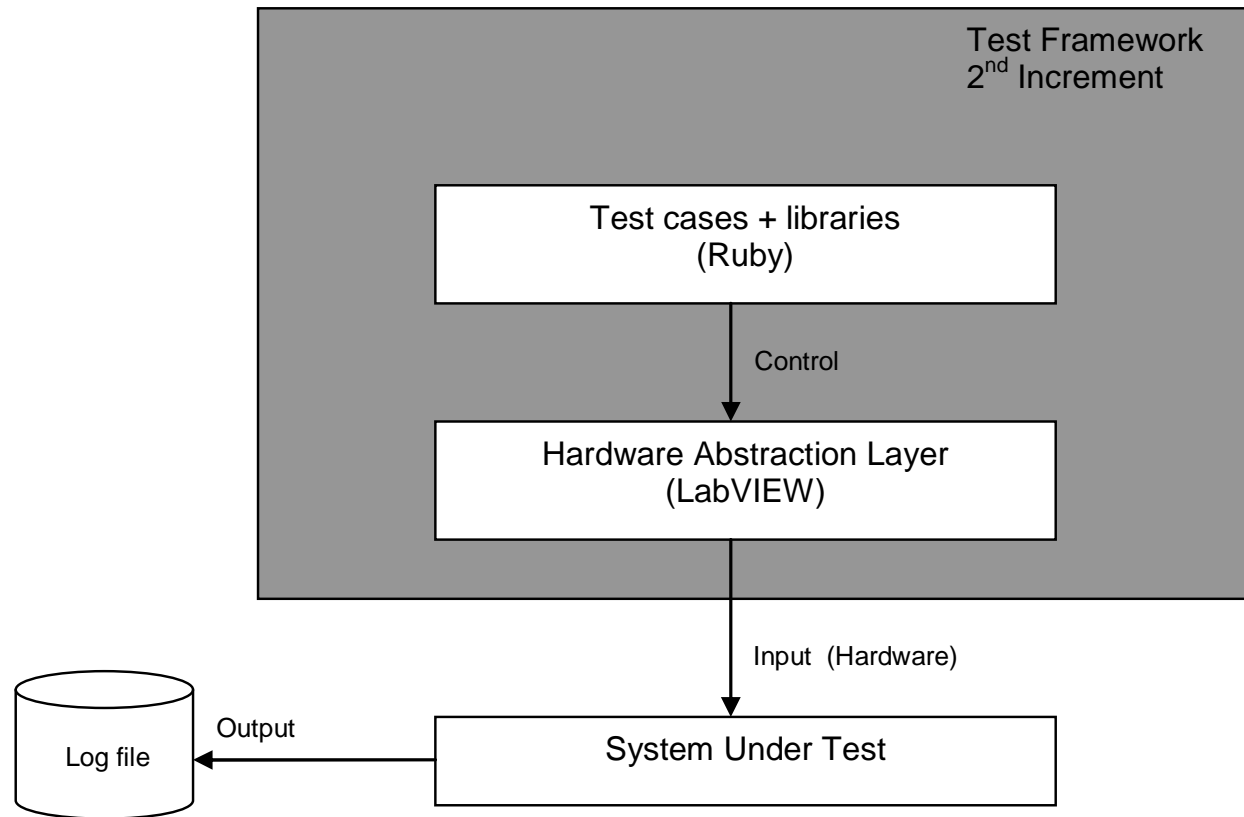
# Increment 2

## A language for the testers

- LabVIEW is not that easy
- Provide general scripting language (Ruby)
- Ruby interfacing with LabVIEW via abstraction layer
  
- Development of test libraries was started
- Still only control, no verification
- Log file analysis after test (tools were used)

