OSTBAYERISCHE
TECHNISCHE HOCHSCHULE
REGENSBURG

Hausarbeit

Interkulturelle Kommunikation zwischen Entwicklungsstandorten - Am Beispiel von BSH Hausgeräte GmbH zwischen Regensburg und Indien

Studienleistung für

Interkulturalitäts- und Entwicklungsmanagement
(Master of Arts)

an der

Ostbayerischen Technischen Hochschule Regensburg

von

Thorsten Klein

Matrikelnummer, Kurs:	3153345, IEM
Dozent:	Frau Bedi-Visschers

Eidesstattliche Erklärung

Ich erkläre, dass ich meine Arbeit mit dem Titel

"Interkulturelle Kommunikation zwischen Entwicklungsstandorten - Am Beispiel von BSH Hausgeräte GmbH zwischen Regensburg und Indien"

selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe und dass ich alle Stellen, die ich wörtlich oder sinngemäß aus Veröffentlichungen entnommen habe, als solche kenntlich gemacht habe.

Die Arbeit hat bisher in gleicher oder ähnlicher Form oder auszugsweise noch keiner Prüfungsbehörde vorgelegen.

Ich versichere, dass die eingereichte schriftliche Fassung der auf dem beigefügten Medium gespeicherten Fassung entspricht.

Regensburg, 10. Januar 2018

Thorsten Klein

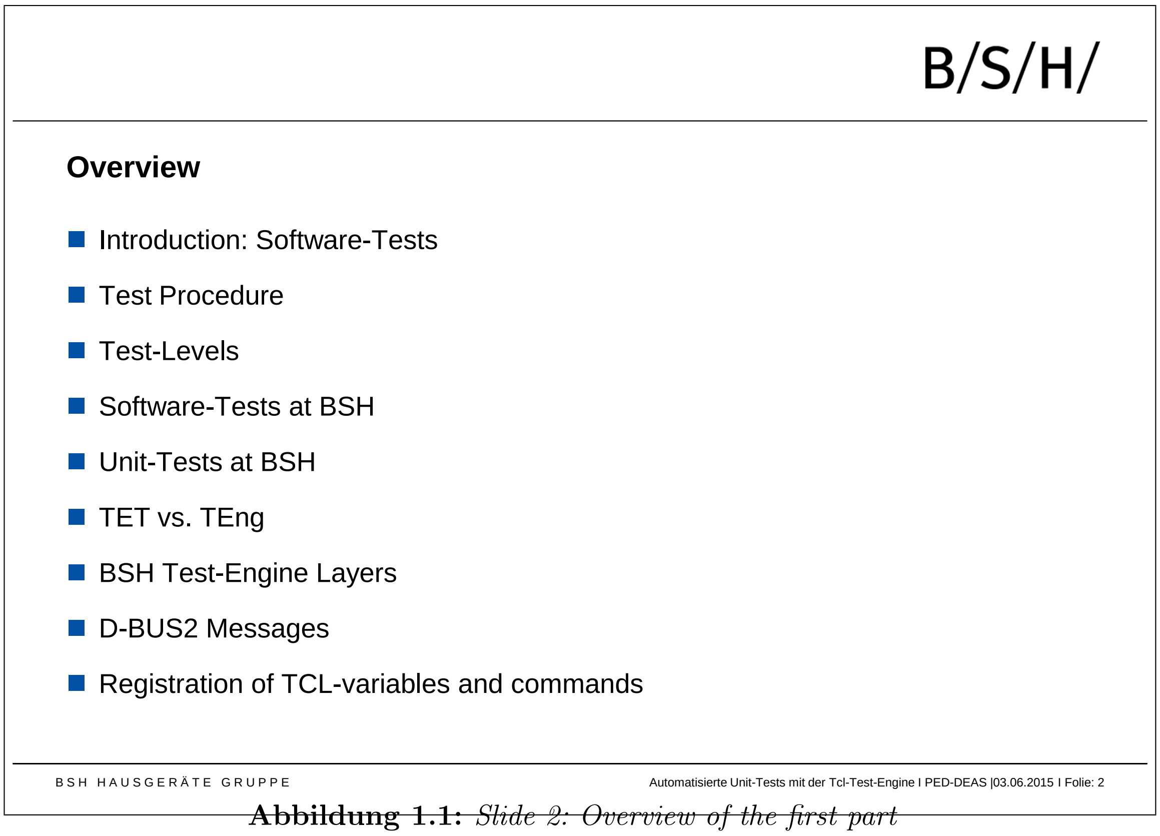
Zusammenfassung

Inhaltsverzeichnis

Abkürzungsverzeichnis

BSH	B/S/H Bosch Hausgeräte GmbH

1 Index



B/S/H/

Overview

- Introduction: Software-Tests
- Test Procedure
- Test-Levels
- Software-Tests at BSH
- Unit-Tests at BSH
- TET vs. TEng
- BSH Test-Engine Layers
- D-BUS2 Messages
- Registration of TCL-variables and commands

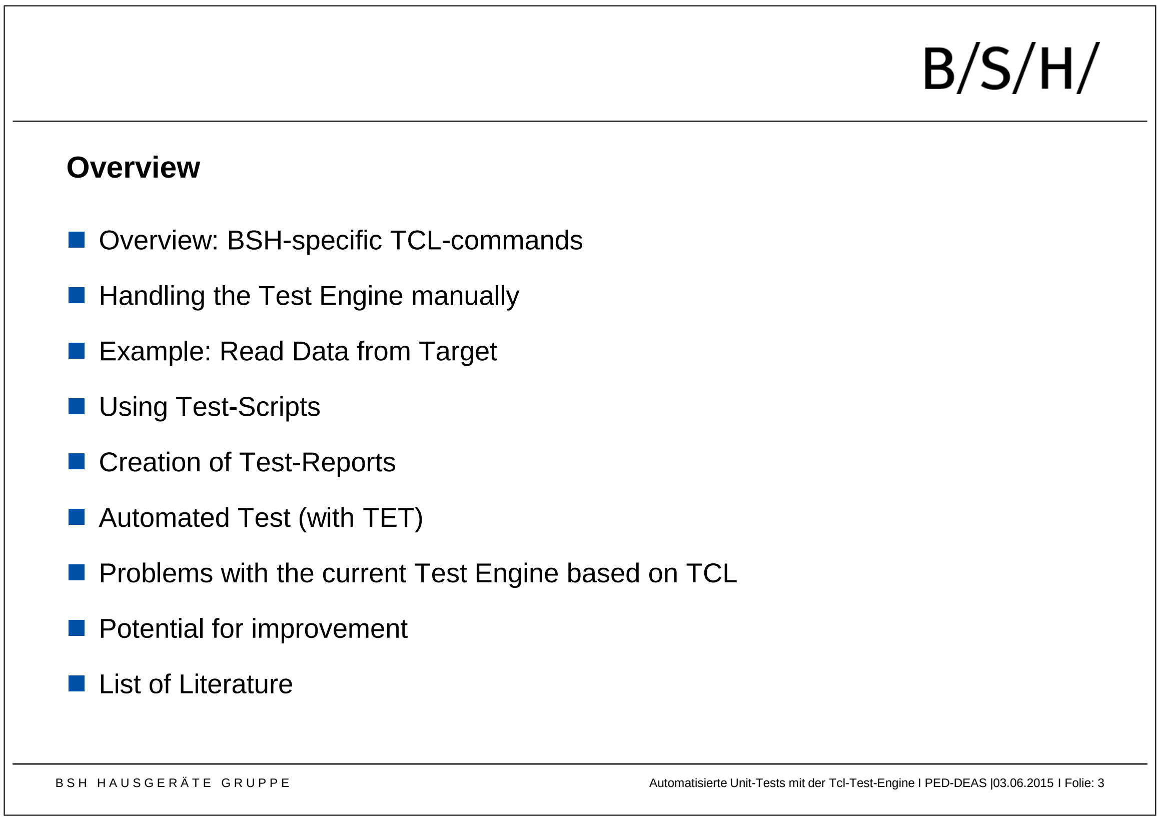
BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 2

Abbildung 1.1: *Slide 2: Overview of the first part*

In the first part of the presentation general information about software tests is provided, including a characterization of test procedures and the different software tests. After that the general information is put in concrete terms with BSH and the company definition of different levels of software tests is given. This also includes the definition of TEng (Test Engine) and TET (Test Engine Tool).

In the end of the first part, the way of registering Tcl-variables in the TEng and TET is outlined.



B/S/H/

Overview

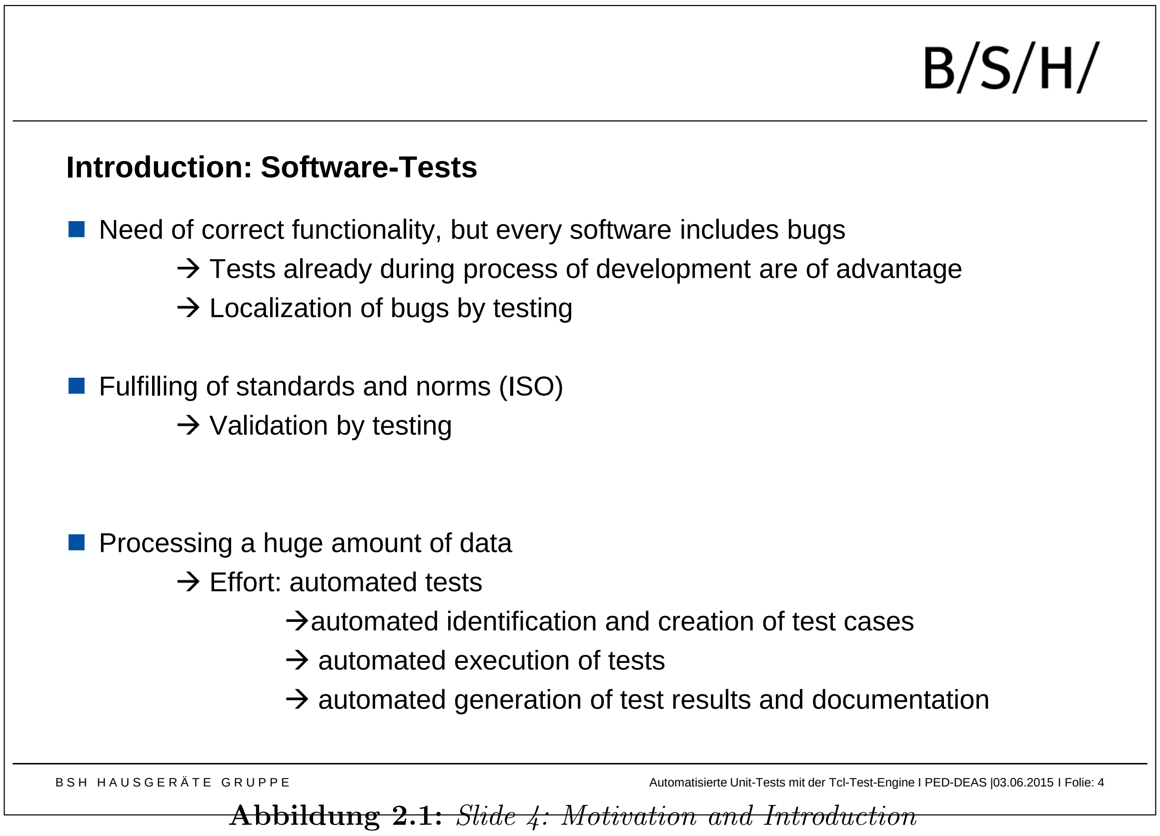
- Overview: BSH-specific TCL-commands
- Handling the Test Engine manually
- Example: Read Data from Target
- Using Test-Scripts
- Creation of Test-Reports
- Automated Test (with TET)
- Problems with the current Test Engine based on TCL
- Potential for improvement
- List of Literature

BSH HAUSGERÄTE GRUPPE — Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 3

Abbildung 1.2: *Slide 3: Overview of the second part*

In the beginning of the second part an overview about the BSH-specific TCL-command-set is presented with a short explanation each. These commands can be used to handle the TEng manually, which is demonstrated by an example.
In the following the usage of the TEng with test scripts and the requirements to test reports are shown.
Subsequently all these information are used to reveal the way of performing automated tests with the aid of the TET. As a conclusion the problems of this method and improvement potential for the future are pointed out.

