PHILIPS

Design for testability in CardioVascular x-ray acquisition control software

Nico van Rooijen - software architect MR

PHILIPS

Content

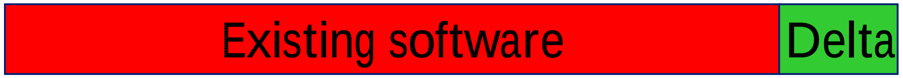
- Focus on what we did from ± 2000-2005
  - every project a small improvement

- Drivers & architecture context
- Test environments
- Test driver tools
- Results checking & coverage
- Analyzability
- Automatic test scripts

- short demo

Drivers

- Very high requirements for product quality
  - CV systems are interventional
- Continuous quality in the archive
  - "always a working system"
  - Continuous development & test cycles
- Testing and bug fixing consumes more and more time
  - Especially regression:

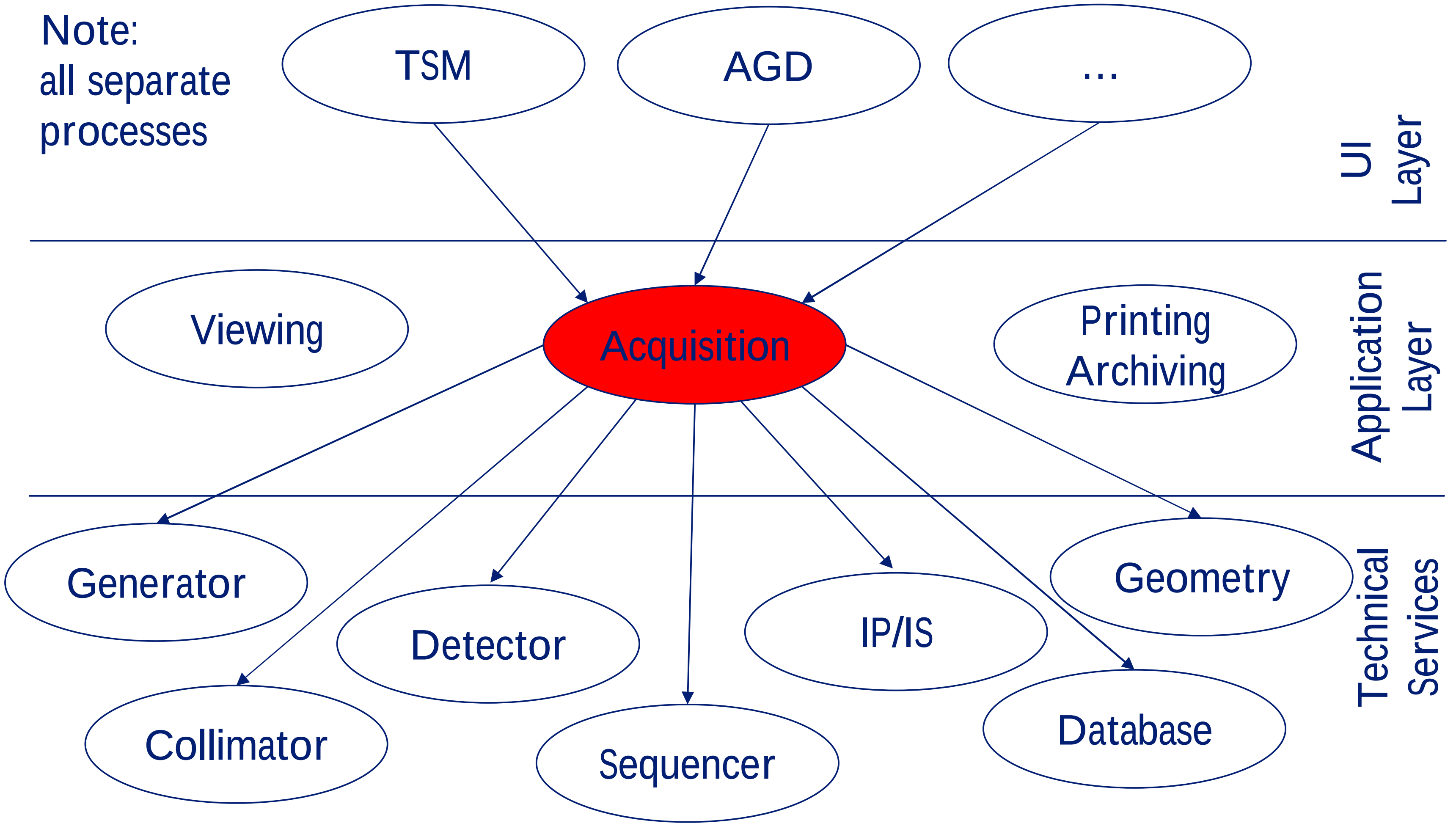


- Test systems are expensive and scarce resources
  - do as much as possible on developer PC