# Increment 2

## A language for the testers

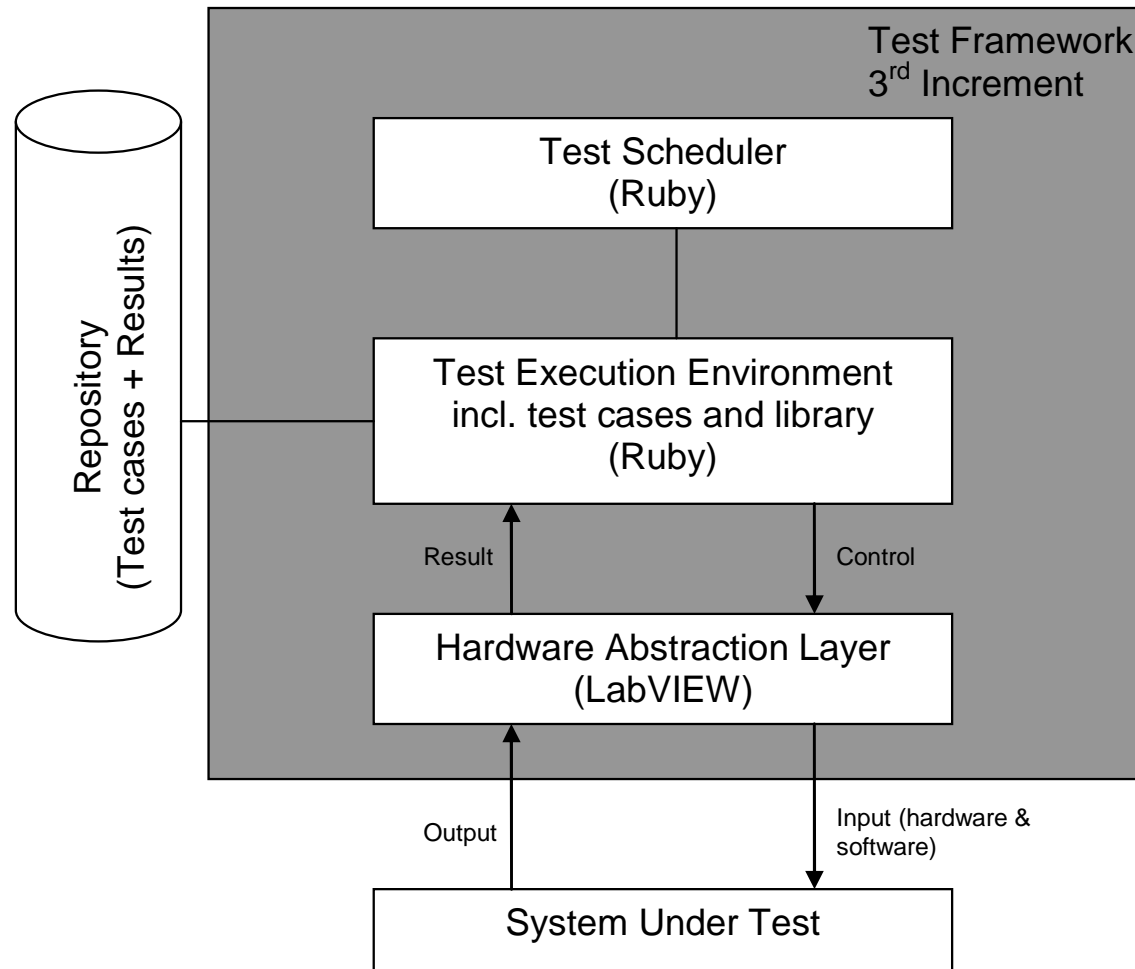


# Increment 3

## Logfile interpretation

- Logfile scanned during test case execution
- Determine pass/fail criteria
- Detect system states and act upon:
  - Hot generator → extensive acquisition not possible
  - Execute other test cases (e.g. power-cycle), until
  - Generator has cooled down
- Log file analysis after test was still performed
- Still no software changes in the SUT, but existing interfaces were available now

# Increment 3 Logfile interpretation

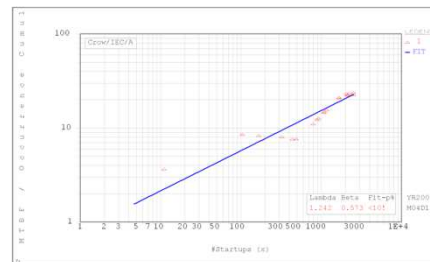
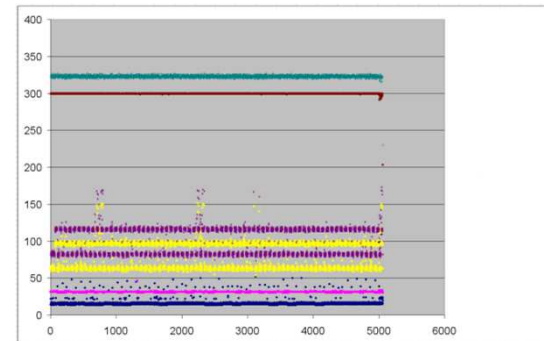
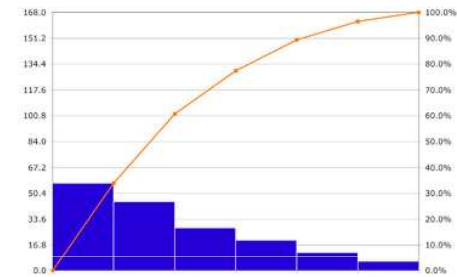