2 Motivation / Introduction



B/S/H/

Introduction: Software-Tests

- Need of correct functionality, but every software includes bugs
  - → Tests already during process of development are of advantage
  - → Localization of bugs by testing
- Fulfilling of standards and norms (ISO)
  - → Validation by testing
- Processing a huge amount of data
  - → Effort: automated tests
    - →automated identification and creation of test cases
    - → automated execution of tests
    - → automated generation of test results and documentation

BSH HAUSGERÄTE GRUPPE — Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 4

Abbildung 2.1: *Slide 4: Motivation and Introduction*

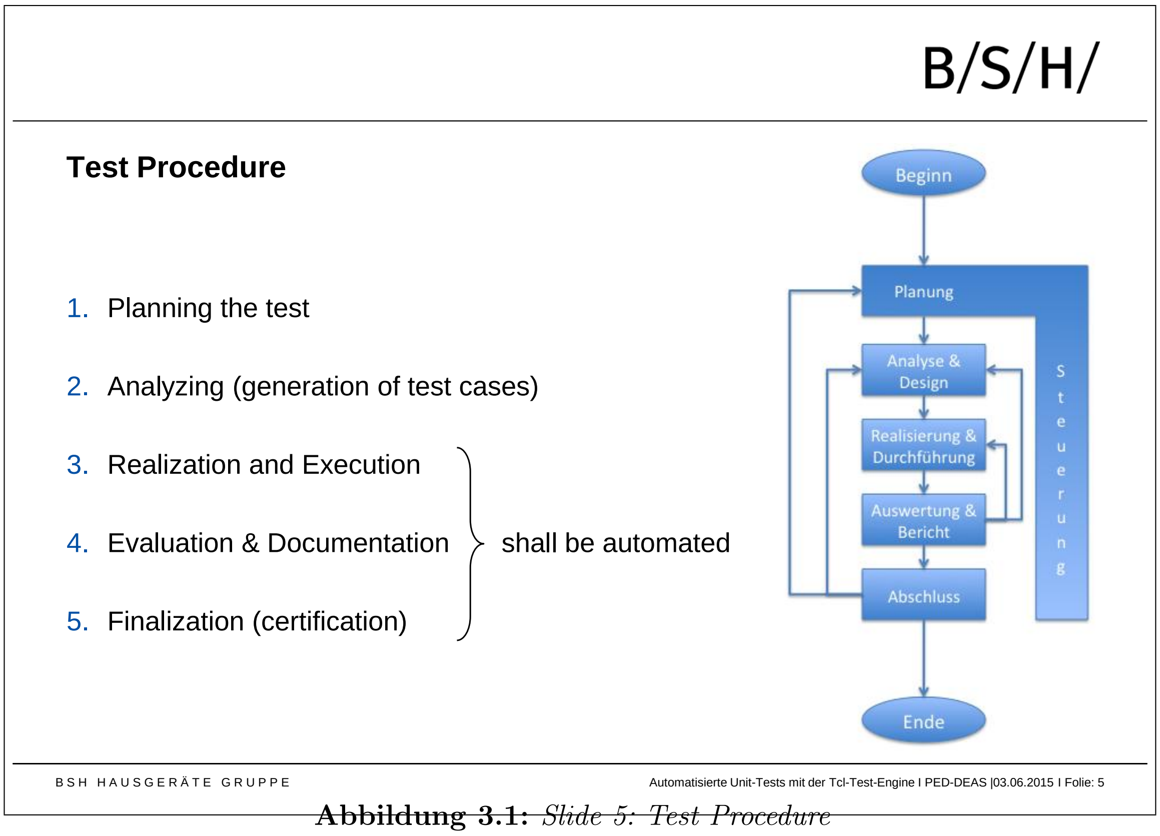
As introduction the need for software tests is made clear.

The most important requirement to software is that the funcionality is correct in any case, whereby bugs appear in a more or less problematic extend. Therefore the functionality has to be tested and confirmed by the developer. The easiest way is that the process of testing attends the process of development, so that the bugs can be localized in a early state of development and fixed directly.

Also the fulfilling of standards and norms, for example ISO-norms, is sometimes required and has to be ensured. Software tests can be used to validate the meeting of all functional requirements.

During the process of development all these software tests have to be executed many times for a huge amount of data . Therefore it is the effort of the industry to automate this task. Especially the identification of test cases, the creation of the test, the execution and the final generation of all documentations (e.g. test results) are the most important steps which should be automated, because they arise repeatedly. Indeed the expenditure of time for programming software code is increasing strongly by coding the software test additionally, but in the long run the productivity in sense of correct code per time is increasing even more (cf. [Köh07], page 193).

3 Project Course


B/S/H/

Test Procedure

1. Planning the test
2. Analyzing (generation of test cases)
3. Realization and Execution
4. Evaluation & Documentation
5. Finalization (certification)

$\left.\begin{matrix}3.\\4.\\5.\end{matrix}\right\}$ shall be automated

Beginn
Planung
Analyse & Design
Realisierung & Durchführung
Auswertung & Bericht
Abschluss
Steuerung
Ende

BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 5

Abbildung 3.1: *Slide 5: Test Procedure*

The test procedure can be divided into five different steps (see [SE15], Abbildung 8), which can be described as follows (cf. [Liu13], page 32):

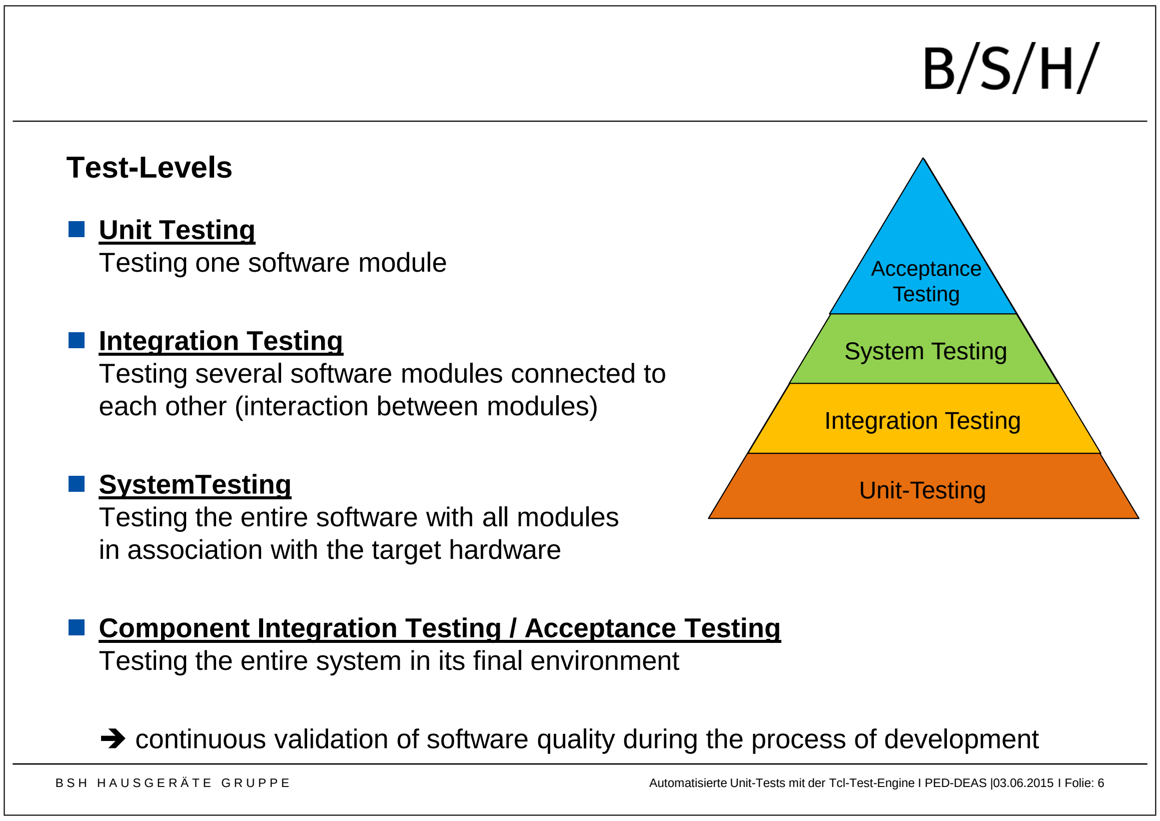
In the first step the software unit test has to be planned. This means that for example the ressources, the timetable and the strategy are planned , but also the test tool is selected.

In the second step the software has to be analyzed and the tests have to be designed. Hereby particularly the analysis of the test scenario and the generation of the different test cases is meant.

In the third step the software test is realized and executed. This means that the test cases are implemented and the test is executed. Thereby the tested software returns an return value.

In the fourth step the test is evaluated and a documentation is created. In this step

the return value is compared to the reference value and as a result a documentation with "passedör "failedïs released.
In the final step a certification is created. This certification confirms that the tester has tested the implied software and that the tests have been passed correctly.



B/S/H/

Test-Levels

- **Unit Testing**
  Testing one software module
- **Integration Testing**
  Testing several software modules connected to each other (interaction between modules)
- **SystemTesting**
  Testing the entire software with all modules in association with the target hardware
- **Component Integration Testing / Acceptance Testing**
  Testing the entire system in its final environment

➔ continuous validation of software quality during the process of development

BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 6

Abbildung 3.2: *Slide 6: Test-Levels*

In the software development software tests are divided into four different levels of testing (cf. [Sch12], slide 4) which can be displayed as a pyramide.

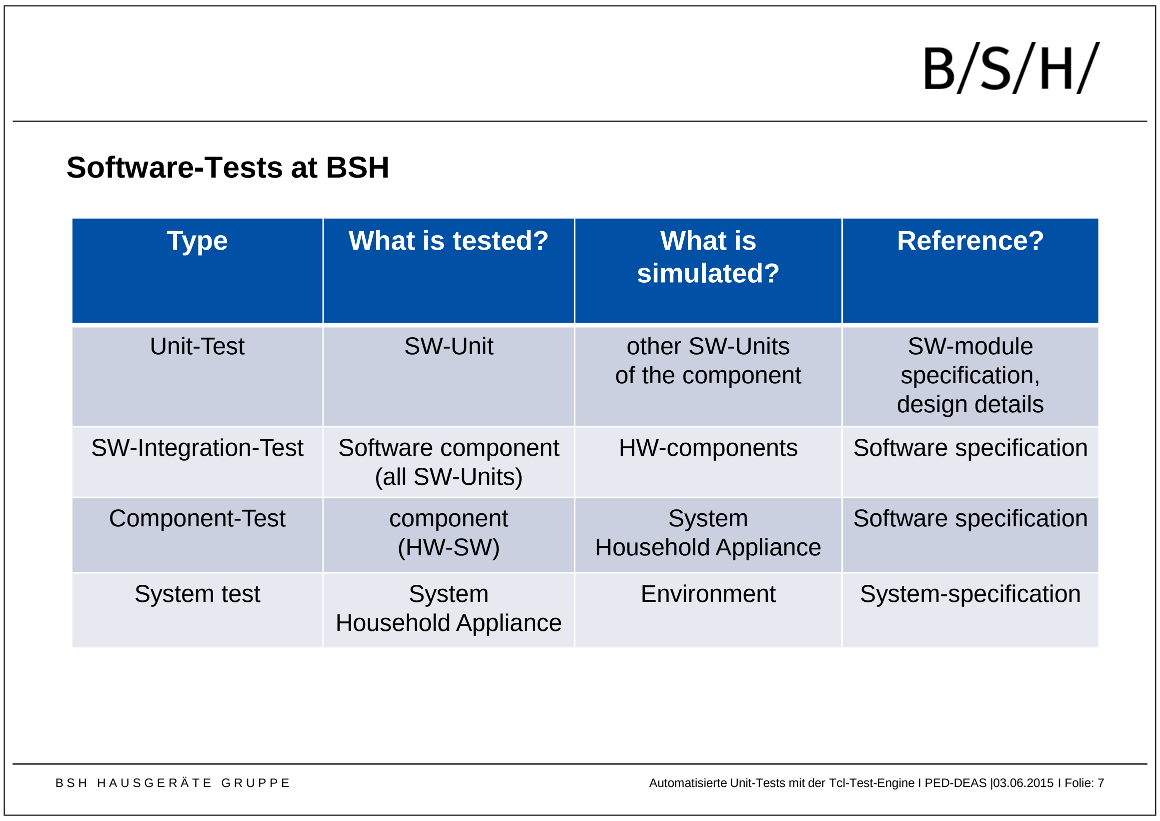
The first level is the unit testing. In this level only one software module is tested. Hence there are existing many different units which build the base of the software, this level is the lowest level and builds the bottom of the pyramide.