PHILIPS

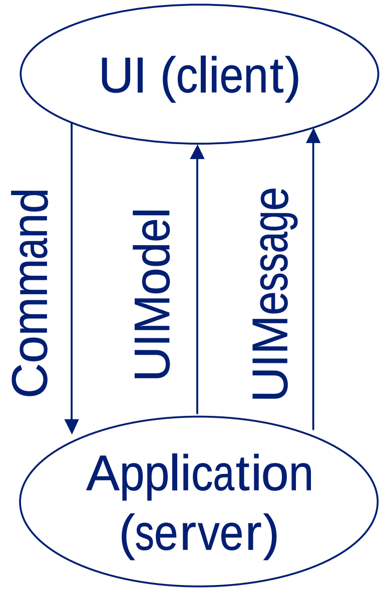
Architectural context (1)

Note:
all separate processes



PHILIPS

Architectural context (2)

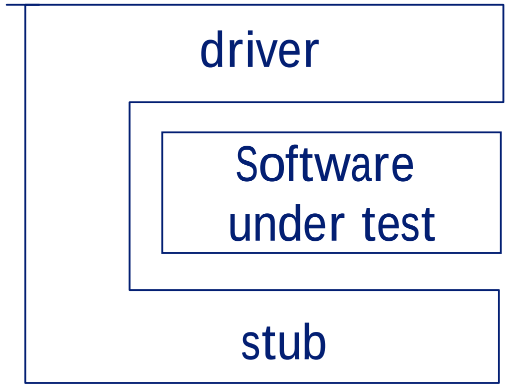


- Very strict interfacing between applications and UI:
  - Commands
  - UIModel (observer)
  - UIMessages (observer)
- Rationale: system has many different UI's

PHILIPS

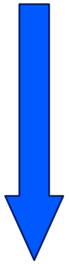
Test environments

- Test harness
  - Classic setup



I

- Standard test environments:
  - LIT (developer PC, VMWare)
  - Villa Volta
  - BOK
  - OTM

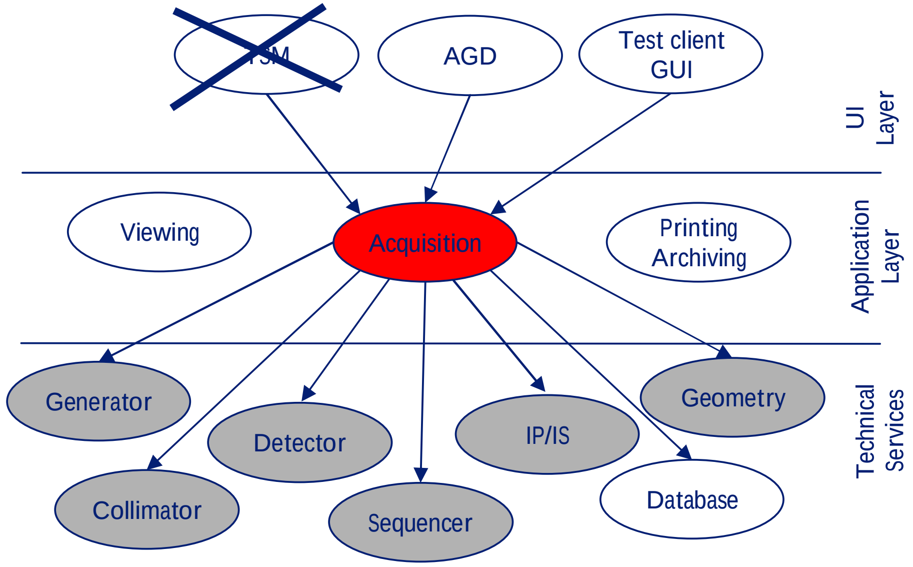


More realistic;
More expensive

PHILIPS

Standard test environments: LIT

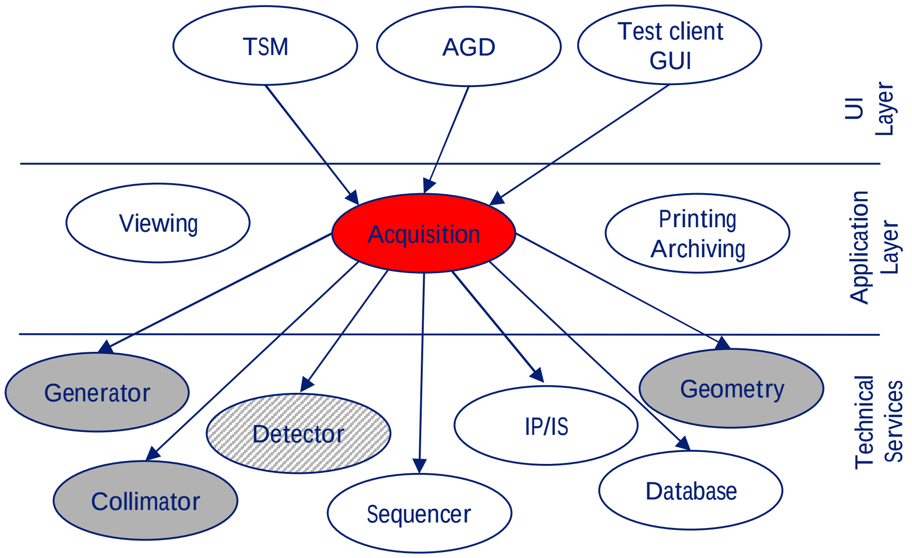
- Developer PC; VMWare
- No hardware
- Simple stubs / simulators (e.g. return callbacks on asynchronous calls)