# Increment 3

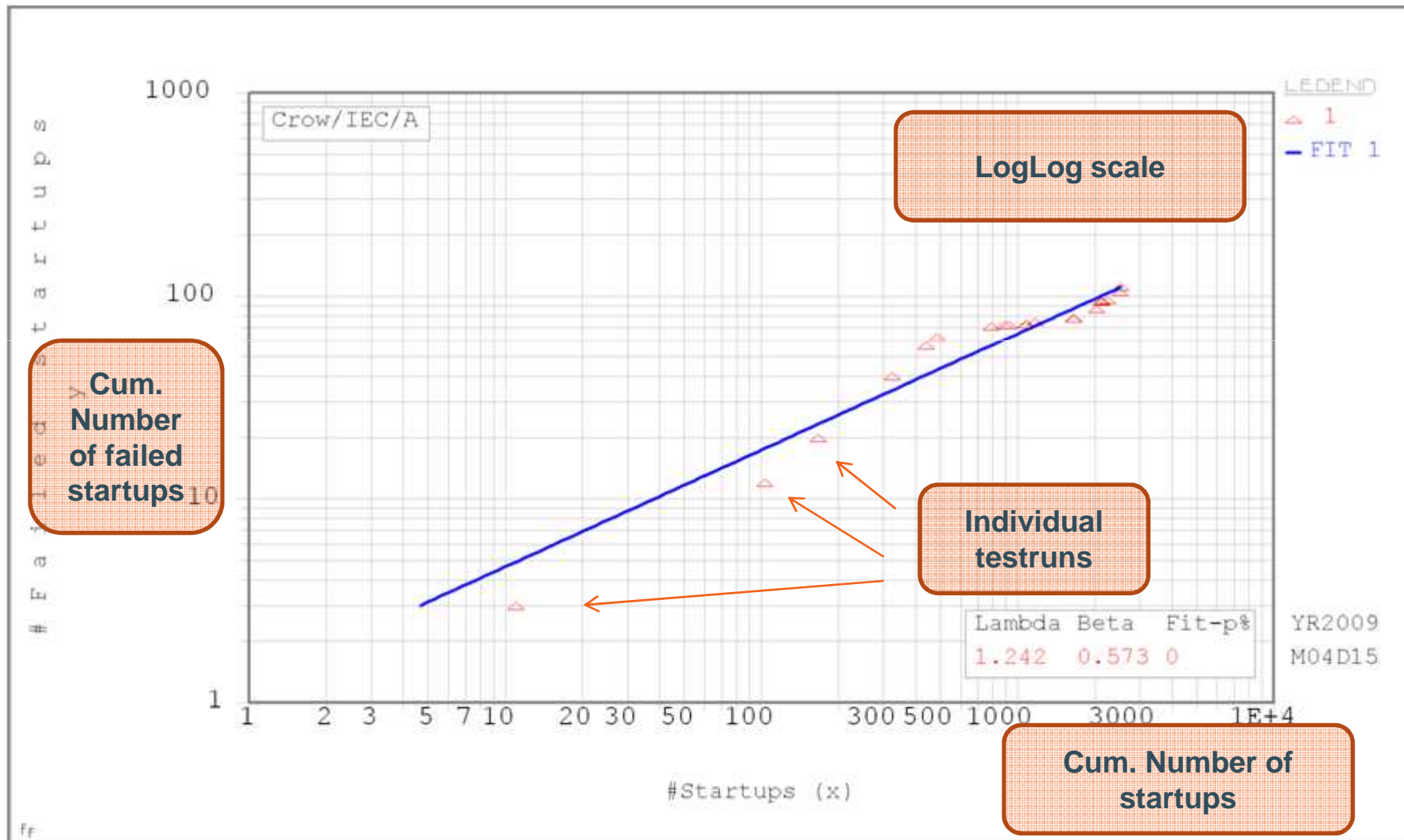
## Logfile interpretation

- Best practise:
  - Test actions by external interfaces
  - Test verification by log file and internal state information
- System statistics extracted from logfile:
  - Number of startups (succeeded and failed)
  - Number of acquisitions (succeeded and failed)