The second level is the integration testing. By this test several software units are tested while they are connected to each other, so the interdependencies between all units and the influence of one unit to each other is spotted.

The third level is the the system testing. By this test the entire software with all units is tested in association with the target hardware, so the influence of all units among themselves and problems related to the target hardware are pointed out.

The fourth level is the component integration testing, also . By this test the entire system is tested for acceptance in its final setting, so a correct functionality of the entire software for the final usage can be confirmed.

By all these tests in different levels of software a continuous validation of software quality during the whole process of development can be achieved (cf. [SBB11], page 7). Quality usually refers to six different characteristics: functionality, reliability, usability, efficiency, changeability and portability (cf. [Liu13], page 25-26).



B/S/H/

Software-Tests at BSH

Type	What is tested?	What is simulated?	Reference?
Unit-Test	SW-Unit	other SW-Units of the component	SW-module specification, design details
SW-Integration-Test	Software component (all SW-Units)	HW-components	Software specification
Component-Test	component (HW-SW)	System Household Appliance	Software specification
System test	System Household Appliance	Environment	System-specification

BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 7

Abbildung 3.3: *Slide 7: Software Tests at BSH*

In BSH there are specific definitions given to unify the use of language. In the following the definition of different tests is presented referred to the questions 'What is tested?','What is simulated?' and 'What is the reference?'.

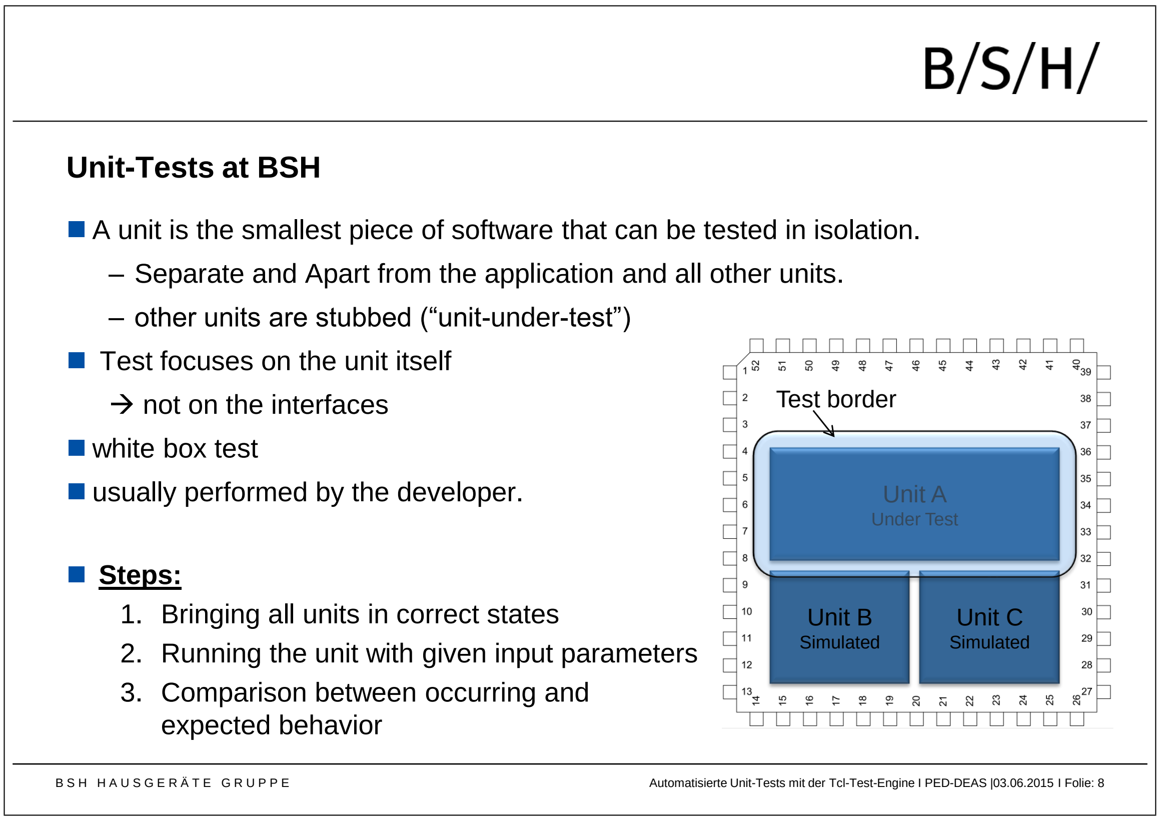
For testing a specific part of a software against its requirement, some other parts have to be simulated (cf. [BSH14], slide 7).

In a unit test one software unit is tested, whereas all other software units of this component are simulated. After testing the return values are compared to the software module specification and the design details (cf. [Grü13], page 6).

In a software integration test the interaction between all software units is tested. Hence this test is not hardware specific, all hardware components are simulated. The test result is referred to the software specification.

In a component test the software is tested in interaction with the target hardware. This test is not depending on the environment, so the system (in our case the household appliance) is simulated. The reference of test result is the given software

specification.
In a system test the entire software is tested in association with the target hardware in its final setting (the complete household appliance), so only the environment has to be simulated. The test result is referred to the system specification.



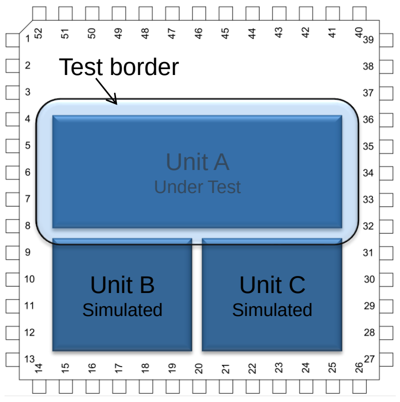
B/S/H/

Unit-Tests at BSH

- A unit is the smallest piece of software that can be tested in isolation.
  - Separate and Apart from the application and all other units.
  - other units are stubbed ("unit-under-test")
- Test focuses on the unit itself

  → not on the interfaces
- white box test
- usually performed by the developer.

- **Steps:**
  1. Bringing all units in correct states
  2. Running the unit with given input parameters
  3. Comparison between occurring and expected behavior



BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 8

Abbildung 3.4: *Slide 8: Unit-Tests at BSH*

There are concrete definitions of software units and unit tests in BSH.

A unit is the smallest piece of a software which can be tested in isolation seperated and apart from the application and all other units. Thereby a isolation mechanism is used, so the unit which shall be tested possesses the attribute ünit-under-test". This means that the unit cannot be called by other units and also cannot call other units itself. Instead of calling the 'original'-function the activation of the isolation mechanism causes the call of the according stub function. By this the test focusses on the unit itself and not on the interfaces or the influences by other units (cf. [Grü13], page 91).

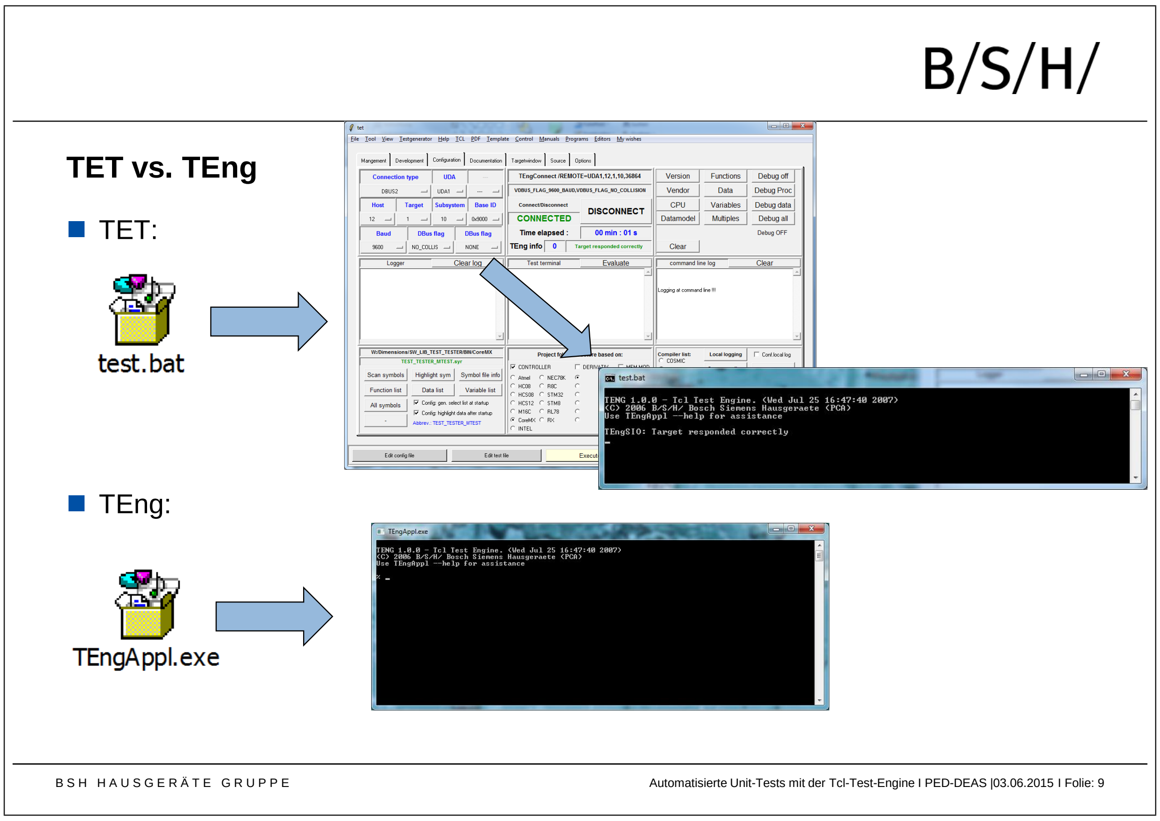
In BSH unit tests are white box tests which are usually perfomed by the developers themselves. This means that the source code and the interfaces of the tested units are known. In case of failure the code can be corrected immediately.

A unit test is executed in 3 steps by a test driver (cf. [Grü13], page 91):

In the first step all units are brought in correct state.

In the second step the unit, which shall be tested, is run with the input parameters defined in the test case.

In the third step the occuring and the expected behavior are compared.



