Designing Hardware-independent Testing Laboratory API

Paweł Wieczorek
August 22, 2019

Samsung R&D Institute Poland
Motivation

Testing laboratory layers

Case study

Summary
Motivation
## Use cases

### Automated testing

1. **New software**
2. **Allocate resources**
3. **Perform tests**
4. **Release resources**
5. **Return results**

### Direct access (hacking)

1. **Issue investigation**
2. **DUT acquisition**
3. **Interaction**
4. **DUT release**
5. **Issue solved**
Abstractions (https://elinux.org/Test_Standards)

- **TM** (Test Manager): Actions initiation
- **TS** (Test Scheduler): Resource allocation, scheduling
- **DUT-C** (DUT Control): Controlling power, providing network, ensuring communication, grabbing logs
- **DUT** (Device Under Test)
Common laboratory structure
Decoupled laboratory structure

- Test manager
- Test scheduler
- DUT Control
- DUT
SD-MUX

https://wiki.tizen.org/SD_MUX
https://wiki.tizen.org/MuxPi
Testing laboratory layers
Challenges

Knowledge
Which actions are necessary? Where can it be performed? How to do it?

Responsibilities
Who performs given action?

Sharing
Who can use DUT? How can DUT be used?
## Implementation

<table>
<thead>
<tr>
<th>Test Manager (minimal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• initiate actions</td>
</tr>
<tr>
<td>• list (or cancel) currently performed actions</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test scheduler (generic)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• list available resources, request specific ones</td>
</tr>
<tr>
<td>• acquire assigned resources (then prolong, finally release)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DUT Control (tricky)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• boot (and login)</td>
</tr>
<tr>
<td>• execute commands</td>
</tr>
<tr>
<td>• copy files</td>
</tr>
</tbody>
</table>
Case study
# Test Manager

## Strengths

- Requires only preparing test plan
- Test plans can be reused among various projects

## Weaknesses

- Keeping compliance
- Catching up with others (e.g. LAVA, SQUAD)
**Test Scheduler**

**Strengths**
- Users treated equally
- Resource type-agnostic

**Weaknesses**
- Requires additional agent
- Capabilities declared up front
<table>
<thead>
<tr>
<th>Strengths</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Only some knowledge required</td>
</tr>
<tr>
<td>● Unification possibility</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Hard initial setup</td>
</tr>
<tr>
<td>● Often unique for a given testing laboratory</td>
</tr>
</tbody>
</table>
Summary
● Unable to demo without specific hardware
● Risky large scale deployments
● Responsibilities division allows easier onboarding
Conclusion

- User-centric approach resulted in smaller building blocks
- Smaller blocks could be easier swapped or used independently
- Improvement needs more *reuse* instead of *rewrite*

https://github.com/SamsungSLAV
Thank you!

Paweł Wieczorek
p.wieczorek2@samsung.com