Andreas Weitl, System Engineer

Automation for Nightly Build Testing

24.10.2013
Automation for Nightly Build Testing

• Introduction CASSIDIAN Test Facilities

• History and past experiences
• First steps: automate build; automate setup generation
• Version management and connection to source control

• Bring everything together
• Model Based Tests integrated in Nightly Build
• Investigation of standard tools
• Benefits

• Screenshots of the Tool
• Next steps
• Conclusion
Introduction Cassidian Test Facilities (1)

Airborne Mission Systems Integration

- Test Facilities
  - Germany

- System Integration FR
  - France

- System Integration Testing SP
  - Spain

EADS

Cassidian

Engineering

- Mission & Communication Systems
  - Spain

- Mission Systems Combat A/C
  - Germany

- Mission Systems Sensors & Armament
  - Germany

- CLAEX
  - Spain
Introduction Cassidian Test Facilities (2)
Introduction Test Support System Family AIDASS®

**AIDASS Integration System:**
Basic core for Integration & Test Facilities for commercial and military A/C’s and UAV’s such as A/C Flight Control Systems and Avionic Systems

**AIDASS Compact RIG** – VME Based
Due to its expandability it can be used as Test Support System for any size of applications

**AIDASS Compact RIG** – PCI Based
As Test Support System for medium size applications

**Mini AIDASS**
- up to 7 PCI slots available
As Test Support System or as a mobile bus analyser for use on A/C

**Fully Rugged Aircraft Ground Equipment**

**Rugged or Semi-Rugged Mini AIDASS** – VME Based
As in field Test Support System or mobile bus analyser.
Introduction  TORNADO System Integration Rig
History and past experiences (1)

- High number of software parts (components)
  Some parts on Windows; some on VxWorks Realtime OS; some on Linux
  Complex built process

- Project gets bigger and bigger.
  One day effort to manually build the system.
History and past experiences (2)

Past

SuT
F1; F2; F3

ATP
T1; T2; T3

Test System
TF 1; TF2; TF3

Now

SuT
F1 .. F1000

ATP
T1 ... T500

Test System
TF 1 .. TF 100
History and past experiences (3)

- Four nations project; lead has BAE SYSTEMS
  Predefined parts of the version string
  Difficult version management

- Different customers → Several variants of the system needed
  Even more difficult version management

- Generation of a new version is time consuming and high risk of mistakes

- Late information about errors
  Testing starts after version generation.
  Build of system is time consuming → this is not often done
First steps; automate build; automate setup generation

- To speed up build → scripts are created  
  Each platform has its own build

- Setup generation with script files as well

- These steps save several hours work time

- Most difficult part was the source control.  
  Due to many interfaces to the source control (selection of source specs) there  
  haven’t been automated scripts in the past.

- These interfaces create the idea of developing a separate tool to coordinate  
  the source control automatism.
Version management and connection to source control (1)

• Version management is difficult because of multinational and high number of components.

• Version string partly predefined
  Additionally version string should be different to distinguish product lines

• Two parts: A free text which hold the current version string
  a second text field which holds the name in the source control

• The text entered in the version control tool is used to automatically check the correct installation of all components
Version management and connection to source control (2)

Framework SC Label: 2012_05_12_Product1

Test Function1
SC Label: XX

Test Function2
SC Label: XX

RTOS SC Label: 2012_05_12_Product1

PU1
SC L: XX

PU2
SC L: XX
Version management and connection to source control (3)

Label String Format

- Label String used for Source Control tool
- YYYY_MM_DD_ProductLine_Version_Addon#PatchString
- DD: Day; MM: Month; YYYY: Year;
- ProductLine: AIDASS; TESIS; MaTE; ..
- Optional #PatchString:
  This allows to mix different labels without generating a version warning
Version management and connection to source control (4)

Framework SC Label: 2012_05_12_AIDASS_R12_0p
Test Function1 2012_05_12_AIDASS_R12_0p#fix1

RTOS SC Label: 2011_09_12_AIDASS_R11_0p
PU1 2011_09_12_AIDASS_R11_0p#patch3

Version Warning Log
Bring everything together

• JAVA based application with own GUI

• Application interacts with source control
  Build relevant information is available:
    Branches; Checkpoints; Label names; etc.

• Fresh sandbox with all source files is generated
  Calls preconfigured script files

• Source control label information automatically into source files
  used to perform self checks about the used components

• No special build code; setup generation code or error analyzing code;
  Encapsulation with script files

• Model Based Tests could be integrated in Nightly Build tool. Also test reports
  are transported and archived.
Model Based Tests integrated in Nightly Build

- Traceability Matrix
- Requirements
- System Specification.doc
- Test Plan
- Test Model
- Test Protocol Template.txt
- Test Protocol.pdf
- Test Analysis
- Result Files
- Result Analyzer Ruby Scripts
- Test Progress Report
- Manually Test Cases
- Auto. Image Test Cases
- Automatic Test Cases
- Dynamic Test Cases
- Stochasic Test Cases
- Manual Test Execution
- Automatic Test Execution
- Nightly Build Tool
- XML-Files

Test Protocol.pdf
Test Protocol Template.txt
Test Plan
Test Model
Test Analysis
Manual Test Cases
Auto. Image Test Cases
Dynamic Test Cases
Stochasic Test Cases
Manually Test Execution
Automatic Test Execution
Test Scripts
Test Generator
Script Generator
Test Generator
Result Analyzer Ruby Scripts
Result Files
Nightly Build Tool
XML-Files
Investigation of standard tools

Most known tools for this purpose are **Jenkins** and **Hudson**

- 6130 open issues – 651 *critical* or higher, a lot of them older than 2 years
  https://issues.jenkins-ci.org/secure/Dashboard.jspa

- Plugin-Interface is not well documented.