B/S/H/

TET vs. TEng

- TET:

test.bat

```
TENG 1.0.0 - Tcl Test Engine. (Wed Jul 25 16:47:40 2007)
(C) 2006 B/S/H/ Bosch Siemens Hausgeraete (PCA)
Use TEngAppl --help for assistance

TEngSIO: Target responded correctly
_
```

- TEng:

TEngAppl.exe

```
TENG 1.0.0 - Tcl Test Engine. (Wed Jul 25 16:47:40 2007)
(C) 2006 B/S/H/ Bosch Siemens Hausgeraete (PCA)
Use TEngAppl --help for assistance

% _
```

BSH HAUSGERÄTE GRUPPE

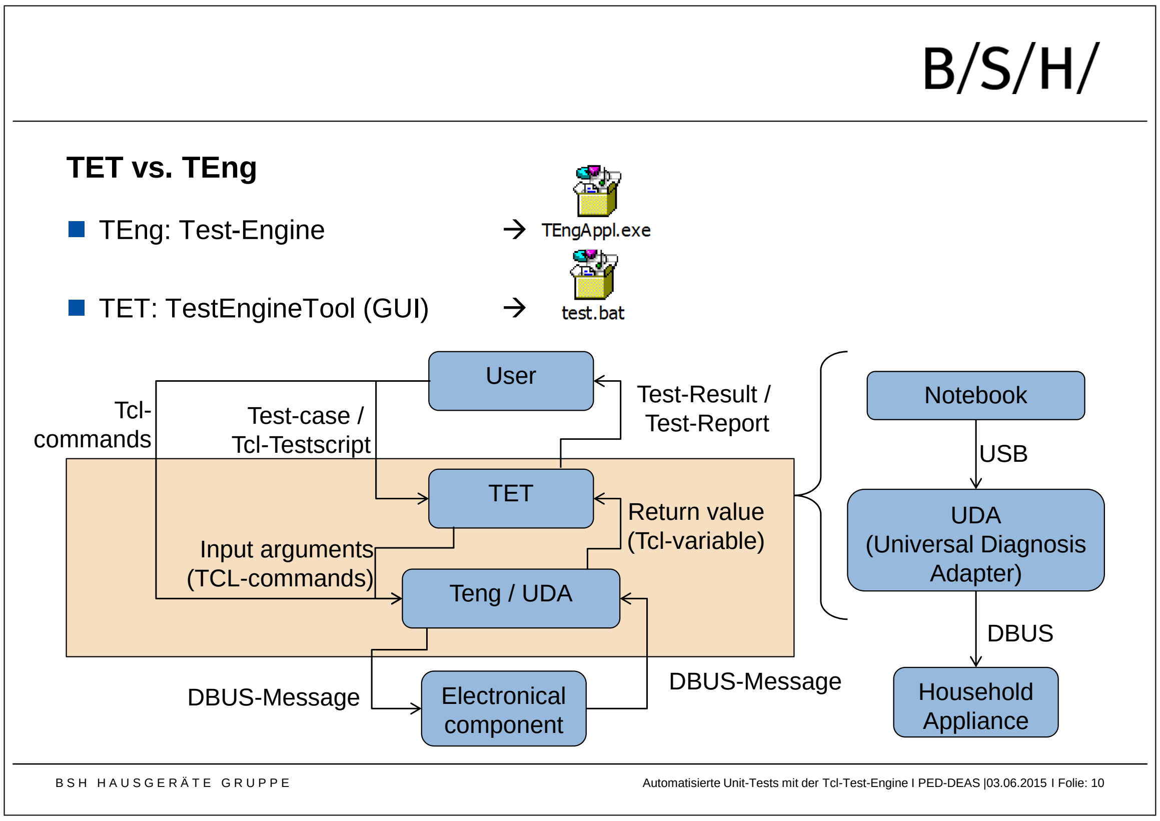
Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 9

Abbildung 3.5: *Slide 9: TET vs. TEng (first part)*

For unit tests in BSH a BSH-own toolchain is used, which consists out of the Test-Engine-Tool (TET) and the Test-Engine (TEng), both based on the scripting language Tcl.

The TEng can be run on the notebook by executing the file TEngAppl.exe, which causes the opening of one Tcl-command-window (Tcl-Interpreter). This Tcl-Interpreter is extended by C-Code and can be seen as a remote control for the target, because it is possible to read variables from the target, write variables to the target and initiate a function call on the target.

The TET can be run by executing the test.bat-file. This causes the opening of a command window and a GUI. In this GUI it is also possible to define test cases, test case specifications or test report specifications, but also the test itself can be executed.



B/S/H/

TET vs. TEng

- TEng: Test-Engine → TEngAppl.exe
- TET: TestEngineTool (GUI) → test.bat

User
Tcl-commands
Test-case / Tcl-Testscript
Test-Result / Test-Report
TET
Return value (Tcl-variable)
Input arguments (TCL-commands)
Teng / UDA
DBUS-Message
Electronical component
DBUS-Message
Notebook
USB
UDA (Universal Diagnosis Adapter)
DBUS
Household Appliance

BSH HAUSGERÄTE GRUPPE   Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 10

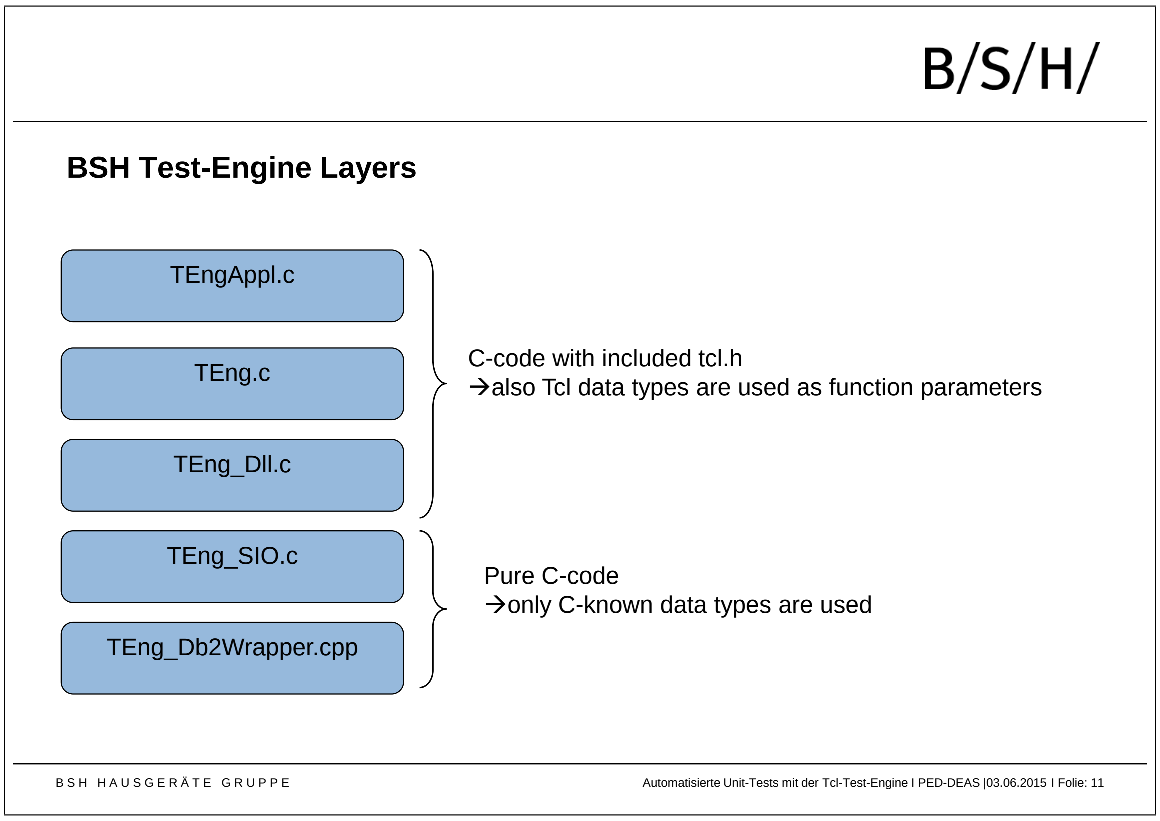
Abbildung 3.6: *Slide 10: TET vs. TEng (second part)*

In the TET the user can specify test cases or a test script. This data is automatically converted into Tcl-commands and passed to the TEng via internal interfaces.

The TEng splits this test case into several commands of either reading or writing variables in the target or initiating a function call in the target. By this all information of the test case can be transferred via D-BUS2 to the electronical component (Target-Hardware) step by step.

In the following a D-BUS2-message is returned from the target-hardware to the TEng, including the return value of the executed function or the requested variable. This value is finally passed to the TET and the test result displayed. Additionally this test result can also be written into a test report.

BSH-unit-tests require different peripheries: On the computer of the developer the TEng-Application or the TET (with included TEng) have to be running. Thereby the TEng is able to communicate with the UDA (Universal Diagnosis Tool). The UDA builds the interface between the notebook and the D-BUS2. In the target-software a message-server, the Test-Message-Server, is implemented which is also able to communicate via DBUS and as a consequence also with the TEng. By this the notebook and the target-hardware can communicate with each other.


B/S/H/

BSH Test-Engine Layers

TEngAppl.c
TEng.c
TEng_Dll.c
C-code with included tcl.h
→also Tcl data types are used as function parameters
TEng_SIO.c
TEng_Db2Wrapper.cpp
Pure C-code
→only C-known data types are used

BSH HAUSGERÄTE GRUPPE

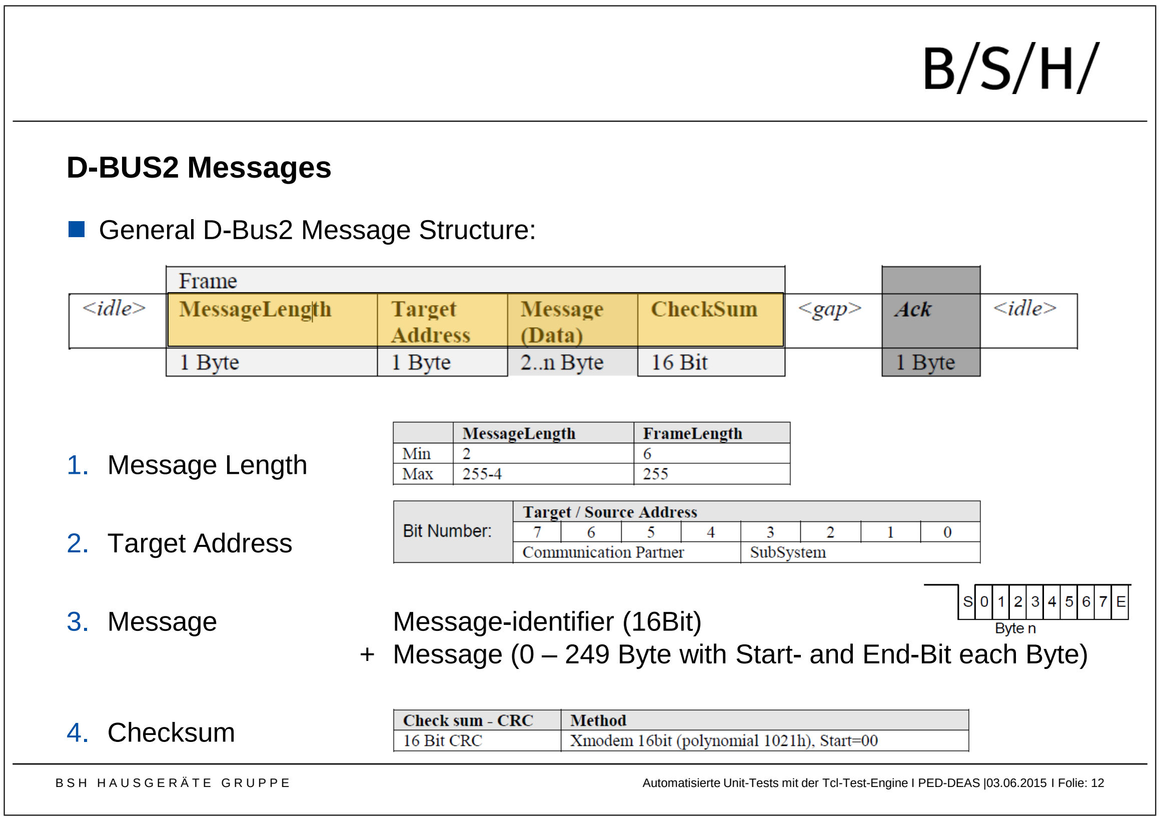
Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 11

Abbildung 3.7: *Slide 11: BSH Test-Engine Layers*

The TEng itself is organized in different layers (cf. [BSH07a]), which can be outlined to two groups: On the one hand layers with pure C-code which build the kernel of the TEng, and on the other hand the layers with C-code extended by including a specific Tcl-Header-file (tcl.h). In this way Tcl-compatibility is created and the interface of the C-Kernel to the Tcl-Interpreter built.

The first group consists of two layers, TEng_SIO and TEng_Db2Wrapper. The TEng-DBUS2-Wrapper is essential for the communication of the TEng via DBUS2. TEng_SIO includes all essential functions and commands for the communication with the target-hardware. In both of them only C-known data types are used.

The first group consists of three layers, TEng_Dll, TEng and TEng_Appl. All these layers have the file "tcl.hïncluded and make use of tcl-specific data types. So tcl.h builds the interface between the programming-languages C and Tcl. In these layers the functionality of all Tcl-specific functions is defined. For example the functions for reading and writing variables in the target are implemented there.