PHILIPS

Standard test environments: Villa Volta

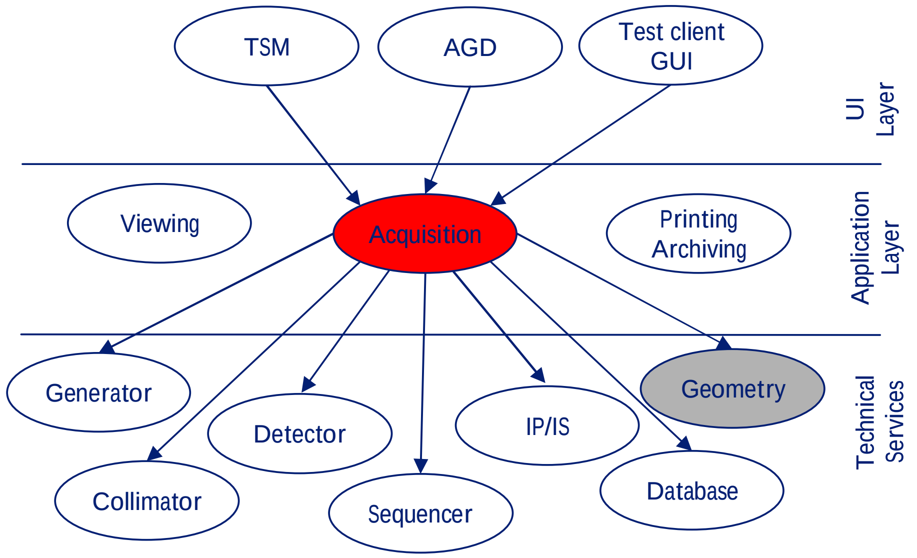
- Target PC hardware
- Some peripheral hardware



