Kompaktseminar

Freie Unittest-Tools für die C-Entwicklung

07. Dezember 2010

Freie Unittesttools für die C-Entwicklung

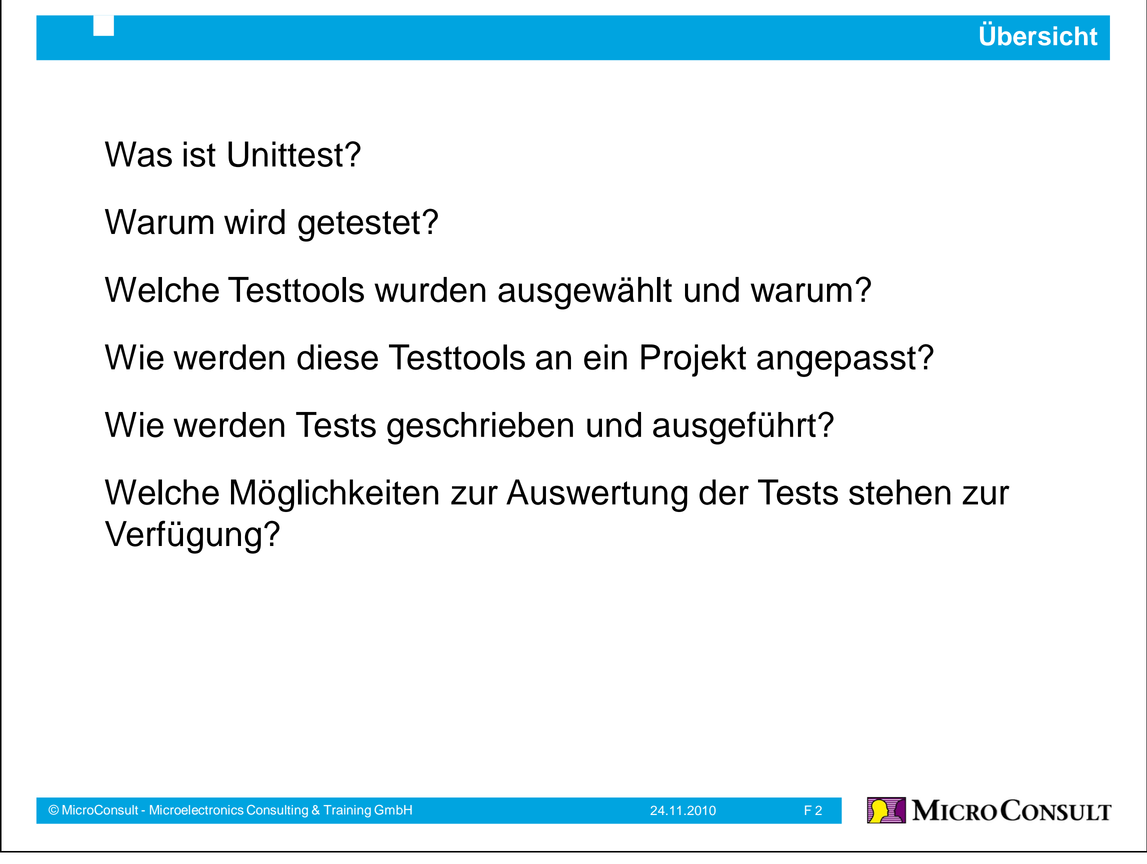
Inhalt

Testtools

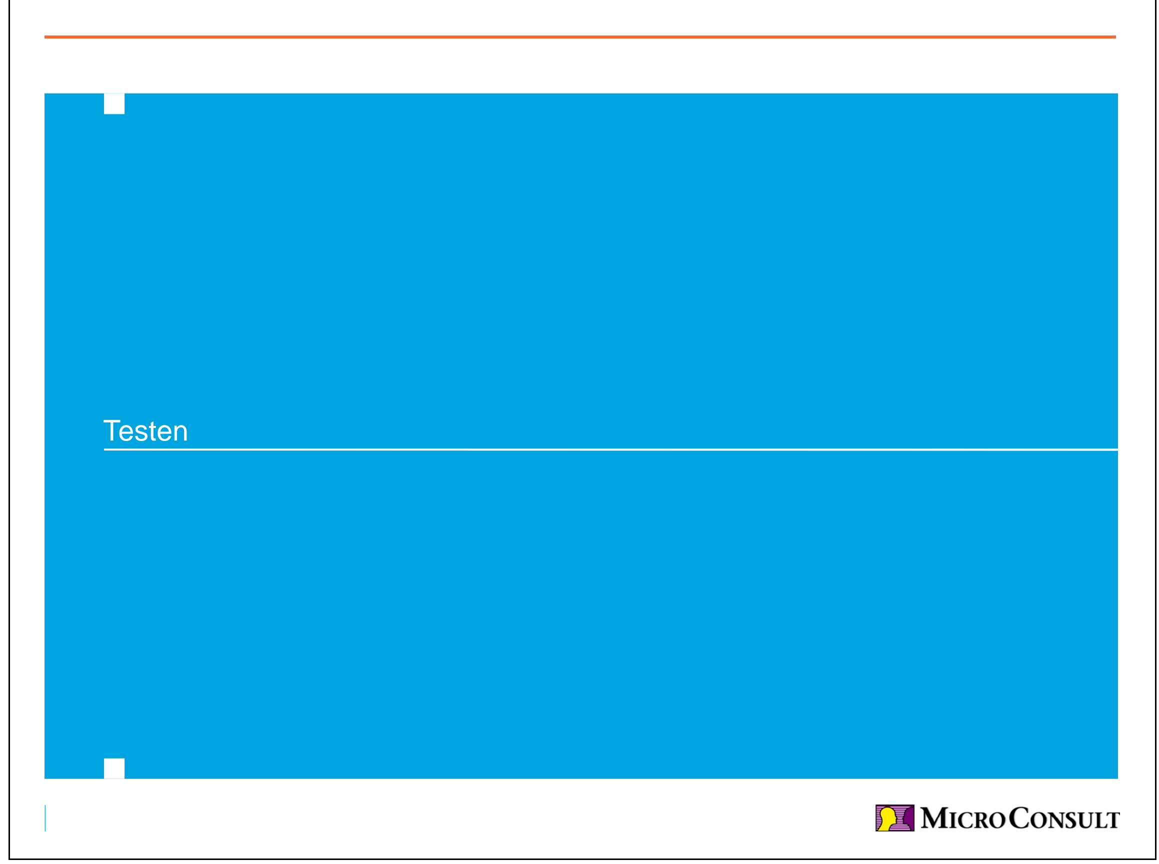
Inhalt ... 2
1 Übersicht ... 3
2 Testen ... 4
2.1 Unittest ... 5
2.2 Regressionstest ... 8
3 Freie Testtools ... 10
3.1 Auswahlkriterien ... 11
3.2 Tools ... 12
3.3 Installation der Tools ... 13
3.4 Anpassung der Tools ... 18
3.4.1 Makefiles ... 19
3.5 Erstellen von Tests ... 21
3.5.1 Beispielprojekt ... 22
3.5.2 AceUnit ... 26
3.5.3 CUnit ... 34
3.5.4 Embedded Unit ... 42
3.5.5 Unity ... 49
3.6 Testauswertung ... 56
3.6.1 AceUnit ... 57
3.6.2 CUnit ... 59
3.6.3 Embedded Unit ... 63
3.6.4 Unity ... 67
4 Zusammenfassung ... 68
5 Ausblick ... 73

© MicroConsult - MicroElectronics Consulting & Training GmbH

1 Übersicht



2 Testen



© MicroConsult - MicroElectronics Consulting & Training GmbH

2.1 Unittest

Unittest

Wikipedia:

In der Softwareentwicklung wird ein Computerprogramm üblicherweise in einzelne Teile mit klar definierten Schnittstellen, sogenannte Module, unterteilt.

Der **Modultest** (auch **Komponententest** oder oft vom engl. **unit test** als **Unittest** bezeichnet) ist der Softwaretest dieser Programmteile, die zu einem späteren Zeitpunkt zusammengefügt (integriert) werden (vgl. Integrationstest).