B/S/H/

D-BUS2 Messages

- General D-Bus2 Message Structure:

	Frame						
<idle>	MessageLength	Target Address	Message (Data)	CheckSum	<gap>	Ack	<idle>
	1 Byte	1 Byte	2..n Byte	16 Bit		1 Byte	

1. Message Length

	MessageLength	FrameLength
Min	2	6
Max	255-4	255

2. Target Address

Bit Number:	Target / Source Address							
	7	6	5	4	3	2	1	0
	Communication Partner				SubSystem			

3. Message

Message-identifier (16Bit)
+ Message (0 – 249 Byte with Start- and End-Bit each Byte)

S	0	1	2	3	4	5	6	7	E

Byte n

4. Checksum

Check sum - CRC	Method
16 Bit CRC	Xmodem 16bit (polynomial 1021h), Start=00

BSH HAUSGERÄTE GRUPPE   Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 12

Abbildung 3.8: *Slide 12: D-BUS2 Messages*

The structure of DBUS2 messages is defined by BSH-standards (see [Hol12], cf. chapter 4).

One Frame consists of 4 different parts: Message-Length, Target-Adress, Message and Check-Sum.

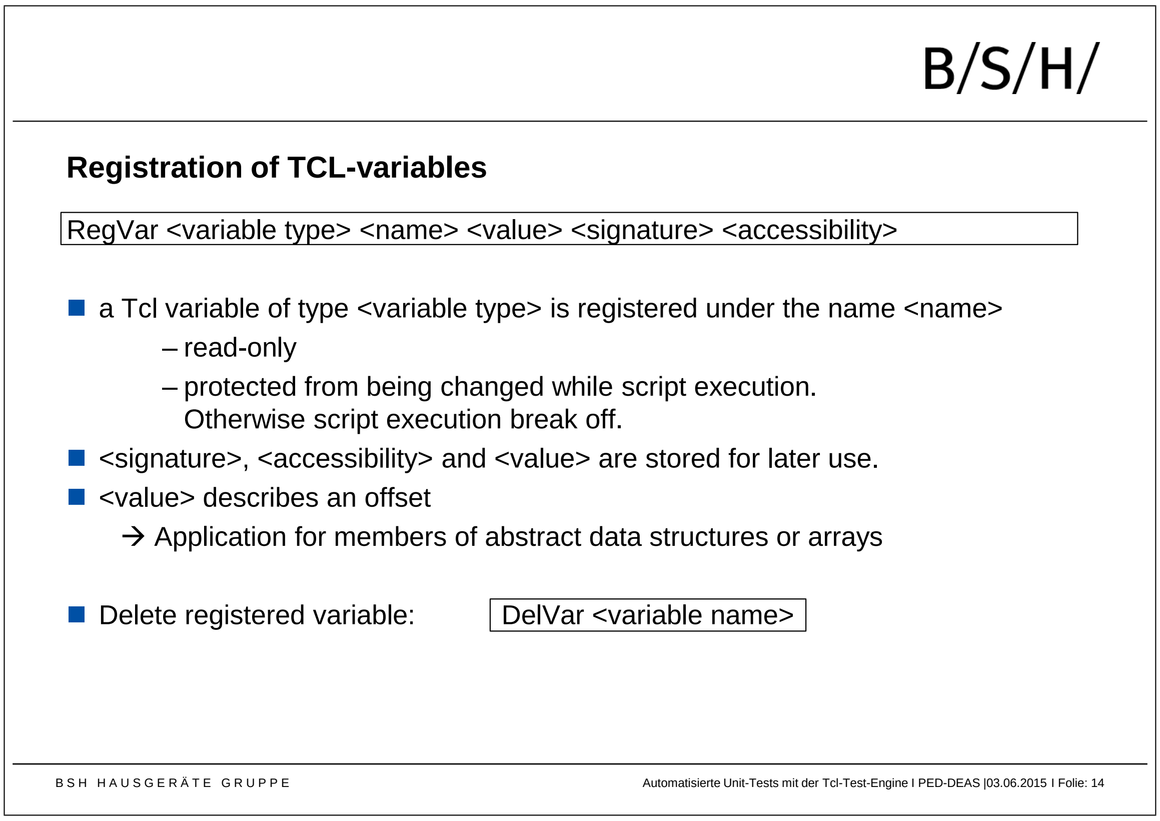
The Message-Length is written into 1 Byte and describes the lenght of the entire frame in bytes. Caused by the lenght of 8 Bit a maximum frame-length of $2^8 - 1 = 255$ Bytes is possible.

The Target adress is also written into 1 Byte and consists of the communication-partner (4 Bits) and the SubSystem (4 Bits).

The message consists of a message identifier (2 Byte) and the message itself. Thereby the message identifier is an uint16 value divided into different subranges, with one specific subrange for test messages.

The CheckSum is written into 2 Bytes and is calculated by CRC (Cyclic Redundancy Check).

As a conclusion it can be summarized that the message itself can have a size between 0 and $255 - 1 - 1 - 2 - 2 = 249$ Bytes (1 Byte Lenghth, 1 Byte Address, 2 Bytes Message-Identifier, 2 Bytes Check-Sum).



B/S/H/

Registration of TCL-variables

RegVar <variable type> <name> <value> <signature> <accessibility>

- a Tcl variable of type <variable type> is registered under the name <name>
  - read-only
  - protected from being changed while script execution. Otherwise script execution break off.
- <signature>, <accessibility> and <value> are stored for later use.
- <value> describes an offset
  → Application for members of abstract data structures or arrays
- Delete registered variable: DelVar <variable name>

BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 14

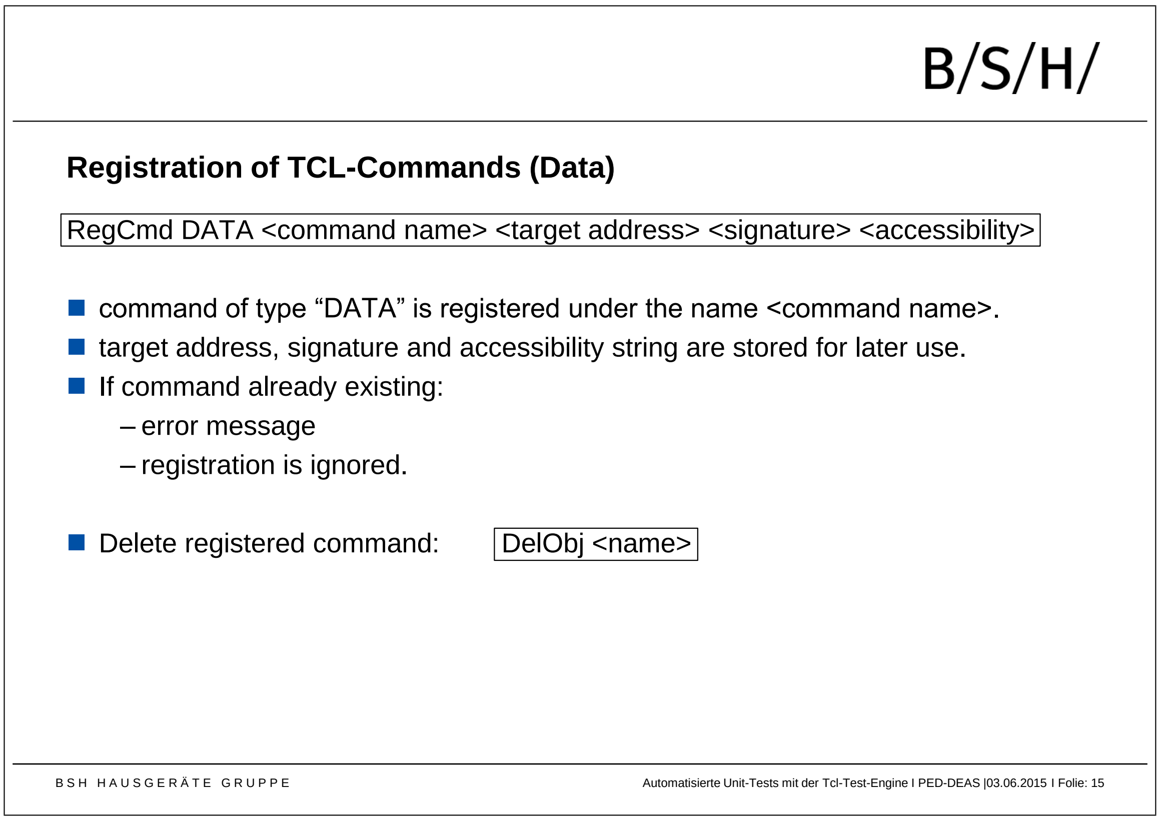
Abbildung 3.9: *Slide 14: Registration of TCL-variables*

In the TEng a Tcl-function called RegVar is defined. As parameters the variable type, the variable name, the initial value, the signature and the accessibility are needed.

The Tcl-variable of the given type is registered under the given variable name. This variable is read-only and protected from being changed while script execution. Besides the initial value also the accessibility and the signature are stored for later use, whereby the initial value describes an offset.

This kind of registration of variables is applied for members of data structures or arrays to be able to access the member in the memory.

To delete a registered variable the Tcl-function DelVar can be used.



B/S/H/

Registration of TCL-Commands (Data)

RegCmd DATA <command name> <target address> <signature> <accessibility>

- command of type “DATA” is registered under the name <command name>.
- target address, signature and accessibility string are stored for later use.
- If command already existing:
  - error message
  - registration is ignored.
- Delete registered command: DelObj <name>

BSH HAUSGERÄTE GRUPPE Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 15

Abbildung 3.10: *Slide 15: Registration of TCL-Commands (Data)*

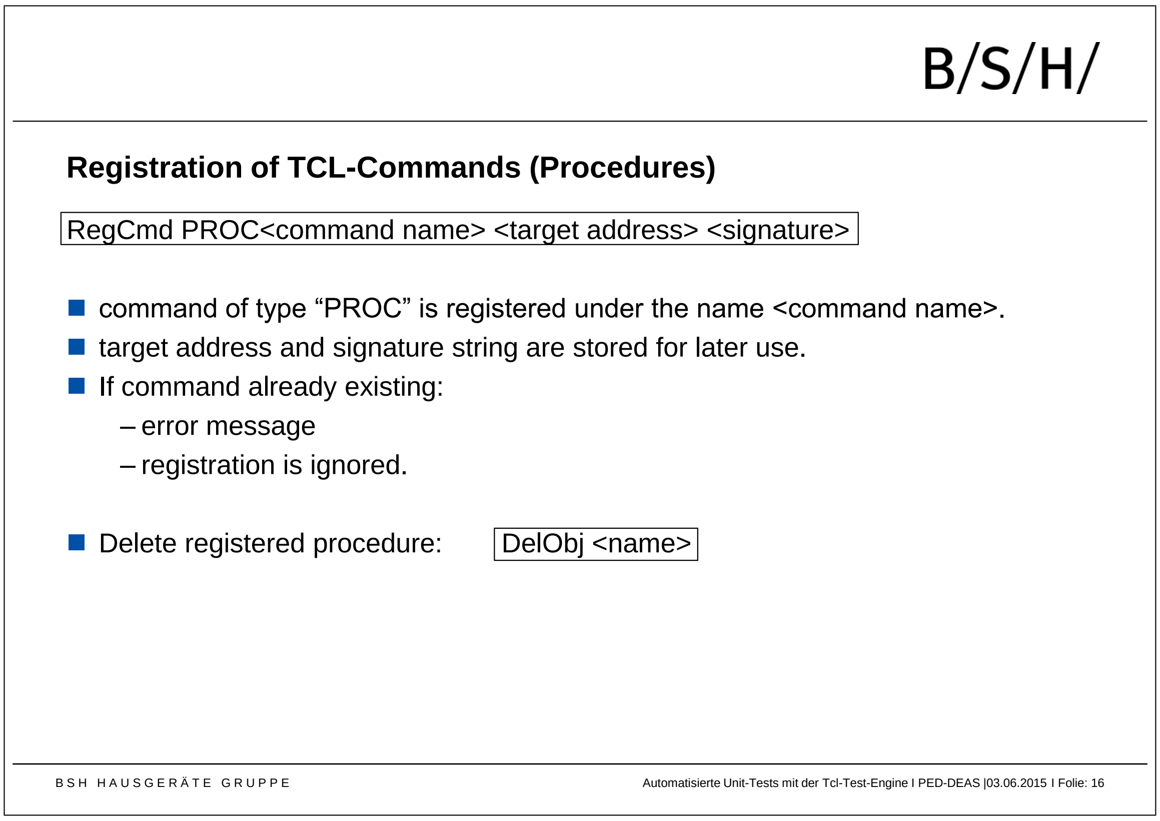
In the TEng also a Tcl-function called RegCmd is defined. As parameters the command type (here DATA), the command name, the target address, the signature and the accessibility are needed.