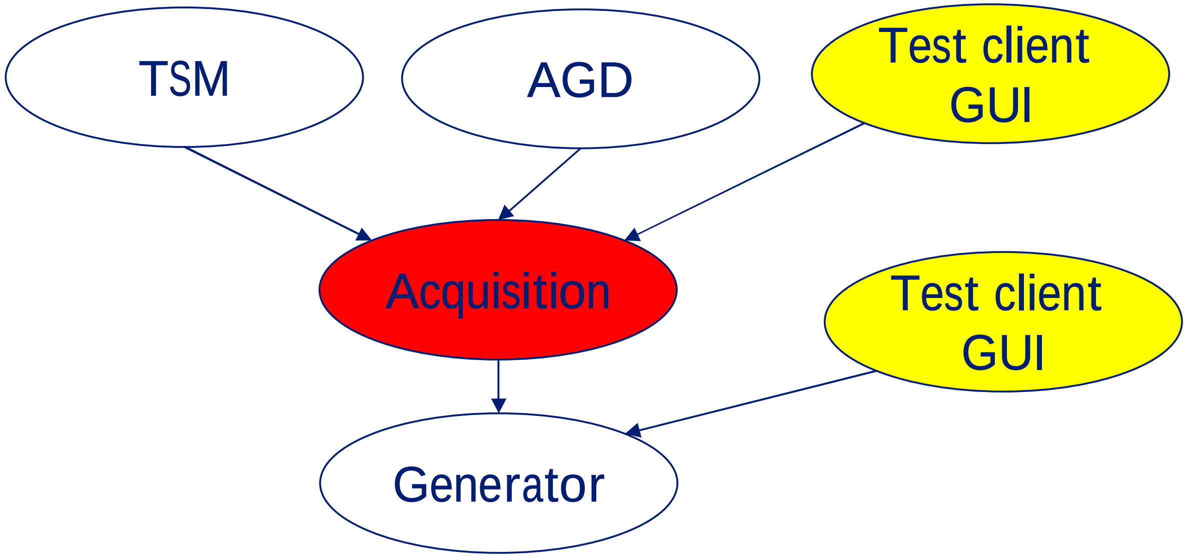
PHILIPS

Standard test environments: BOK/OTM

- Ever more realistic (and expensive)
- OTM has real geometry

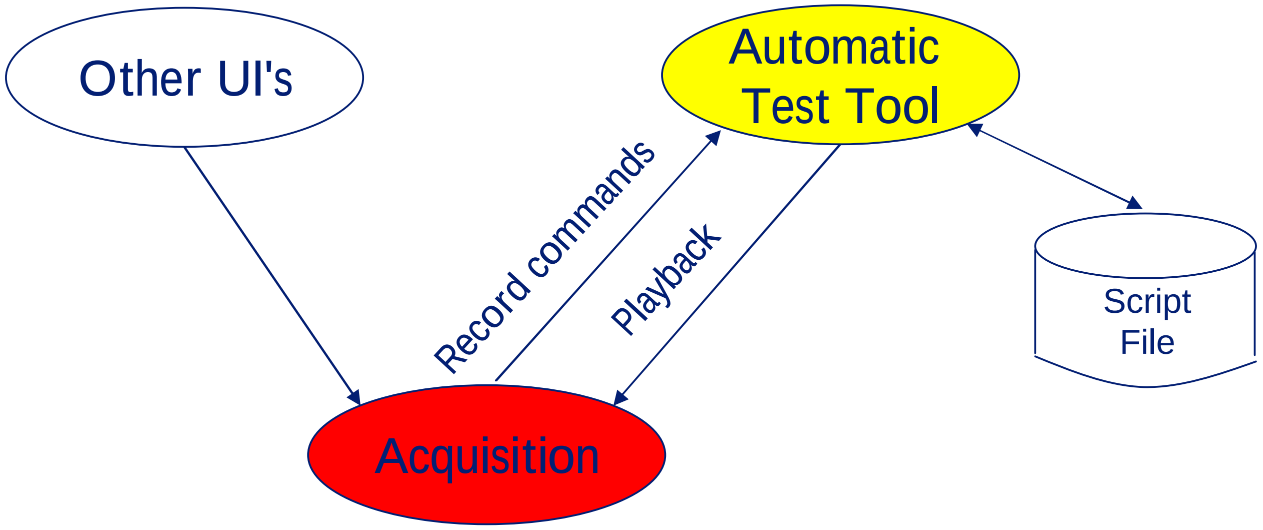


Test client GUI's



- Given the architecture almost a trivial step
- Run side by side with other UI's on a running system
  - start/stop at any time
- Also available for technical layer

Automatic test tool



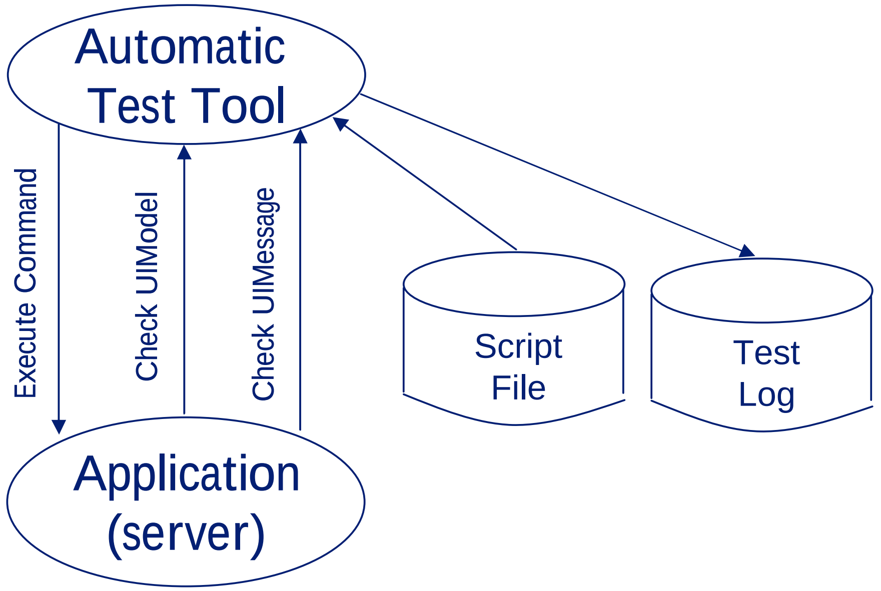
- Very simple tool
- Run side by side with other UI's
- Record/playback all commands (originating from any UI) of all applications to/from simple script text file
- Can run in any of the standard test environments!!!