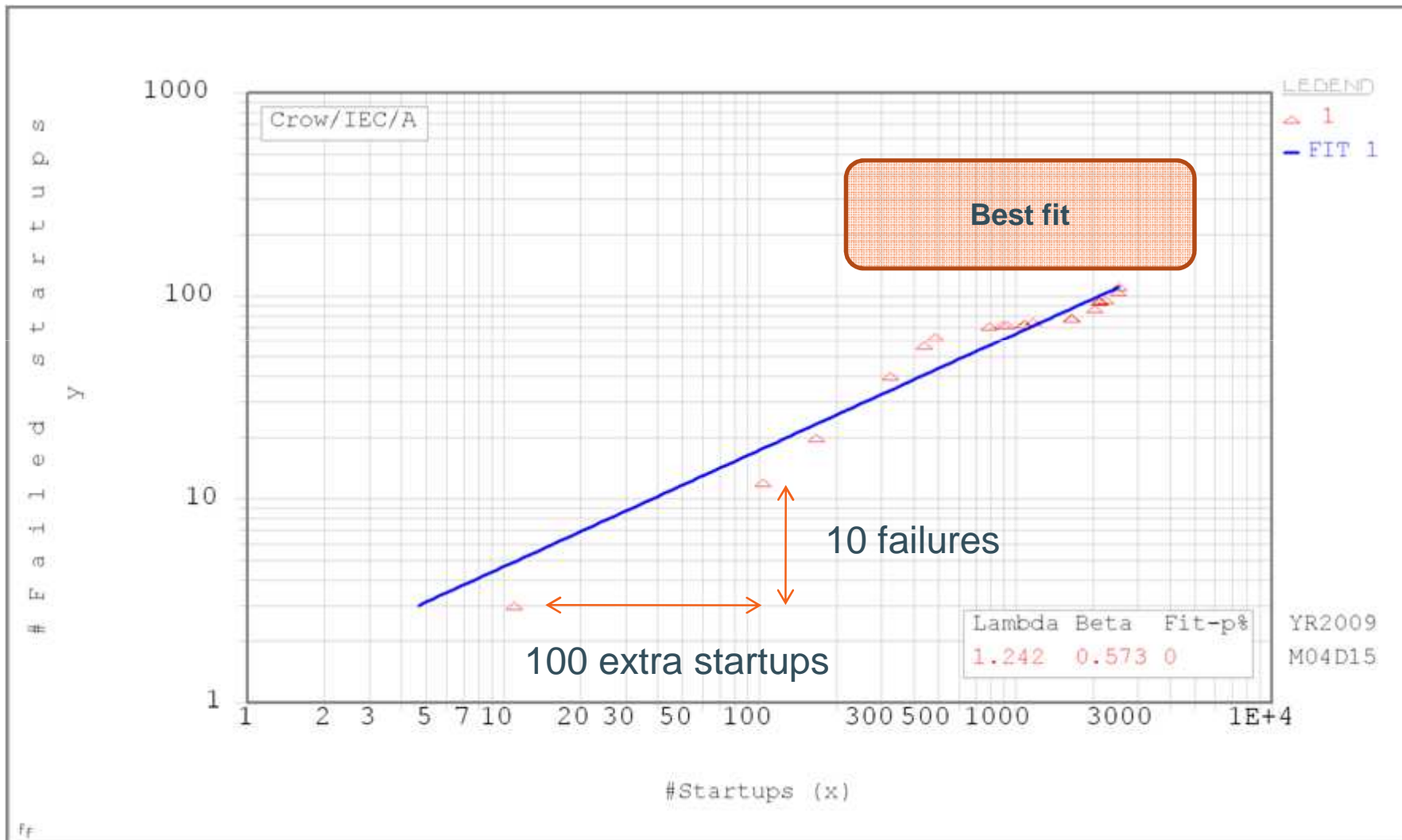
- Reliability hits could be identified from logfile (semi-automatic)
- Pareto charts
- Performance measurements (timing info in logfile)
- Crow-AMSAA