The Tcl-variable of type DATA is registered under the given command name. Besides the target address also the signature and the accessibility are stored for later use.

If the command name already exists, a error message is shown and the registration ignored.

This kind of registration is used to specify the memory adress of a C-variable in the target. By using the registered DATA-name, the value in the memory of the target is used instead of a local value stored in the local Tcl-data-table. By this it is possible to access the target for both reading and writing.

To delete a registered command the Tcl-function DelObj can be used.


B/S/H/

Registration of TCL-Commands (Procedures)

RegCmd PROC<command name> <target address> <signature>

- command of type "PROC" is registered under the name <command name>.
- target address and signature string are stored for later use.
- If command already existing:
  - error message
  - registration is ignored.

- Delete registered procedure: DelObj <name>

BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 16

Abbildung 3.11: *Slide 16: Registration of TCL-Commands (Procedures)*

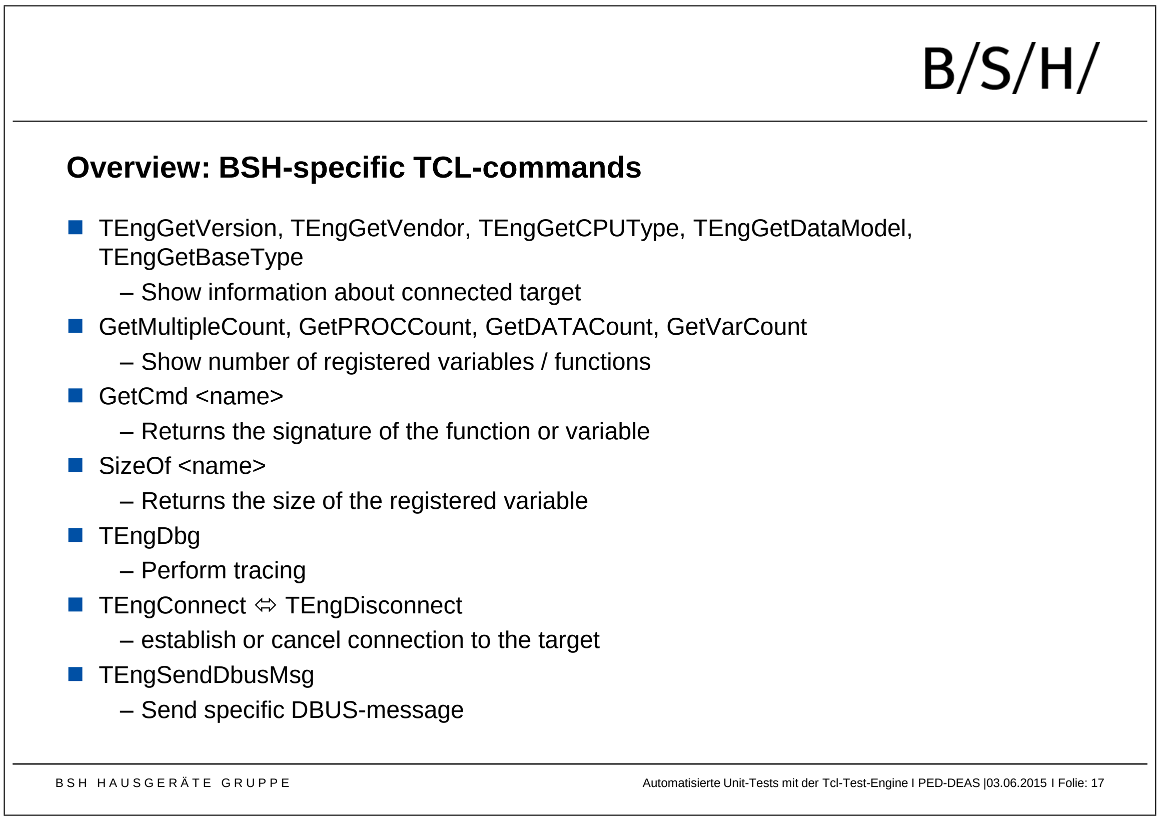
With the aid of the function RegCmd it is also possible to register a new Tcl-Procedure. As parameters the command type (here PROC), the command name, the target address and the signature are needed.

The Tcl-procedure is registered under the given command name. Besides the target address also the signature is stored for later use. The string is divided into two parts, separated by an underline "_". The first part before this underline defines the kind of return value, the second part defines the kind of input parameters, e.g. $B1\_B2U2$.

If the command name already exists, a error message is shown and the registration ignored.

This kind of registration is used to specify the memory adress of a C-function in the target. By calling the registered PROC-name, the procedure is executed on the target instead of the local procedure.

To delete a registered command the Tcl-function DelObj can be used.


B/S/H/

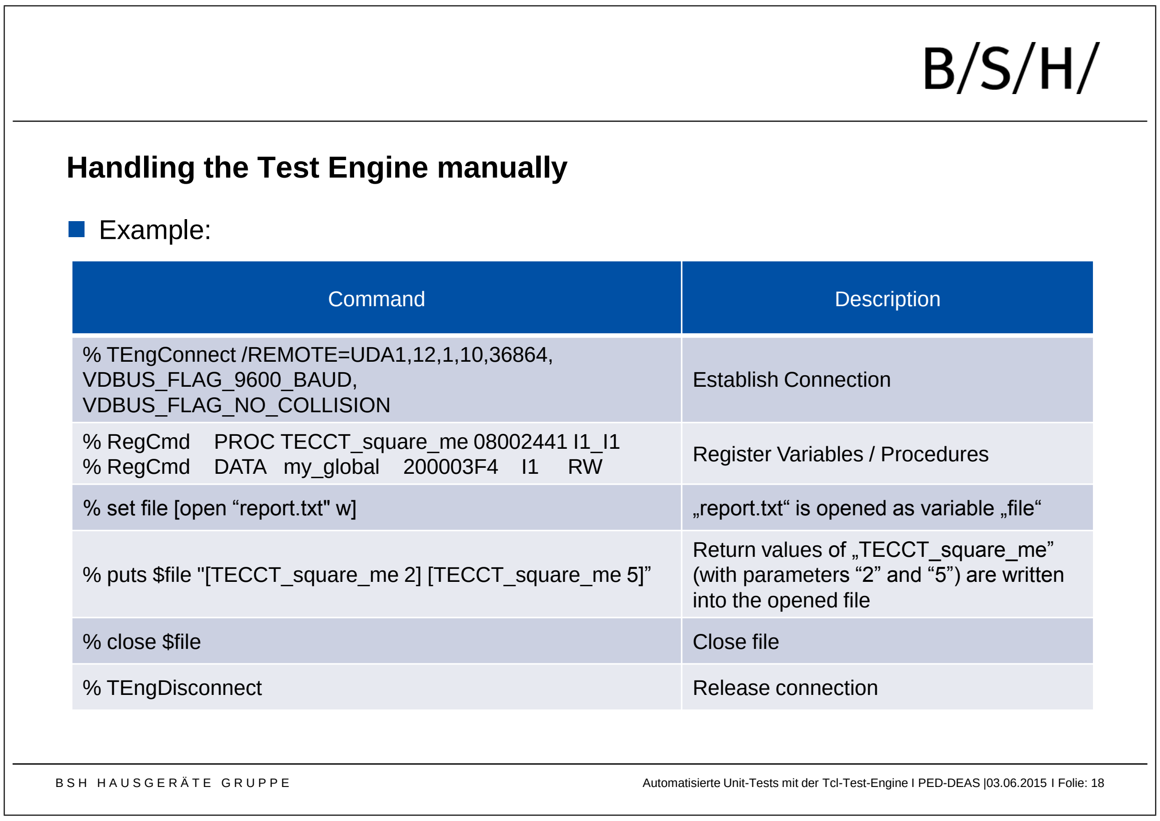
Overview: BSH-specific TCL-commands

- TEngGetVersion, TEngGetVendor, TEngGetCPUType, TEngGetDataModel, TEngGetBaseType
  – Show information about connected target
- GetMultipleCount, GetPROCCount, GetDATACount, GetVarCount
  – Show number of registered variables / functions
- GetCmd <name>
  – Returns the signature of the function or variable
- SizeOf <name>
  – Returns the size of the registered variable
- TEngDbg
  – Perform tracing
- TEngConnect ⇔ TEngDisconnect
  – establish or cancel connection to the target
- TEngSendDbusMsg
  – Send specific DBUS-message

BSH HAUSGERÄTE GRUPPE   Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 17

Abbildung 3.12: *Slide 17: Overview of all BSH-specific TCL-commands*

There are several different BSH-specific Tcl-functions registered which are necessary to communicate with target hardware via TEng.
With the Tcl-function TEngConnect and TEngDisconnect it is possible to establish or cancel a connection to a target.
Some Tcl-functions only show information about the connected target (TEngGetVersion, TEngGetVendor, TEngGetCPUType, TEngGetDataModel, TEngGetBaseTy-pe), while other Tcl-functions show the number of currently BSH-specific registered variables / commands (GetPROCCount, GetDATACount, GetVarCount, GetMultipleCount).
The Tcl-function GetCmd returns the signature-string of a BSH-specific registered command (DATA or PROC), whereas the function SizeOf <variablename> returns the size of an registered variable <variablename>.
With the aid of the Tcl-function TEngSendDbusMsg it is possible to send an own specific DBUS message.


B/S/H/

Handling the Test Engine manually

- Example:

Command	Description
% TEngConnect /REMOTE=UDA1,12,1,10,36864, VDBUS_FLAG_9600_BAUD, VDBUS_FLAG_NO_COLLISION	Establish Connection
% RegCmd PROC TECCT_square_me 08002441 I1_I1 % RegCmd DATA my_global 200003F4 I1 RW	Register Variables / Procedures
% set file [open "report.txt" w]	„report.txt" is opened as variable „file"
% puts $file "[TECCT_square_me 2] [TECCT_square_me 5]"	Return values of „TECCT_square_me" (with parameters "2" and "5") are written into the opened file
% close $file	Close file
% TEngDisconnect	Release connection

BSH HAUSGERÄTE GRUPPE Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 18

Abbildung 3.13: *Slide 18: Handling the Test Engine manually*

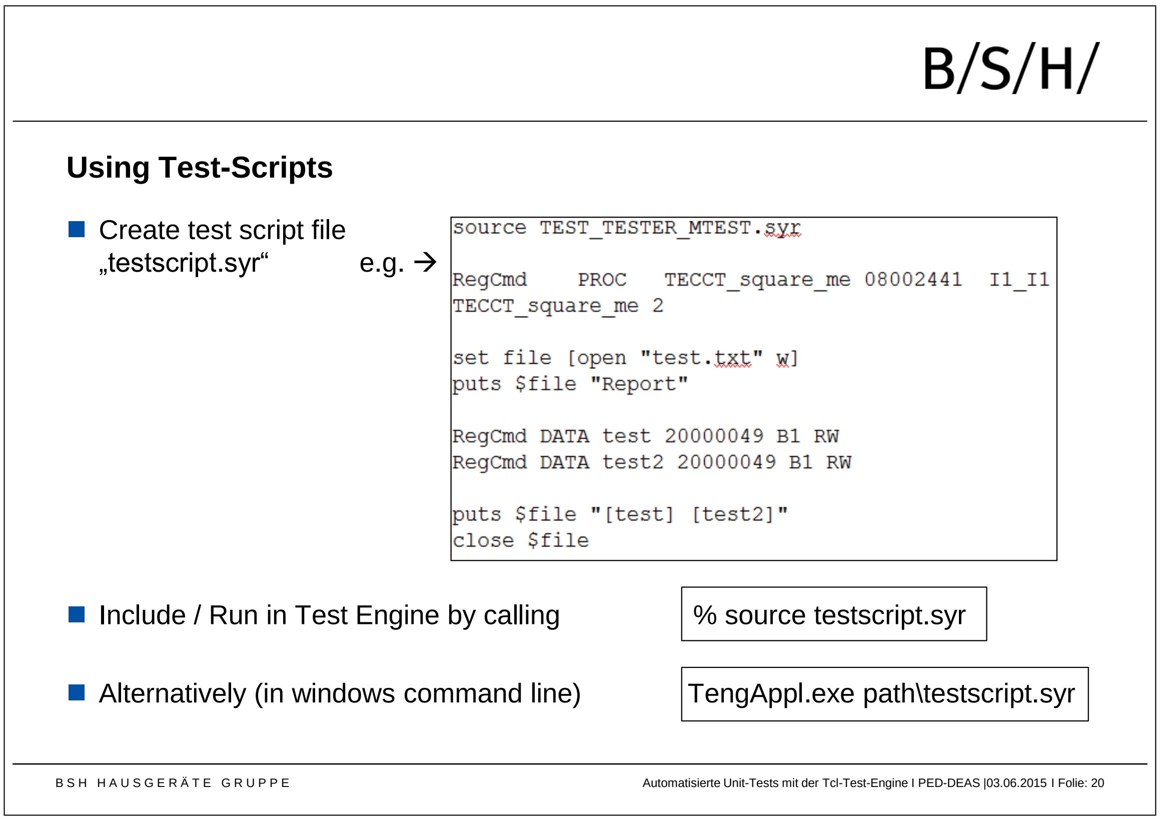
In this slide the manual handling of the TEng is shown.