Discussion: module tests

- We started doing module tests using the test harness.
  - Lots of maintenance, main causes: a) large fan out of interfaces b) each module own implementation of test environment c) lots of poorly designed testcode
  - Still many problems during integration ("correct" modules do not yield correct end-user requirements)
  - Conclusion: only efficient for some modules
- More succesful approach *(Focus of presentation):*
  - use standard test environments
  - Use and extend Automatic Test Tool
  - Focus on testing end-user requirements
  - Add test interfaces to the application to be able to reach sufficient coverage

PHILIPS

Results checking



- Simple additions to Test Tool:
  - Check UIModel item for value
  - Check for UIMessage
- Synchronization
  - Wait for UI model item to reach some value (losing timing dependencies!!)
- Making available some internals for checking
  - E.g. state of most important state machines
  - Using UIModel, so available on Test GUI & Automatic Test Tool
- Discussion: value of indirect results (e.g. x-ray on UIModel item)

PHILIPS

Enhancing coverage

Operation of Acquisition application is highly parameterized by:

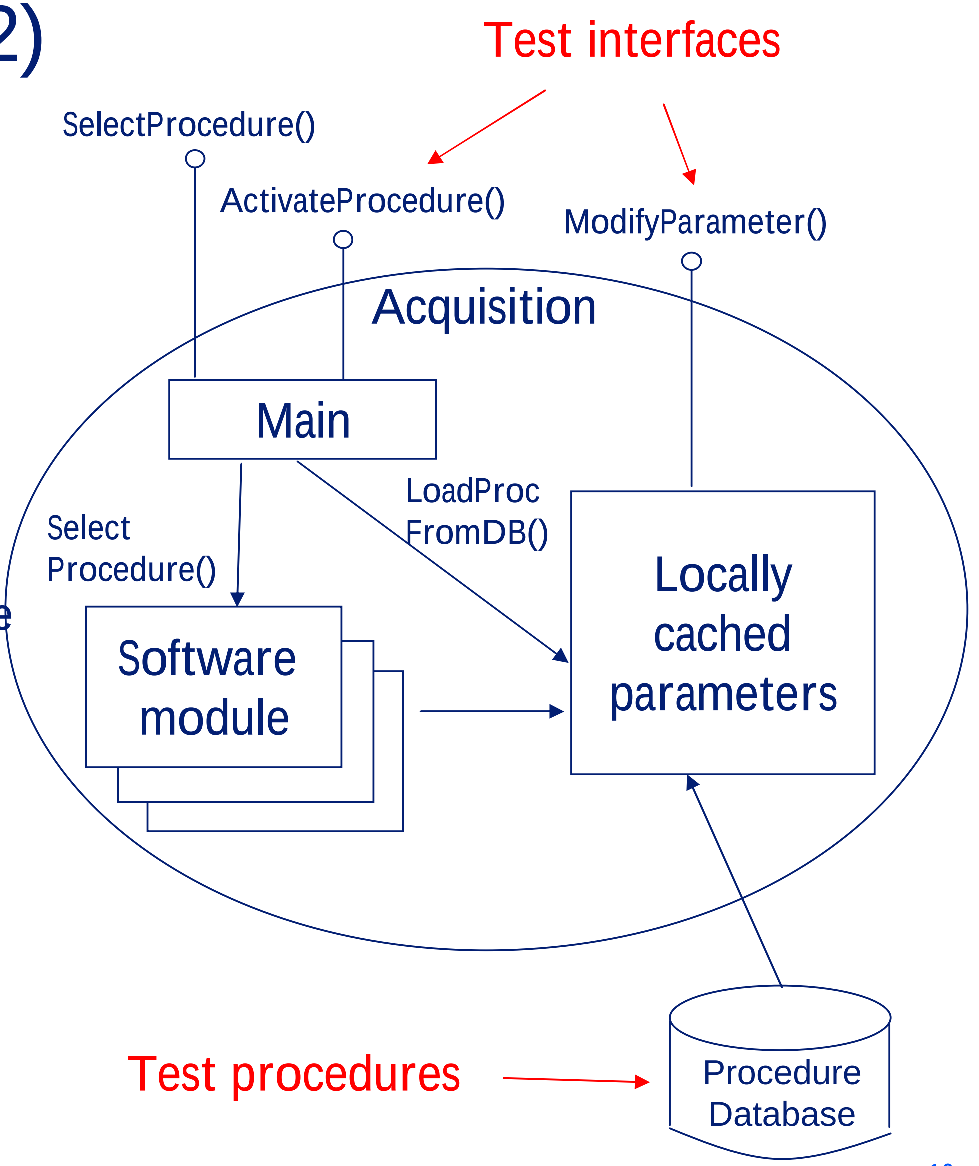
- procedures ("acquisition recipies") in the Database (200 params)
  - Preset procedures are delivered by Philips, but are modified in the field
  - Parameter combinations need to be validated against each other and the configuration. Validation itself needs to be tested.
  - Huge parameter space
- Available licenses
  - Usually only the "full options" situation is tested during everyday testing. Creating different license files is cumbersome.
  - Many combinations possible
- Configuration
  - Limited amount (so far …..)