- **Code Quality:**
  “mostly undocumented code base with lots of open issues in Jira, a huge unmanaged set of dependencies and no test coverage to speak of. From my short glances at the code it looked like a hackish open source project with a few cool ideas but overall little emphasis on quality.”
  http://blog.mafr.de/2011/12/27/hudson-vs-jenkins-revisited

- Available Plugins do not create an advantage compared to the already available own solution.
Benefits

• Save a lot of time generating a version

• Prevents mistakes during generation of the system

• Allows an automatic build every night which checks automatically any build and compile errors on a daily basis

• We know the day after if something is broken in the source code and can start fix activities immediately

• Possibility to introduce automatic systems checks which extend the automatic testing also to the functionality itself

• Because tool itself is encapsulated with script files an adaption to new compile tools is easily and quickly possible without changing the Nightly Build Tool
Nightly Build Tool - Screen shots (1)
Nightly Build Tool - Screen shots (2)

![Nightly Build Tool](image)

<table>
<thead>
<tr>
<th>#</th>
<th>Project</th>
<th>Sandbox</th>
<th>Batch File</th>
<th>Parameter</th>
<th>Result File</th>
<th>Output Files</th>
<th>Abort</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>WinSymbol</td>
<td>A2k_SymbolLink</td>
<td>Sym</td>
<td>NB</td>
<td>no Result File</td>
</tr>
<tr>
<td>2</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>SymLink</td>
<td>MeTE_SymbolLink</td>
<td>Symbol</td>
<td>NB</td>
<td>no Result File</td>
</tr>
<tr>
<td>3</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>Windows</td>
<td>Works</td>
<td></td>
<td>D_A2k_All</td>
<td>D_A2k_Year</td>
</tr>
<tr>
<td>4</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>Windows</td>
<td>Works</td>
<td></td>
<td>D_A2k_All</td>
<td>D_A2k_Year</td>
</tr>
<tr>
<td>5</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>Windows</td>
<td>Works</td>
<td></td>
<td>D_A2k_All</td>
<td>D_A2k_Year</td>
</tr>
<tr>
<td>6</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>Windows</td>
<td>Works</td>
<td></td>
<td>D_A2k_All</td>
<td>D_A2k_Year</td>
</tr>
<tr>
<td>7</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>Windows</td>
<td>Works</td>
<td></td>
<td>D_A2k_All</td>
<td>D_A2k_Year</td>
</tr>
<tr>
<td>8</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>Windows</td>
<td>Works</td>
<td></td>
<td>D_A2k_All</td>
<td>D_A2k_Year</td>
</tr>
<tr>
<td>9</td>
<td>MeTE</td>
<td>dtl2_gen_v11.1 (new)</td>
<td>Windows</td>
<td>Works</td>
<td></td>
<td>D_A2k_All</td>
<td>D_A2k_Year</td>
</tr>
</tbody>
</table>

**Build Options**

- **Project**: MeTE
- **Sandbox**: dtl2_gen_v11.1 (new)
- **Batch File**: Windows|Works | D_A2k_All | D_A2k_Year
- **Parameter**: NB
- **Result File**: automatic
- **Abort**: errors
- **Send Mail**: errors
- **Job Name**: automatic

Apply Changes | Remove Build | Clear | Add Build
Nightly Build Tool - Screen shots (3)
Nightly Build Tool - Screen shots (4)
Nightly Build Tool - Screen shots (5)
Nightly Build Tool - Screen shots (6)

Build Result

Total
- Project: Windows Mote
- Type: Release
- Start Time: Mon Sep 16 13:33:55 CEST 2013
- Duration: 00:03:46

Subproject amount: 205
- Warnings: 54
- Errors: 0

Subprojects

- Name
- Warnings
- Errors

Build
- SP_InterfaceEvents_ControlServer
- SP_InterfaceEvents_ControlUI
- SP_Generic_Messages
- LCU_MPUX_Messages
- SP_InterfaceTracer_ControlServer
- SP_InterfaceTracer_ControlUI
- SP_StatusDisplay_StandaloneServer
- SP_StatusDisplay_StandaloneUI
- STM Main
- TMP_ControlUI
- LC_Services
- LG_DictionaryUI
- LR_DictionaryServer
- LR_DictionaryUI
- SL_DictionaryUI
- TM_ControlEnvironment
- SVC_RolesServer

Output

LINK 1: warning link4059: all references to "PPC412.DLL" discarded by JOPT1REP
LINK 1: warning link4059: all references to "SHLWAPI.DLL" discarded by JOPT1REP
LINK 1: warning link4059: all references to "USER32.dll" discarded by JOPT1REP
Next steps

✓ Automated Self Tests; we call it „Auto TPs“

✓ Unattended run

✓ Failsafe system

✓ Add tests to source control

✓ Include automated tests in the Nightly Build tool
  • Add model based tests in the Nightly Build tool, which run automated

✓ Change Nightly Build tool to run without source control connection
Conclusion

- Increased number of features → workload for version creation and validation increased
- Connection of build and test → increased effectiveness → keeps workload in acceptable borders
- Automatic report → affected developers → faster response
- Increased quality → less problem reports
- Higher customer satisfaction
Thank you for your attention!