In thee first step the connection to the target is established, whereby various parameters are set (e.g. connection via UDA1,...).

In the second step a procedure named "TECCT_square_meïs registered at the address 08002441 with the signature I1_I1. This signature means that the return value is an Integer with size 1 Byte, and that is has one input parameter (Also Integer with size 1 Byte). Also the variable (DATA) "my_globalïs registered at the address 200003F4. It has the type I1, which means Integer with 1 Byte. The accessibility is RW, which means readable-writeable.

In the next steps the variable "fileïs set as "report.txt"with the attribute writeable. By "puts $file «string> the given string is written into the variable "file", hence "report.txt".

After this the variable "fileïs closed and the connection released.


B/S/H/

Using Test-Scripts

- Create test script file „testscript.syr“ e.g. →

```
source TEST_TESTER_MTEST.syr

RegCmd    PROC   TECCT_square_me 08002441  I1_I1
TECCT_square_me 2

set file [open "test.txt" w]
puts $file "Report"

RegCmd DATA test 20000049 B1 RW
RegCmd DATA test2 20000049 B1 RW

puts $file "[test] [test2]"
close $file
```

- Include / Run in Test Engine by calling

```
% source testscript.syr
```

- Alternatively (in windows command line)

```
TengAppl.exe path\testscript.syr
```

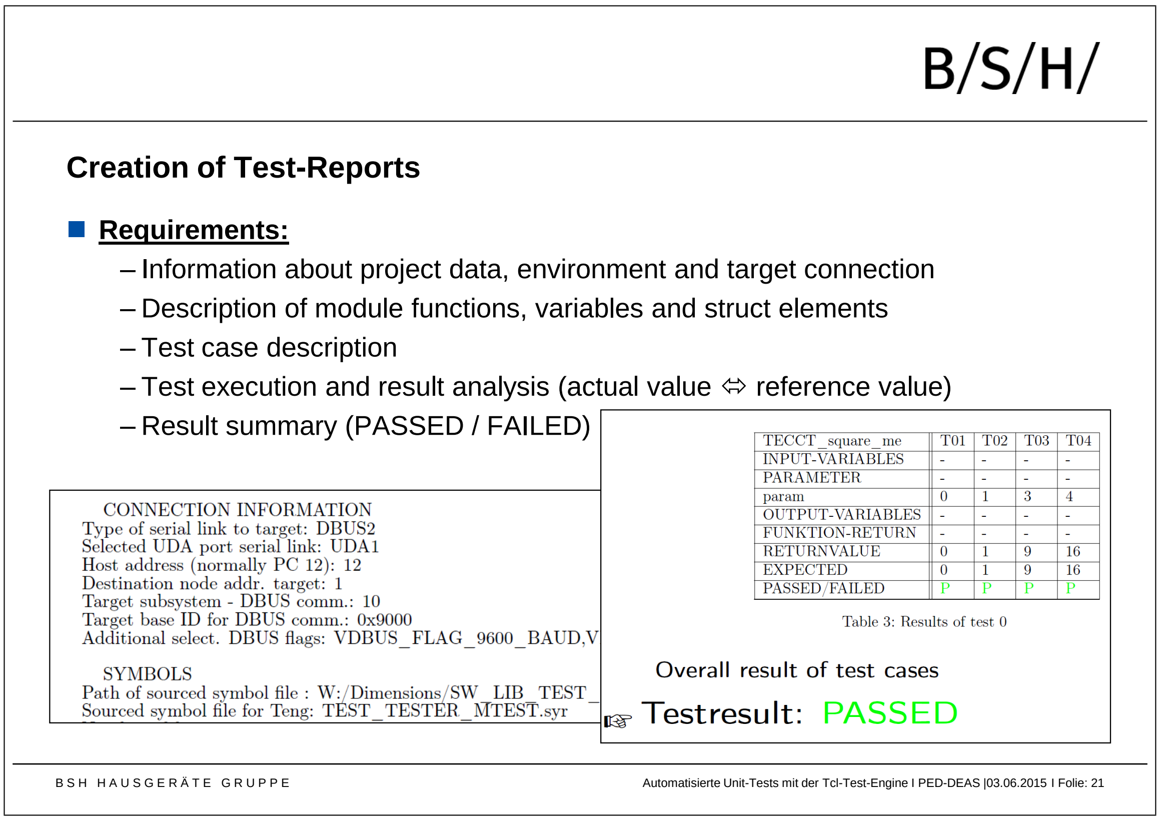
BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 20

Abbildung 3.14: *Slide 20: Using Test-Scripts*

The usage of a test script file (e.g. testscript.syr) is also possible in the TEng, in case that it is written in Tcl. By running the Tcl-command ßource testscript.syr"the given file is opened and executed line by line (cf. [BSH07b], page 10).

Alternatively the TEng-Application (TengAppl.exe) can be started with the path to the testscript as input value.



B/S/H/

Creation of Test-Reports

- **Requirements:**
  - Information about project data, environment and target connection
  - Description of module functions, variables and struct elements
  - Test case description
  - Test execution and result analysis (actual value ⇔ reference value)
  - Result summary (PASSED / FAILED)

```
   CONNECTION INFORMATION
Type of serial link to target: DBUS2
Selected UDA port serial link: UDA1
Host address (normally PC 12): 12
Destination node addr. target: 1
Target subsystem - DBUS comm.: 10
Target base ID for DBUS comm.: 0x9000
Additional select. DBUS flags: VDBUS_FLAG_9600_BAUD,V

   SYMBOLS
Path of sourced symbol file : W:/Dimensions/SW_LIB_TEST_
Sourced symbol file for Teng: TEST_TESTER_MTEST.syr
```

TECCT_square_me	T01	T02	T03	T04
INPUT-VARIABLES	-	-	-	-
PARAMETER	-	-	-	-
param	0	1	3	4
OUTPUT-VARIABLES	-	-	-	-
FUNKTION-RETURN	-	-	-	-
RETURNVALUE	0	1	9	16
EXPECTED	0	1	9	16
PASSED/FAILED	P	P	P	P

Table 3: Results of test 0

Overall result of test cases

☞ Testresult: PASSED

B S H H A U S G E R Ä T E G R U P P E

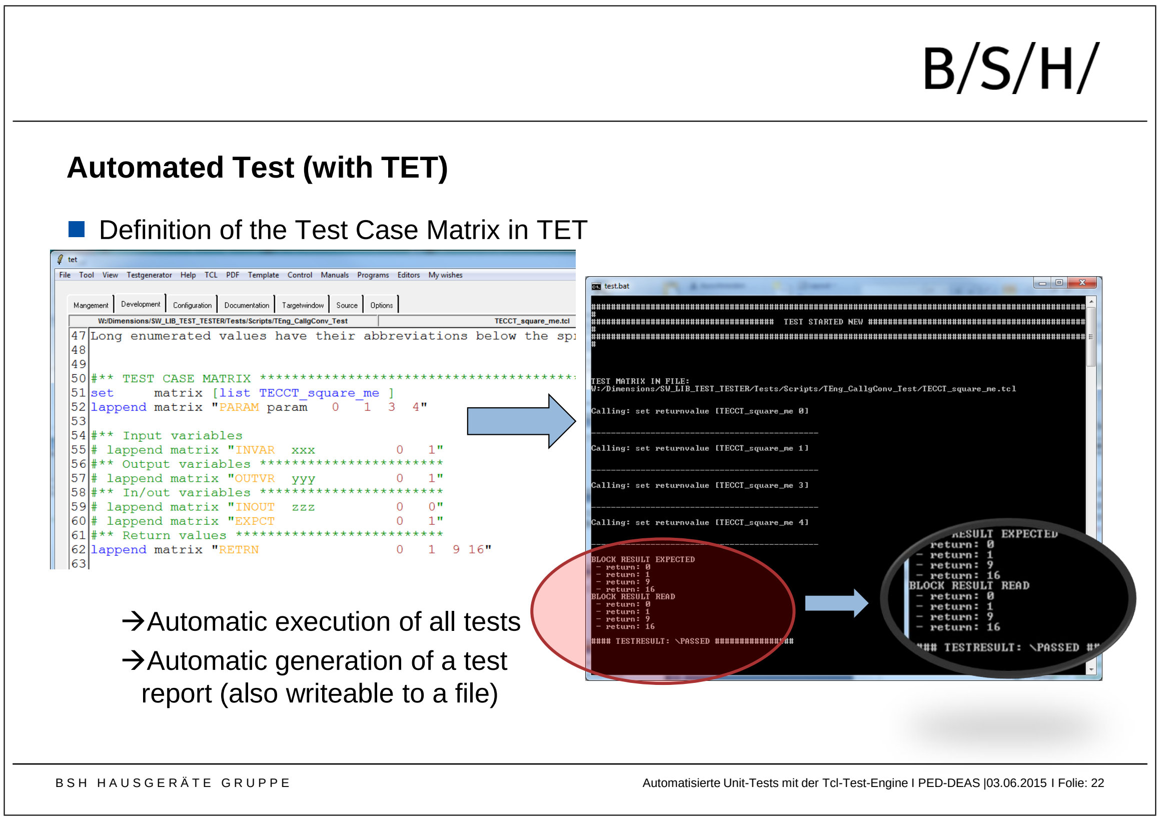
Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 21

Abbildung 3.15: *Slide 21: Creation of Test-Reports*

There are existing several requirements to test reports which have to be met.

On the one hand all information about the project data, the setting and the target connection have to be pointed out, including the description of all used functions, variables and struct-elements.

But on the other hand also all information about the test case and the test execution especially the test result and analysis have to be indicated. This means that the current actual-value has to be compared to the expected value, so that in the end a test result can be represented (PASSED / FAILED).