- How to get good coverage?

PHILIPS

Enhancing coverage (2)

- Procedure parameters:
  - Limited number of test procedures in DB
  - Test interface to create "variations on a theme" + activate modified procedure without reload from Database
  - Use interface from test client GUI & Automatic test tool
- Licensing is done analogous
- Configuration not solved yet



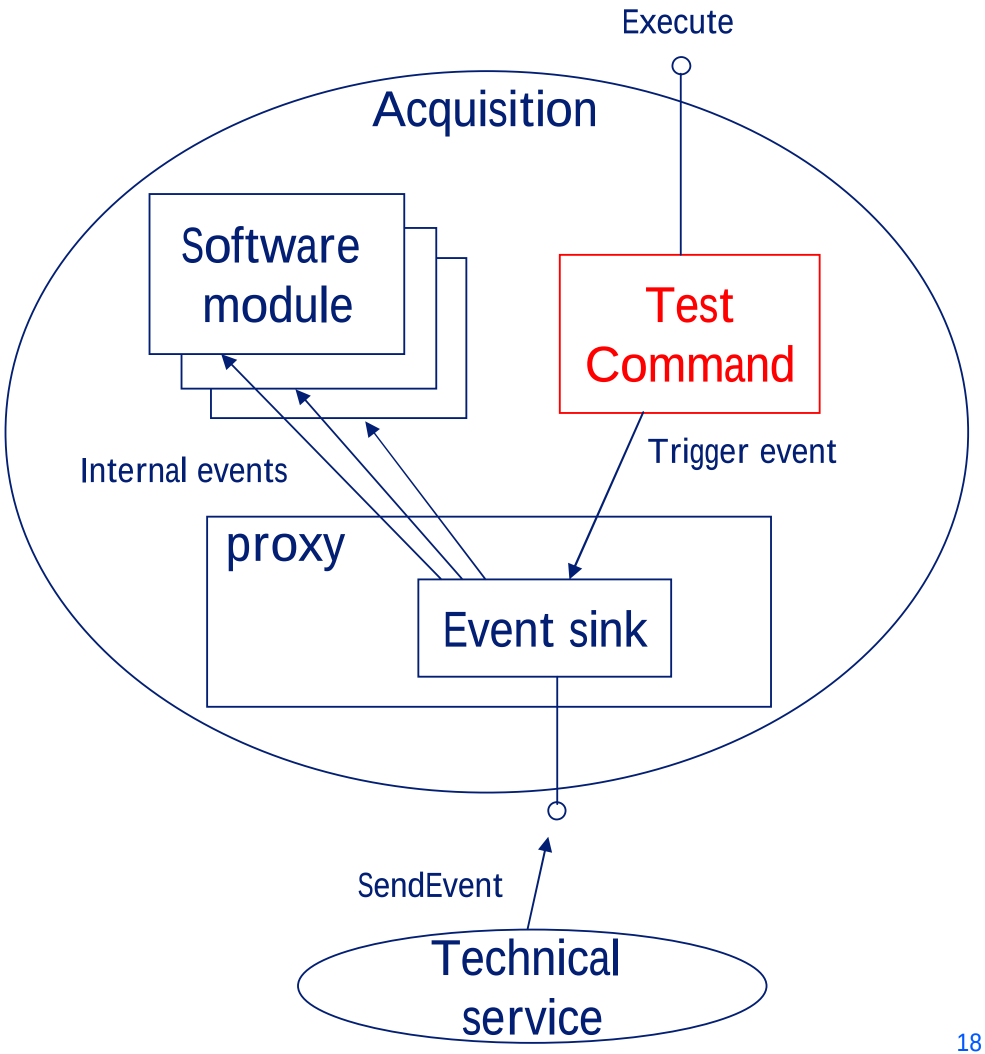
PHILIPS

Enhancing coverage (3)

- Avoid needing shutdown/restart for reloading new parameters
  - takes lot of time, makes manual testing annoying
- Rule: all parameters are re-evaluated at a procedure selection

Enhancing coverage (4)

- Generate stimuli from other then UI sources: events from the technical layer, e.g.:
  - Door open
  - Tube too hot
- Two possibilities:
  - Make more sophisticated simulators with test interface
  - Generate internally
- Last possibility chosen. Can also be used in complete system