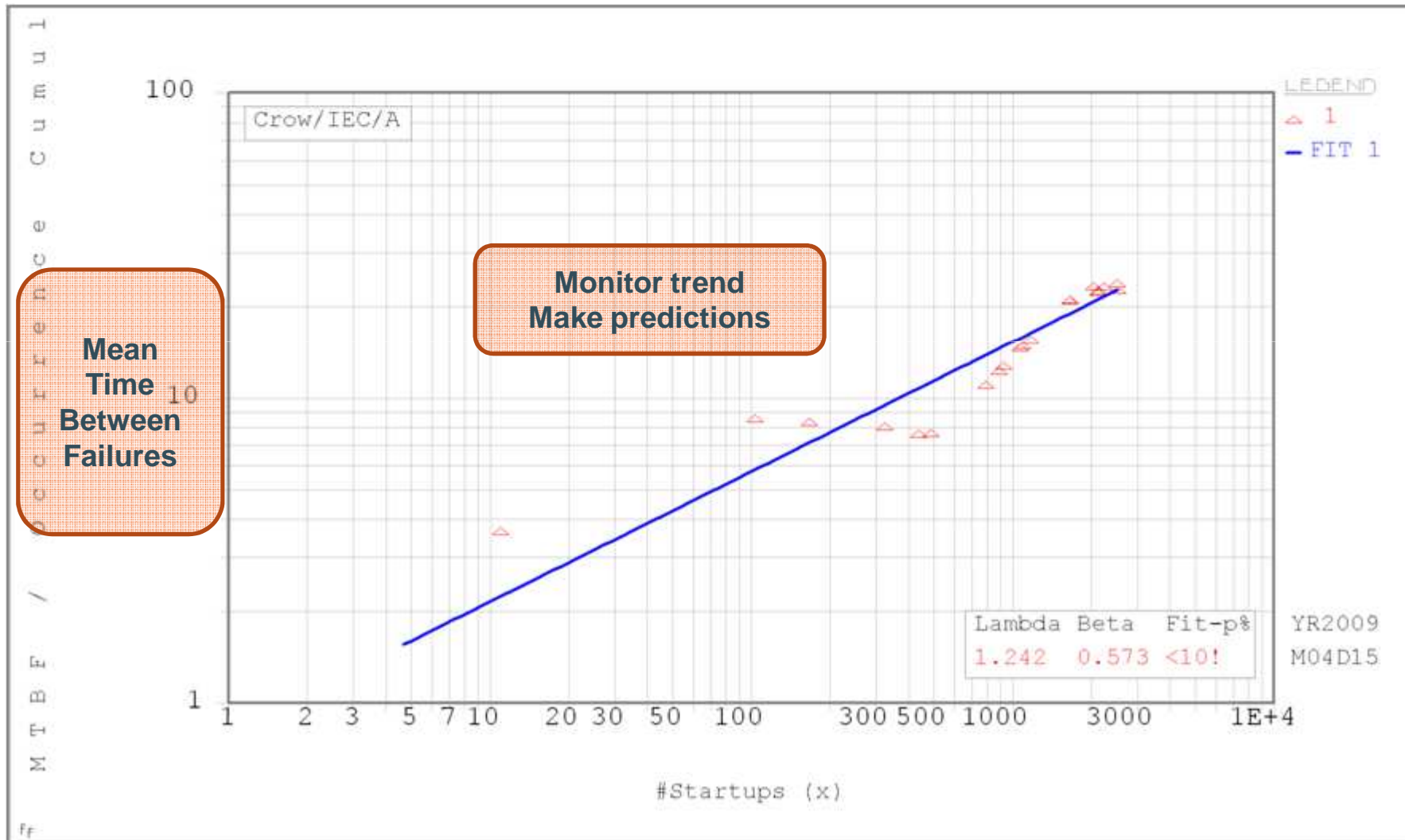
# Crow-AMSAA Failure Plot Example



# Crow-AMSAA Failure Plot Example



# Crow-AMSAA MTBF Plot Example



- >100 reliability hits identified
  - Which ones would have slipped through other tests?
  - Which ones would the customer complain about?
  
- “Independent” analysis of hits:
  - 8 would have been in system test, but not earlier
  - 7 would not have been found, but customer would complain (and fix would be necessary)

- ROI:  
 $(8 \times X_1) + (7 \times X_2) - \text{costs} > 0$
- Costs (manhours + material) = 200K Euro
- $X_1$ : costs of defect found in system test: 10K Euro
- $X_2$ : costs of field defect: 200K Euro
  
- $80K + 1.4M - 200K \rightarrow$  **1.2M Euro saved**
  
- More money and time became available...
  - Implementing/executing more tests
  - More projects/products

- Numerous reliability hits identified + solved
- MTBF measured and predicted
- Startup MTBF increased by factor 7.6
- Acquisition MTBF incr. by factor 18
- More testing hours on systems
- Customer satisfaction
- More projects wanted this approach
- Only 5 system test cycles remaining (was 15)

- Choose right project at the right time
- Incremental development (early visible benefit)
- Communication / ROI
- Clear and simple reporting (Crow-AMSAA)
- Hardware interfaces
  - Low probe effect (not a single false alarm)
  - Easy ported to different products





- This case study is described in detail:

Dorothy Graham & Mark Fewster  
Experiences of Test Automation

Case studies of software test automation

ISBN 0321754069



Source of your development.

010101001010101010111010100011  
1010010101101001010010100100  
11101101010101010101010101010  
101010101010101010101010101010  
001010101010101010101010101010  
101010101010101010101010101010



[www.siox.eu](http://www.siox.eu)



[bryan.bakker@siox.eu](mailto:bryan.bakker@siox.eu)



+31 (0)40 26 77 100