B/S/H/

Automated Test (with TET)

- Definition of the Test Case Matrix in TET

tet
File Tool View Testgenerator Help TCL PDF Template Control Manuals Programs Editors My wishes
Mangement Development Configuration Documentation Targetwindow Source Options
W:/Dimensions/SW_LIB_TEST_TESTER/Tests/Scripts/TEng_CallgConv_Test
TECCT_square_me.tcl

```
Long enumerated values have their abbreviations below the spı


#** TEST CASE MATRIX ****************************************
set     matrix [list TECCT_square_me ]
lappend matrix "PARAM param   0   1  3  4"

#** Input variables
# lappend matrix "INVAR  xxx           0   1"
#** Output variables ************************
# lappend matrix "OUTVR  yyy           0   1"
#** In/out variables ************************
# lappend matrix "INOUT  zzz           0   0"
# lappend matrix "EXPCT                0   1"
#** Return values **************************
lappend matrix "RETRN                  0   1  9 16"

```

```
################################################################################
#
####################################  TEST STARTED NEW ##########################
#
################################################################################
#



TEST MATRIX IN FILE:
W:/Dimensions/SW_LIB_TEST_TESTER/Tests/Scripts/TEng_CallgConv_Test/TECCT_square_me.tcl


Calling: set returnvalue [TECCT_square_me 0]

---------------------------------------------

Calling: set returnvalue [TECCT_square_me 1]

---------------------------------------------

Calling: set returnvalue [TECCT_square_me 3]

---------------------------------------------

Calling: set returnvalue [TECCT_square_me 4]

---------------------------------------------
BLOCK RESULT EXPECTED
 - return: 0
 - return: 1
 - return: 9
 - return: 16
BLOCK RESULT READ
 - return: 0
 - return: 1
 - return: 9
 - return: 16

#### TESTRESULT: \PASSED ################
```

```
 RESULT EXPECTED
  return: 0
 - return: 1
 - return: 9
 - return: 16
BLOCK RESULT READ
 - return: 0
 - return: 1
 - return: 9
 - return: 16

### TESTRESULT: \PASSED ##
```

→Automatic execution of all tests
→Automatic generation of a test report (also writeable to a file)

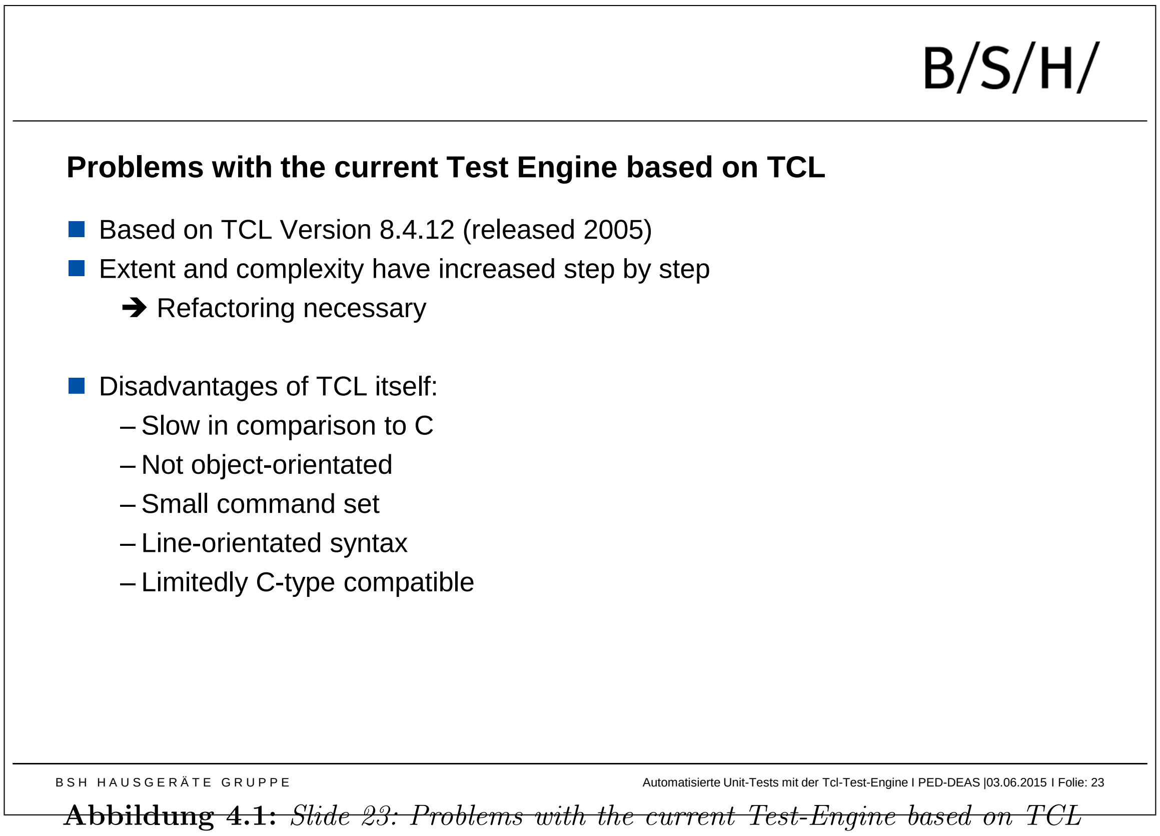
BSH HAUSGERÄTE GRUPPE

Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 22

Abbildung 3.16: *Slide 22: Automated Testing (with TET)*

For the automation of this sequence also a test case matrix can be defined in the TET. With the aid of this matrix an automated test execution can be run. Thereby also a test report is automatically generated and shown in the TEng, but it is also writeable to a file.

4 Result



B/S/H/

Problems with the current Test Engine based on TCL

- Based on TCL Version 8.4.12 (released 2005)
- Extent and complexity have increased step by step
  ➔ Refactoring necessary
- Disadvantages of TCL itself:
  – Slow in comparison to C
  – Not object-orientated
  – Small command set
  – Line-orientated syntax
  – Limitedly C-type compatible

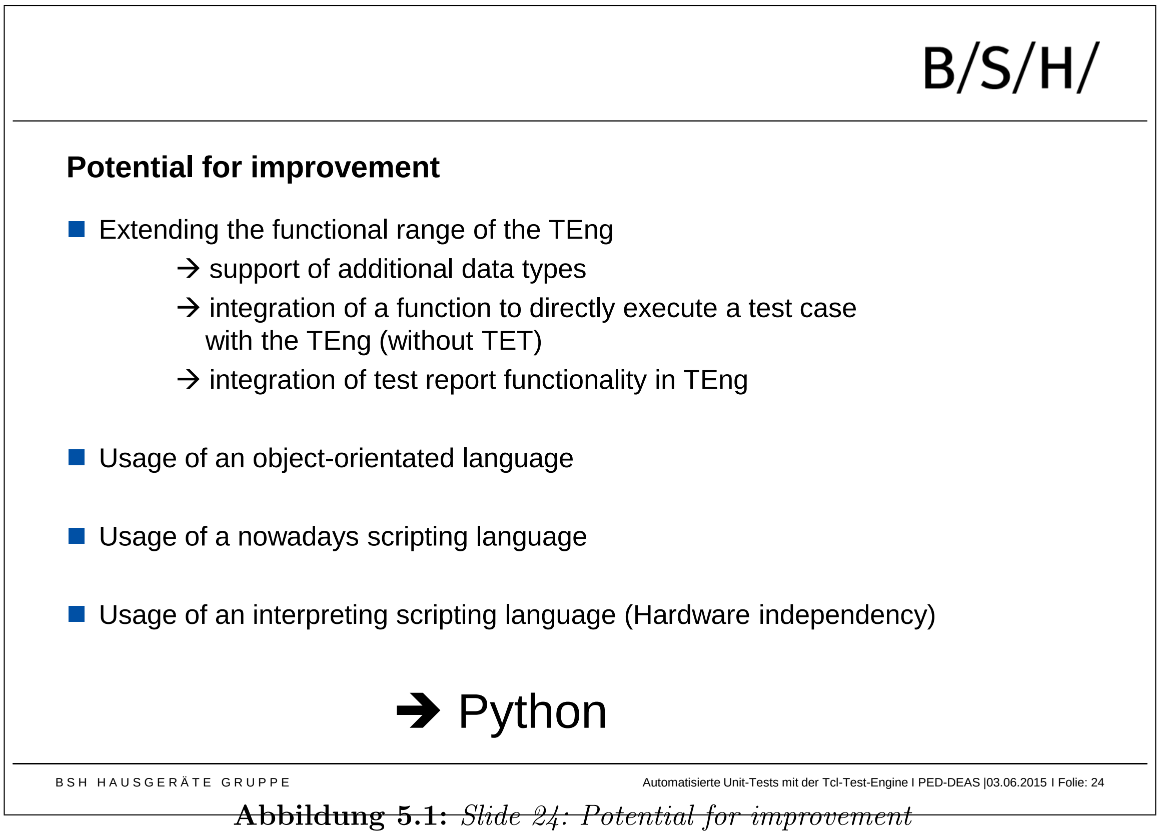
BSH HAUSGERÄTE GRUPPE   Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 23

Abbildung 4.1: *Slide 23: Problems with the current Test-Engine based on TCL*

The problem of the TET and TEng are on the one hand, that the application is based on Tcl Version 8.4.12 which was released in 2005. Since then a lot of changes have been made in TET and TEng, so the extent and complexity have continuously increased. Hence a refactoring of the software is necessary in any case.

Thereby there is a need to reflect if the refactored software should be based on TCL again or if it is changed to another scripting language. By this also disadvantages of TCL itself could be fixed. These are particularly that Tcl is slow in comparison to C (for example in loops), that Tcl has line-orientated syntax and that Tcl is not object-orientated (c.f.[Neu15]). Furthermore it has only a small command set and is only partially compatible to the programming language C, which is needed to communicate with the TEng message server on the target.

5 Potential for Improvement



B/S/H/

Potential for improvement

- Extending the functional range of the TEng
  - → support of additional data types
  - → integration of a function to directly execute a test case with the TEng (without TET)
  - → integration of test report functionality in TEng
- Usage of an object-orientated language
- Usage of a nowadays scripting language
- Usage of an interpreting scripting language (Hardware independency)

➔ Python

BSH HAUSGERÄTE GRUPPE Automatisierte Unit-Tests mit der Tcl-Test-Engine I PED-DEAS |03.06.2015 I Folie: 24

Abbildung 5.1: *Slide 24: Potential for improvement*

As potential for improvement some different points can be mentioned.
Firstly the functional range of the TEng should be extended. This means that additional data types (e.g. float) should be supported or a function to execute a test case directly without the need of the TET should be implemented, so a result (passed/failed) is returned by the TEng directly. Also the functionality to create a complete test report with only the TEng should be integrated.
But also the usage of an object-orientated scripting-language with an interpreter should be preferred. By object-orientation a high degree of reusability can be ensured. A scripting language with an interpreter ensures independency from the hardware. By this it is also possible to publish only the test scripts with small filesize instead of compiled code as exe-file with bigger filesize for each system.

Abbildungsverzeichnis

Literatur

[BSH07a] Peter Cambeis (BSH-intern). *The Test Engine - Book One*. BSH-Intranet, 2007 (siehe S. 16).

[BSH07b] Peter Cambeis (BSH-intern). *The Test Engine - Book Two*. BSH-Intranet, 2007 (siehe S. 24).

[BSH14] Jens Lehmann (BSH-intern). *Test Levels: Workshop Automated Testing*. BSH-Intranet, 2014 (siehe S. 9).

[Grü13] Stephan Grünfelder. *Software-Test für Embedded Systems. Ein Praxishandbuch für Entwickler, Tester und technische Projektleiter*. dpunkt.verlag, 2013. ISBN: 978-3-86490-048-8 (siehe S. 9, 11).

[Hol12] Rune Holen. *BSH General Bus Specification D-Bus-2*. BSH Hausgeräte GmbH. BSH-Intranet, 2012 (siehe S. 17).

[Köh07] Achim Köhler. *Der C/C++-Projektbegleiter. C/C++ Projekte planen, dokumentieren, bauen und testen*. dpunkt.verlag, 2007. ISBN: 978-3-89864-470-9 (siehe S. 4).

[Liu13] Huimei Liu. „Entwicklung einer Vorgehensweise zur Testautomatisierung im Rahmen des PDM am Beispiel der HELLA KGaA Hueck & Co.“ Magisterarb. Universität Stuttgart: Fakultät Informatik, Elektrotechnik und Informationstechnik, Mai 2013 (siehe S. 5, 8).

[Neu15] Gustaf Neumann. *Objekt-orientiertes Design und Systementwicklung: TCL*. WU Wirtschaftsuniversität Wien. 10. Mai 2015. URL: http://wi.wu-wien.ac.at/studium/Abschnitt_2/LVA_ws99/neumann/oo/einheit3/03-9.html (siehe S. 27).

[SBB11] Richard Seidl, Manfred Baumgartner und Thomas Bucsics. *Basiswissen Testautomatisierung: Konzepte, Methoden und Techniken*. dpunkt.verlag, 2011. ISBN: 978-3-89864-724-3 (siehe S. 8).

[Sch12] Dr. Andreas Schröder. *Unit Testing mit JUnit*. Ludwig-Maximilian-Universität München. 2012. URL: http://www.pst.ifi.lmu.de/Lehre/wise-12-13/sep/06-unit-testing.pdf (siehe S. 7).

[SE15] Marco Stark und David Espin. *Modellbasiertes Testen*. FOM Essen. 10. Mai 2015. URL: http://winfwiki.wi-fom.de/index.php/Modellbasiertes_Testen (siehe S. 5).