PHILIPS

Analyzability

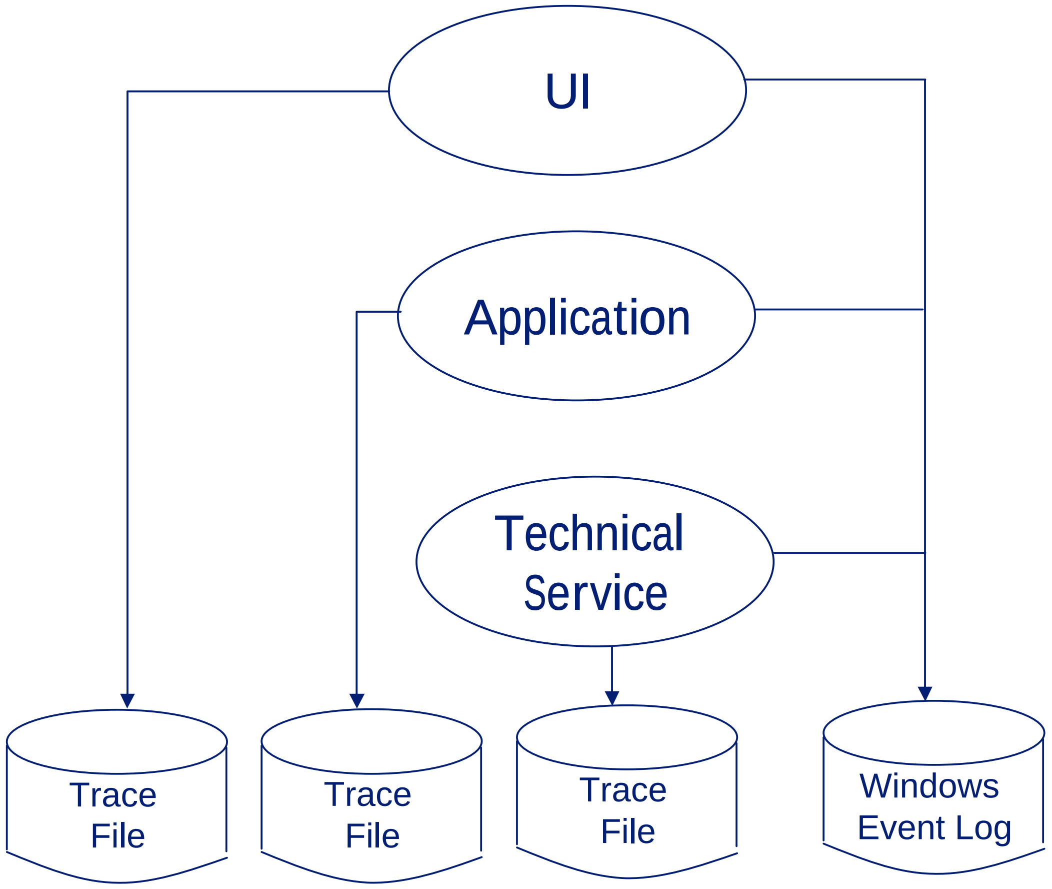
- What if a problem is found (either by automatic or manual testing)?
- We started by using the debugger with dissatisfying results:
  - a) always needing to reproduce, takes time b) what exactly happened? c) intermittent problems d) breakpoint wrong --> next cycle
  - Consequence: test systems occupied by debugging developers
- We set ourselves the goal:
  - Analyze 80% of problems a) ***offline*** b) ***without reproducing***

PHILIPS

Analyzability (2)

- We had logging in Windows event log
  - Meant for field service engineers
  - Global overview, lack of detail, too slow mechanism

- Added simple "tracing"
  - Simple and efficient (>20 traces per ms) mechanism, writes to text files
  - Meant for development debugging
  - Most tracing always on!



PHILIPS

Analyzability (3)

Design trace file contents: use design knowledge

- Get as much as possible valuable info against little performance penalty

- Outgoing/incoming trafic to/from outside (including values of variables)
- Windows message pump
- Values of used procedure parameters and licenses
- Internal state transitions
- Critical sections, locks, windows events
- For each trace: Thread ID, time stamp (enables performance analysis)

- OO: give objects instance names to know who traces

Analyzability (4)

Results:

- Goal reached, mayor efficiency improvement
- Way of working completely changed
  - Tester just saves data, submits PR, continues testing
  - Took time to get everybody on board, now nobody wants to go back
- Very strong combination with automatic testing

Discussion: testability in production code

- Not everybody liked the idea to keep test interfaces and tracing in the production code.
- However:
  - never had a problem with this
  - Release the system as it was tested
  - Have capabilities in the field
- We keep it in

Managing test scripts

- OK, now we have some nice Automatic Test Tool, test interfaces, analysis capabilities. Let's make some scripts!
- Oops: Test scripts themselves became a maintenance burden
- Therefore, we felt the need to:
  - Clearly organize the scripts
  - Document what we have

PHILIPS

Managing test scripts: organizing

3 levels of test scripts:

- Utility
  - Reusable "sub"-scripts of actions/checks
- Testcase
  - Create their own preconditions, can be executed in any order
  - Composed by executing Utility scripts
- Batch
  - Sequence of testcase scripts
  - Long/short batches

PHILIPS

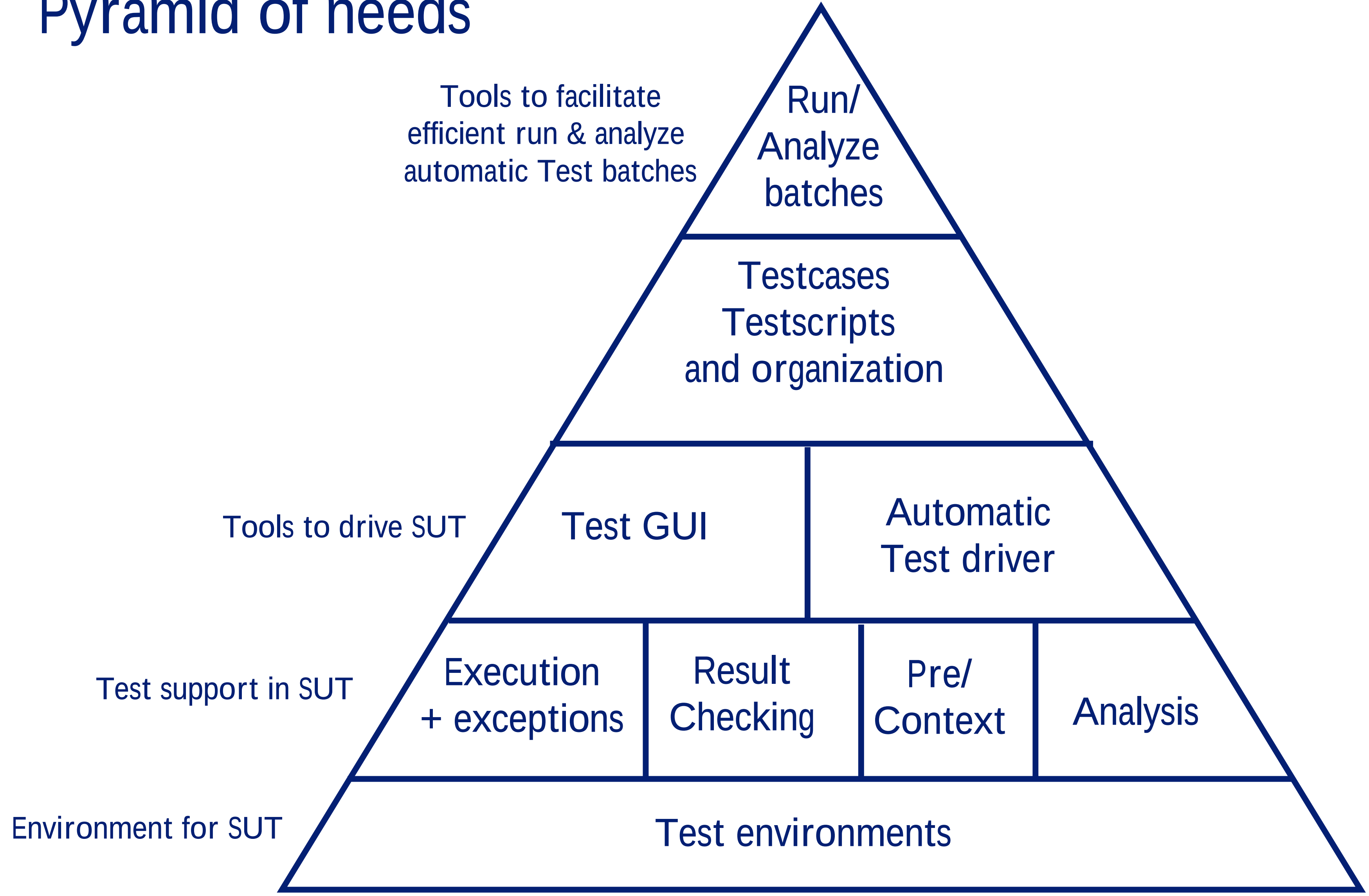
Managing test scripts: documenting

- We had a good pool of manual test cases
- Simple decision: one-to-one correspondence of Testcase scripts with manual testcases in the TS
- Some manual checks could not be automated (yet). Therefore two modes of execution are supported by Automatic Test Tool:
  - Attended: scripts pauses and prompts for manual check
  - Unattended: scripts only performs automatic checks

Facilitating to run automatic test batches

- Some tooling was created to:
  - Easily setup test systems to run test batches
  - Save all test results (test log, Windows event log, trace files) on central server
  - Easily get an summary of the test results + starting point for analysis in case of failures

Pyramid of needs



Current status

- We have come a long way
- No post without running the automatic batches
- Every developer runs batches on developer PC while developing
- Continuous improvement each project
  - Investments pay off!

Short demo