Ziel des Modultests ist es, frühzeitig Programmfehler in den Modulen einer Software (z. B. von einzelnen Klassen) zu finden. Die Funktionalität der Module kann so meist einfacher getestet werden, als wenn die Module bereits integriert sind, da in diesem Fall die Abhängigkeit der Einzelmodule mit in Betracht gezogen werden muss.

Quelle: http://de.wikipedia.org/wiki/Modultest

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 4

Unittest

Unittest

Der Unittest prüft jede C-Funktion des Codes einzeln. Der Test prüft, ob die Funktion macht was sie soll. Daher findet er Fehler, bevor sie größere Auswirkungen haben.

Der Unittest dokumentiert weiterhin das entstehende Programm. Er zeigt eindeutig, was das einzelnen Funktionen machen.

Unit-Testprogramme sind einfach aufgebaut:

- Die Methoden des zu testenden Programms aufrufen
- Return-Werte aufnehmen
- mit Soll-Werten vergleichen
- Ergebnisse aufsammeln und weiterverarbeiten oder anzeigen

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 5

© MicroConsult - MicroElectronics Consulting & Training GmbH

Unittest

Eine besondere Bedeutung hat der Unittest in der Testgetriebenen Entwicklung (Test driven development – TDD). Es wird immer ein Unittest vor dem Erstellen bzw. Ändern des eigentlichen Programmcodes erstellt.

Vorteile:

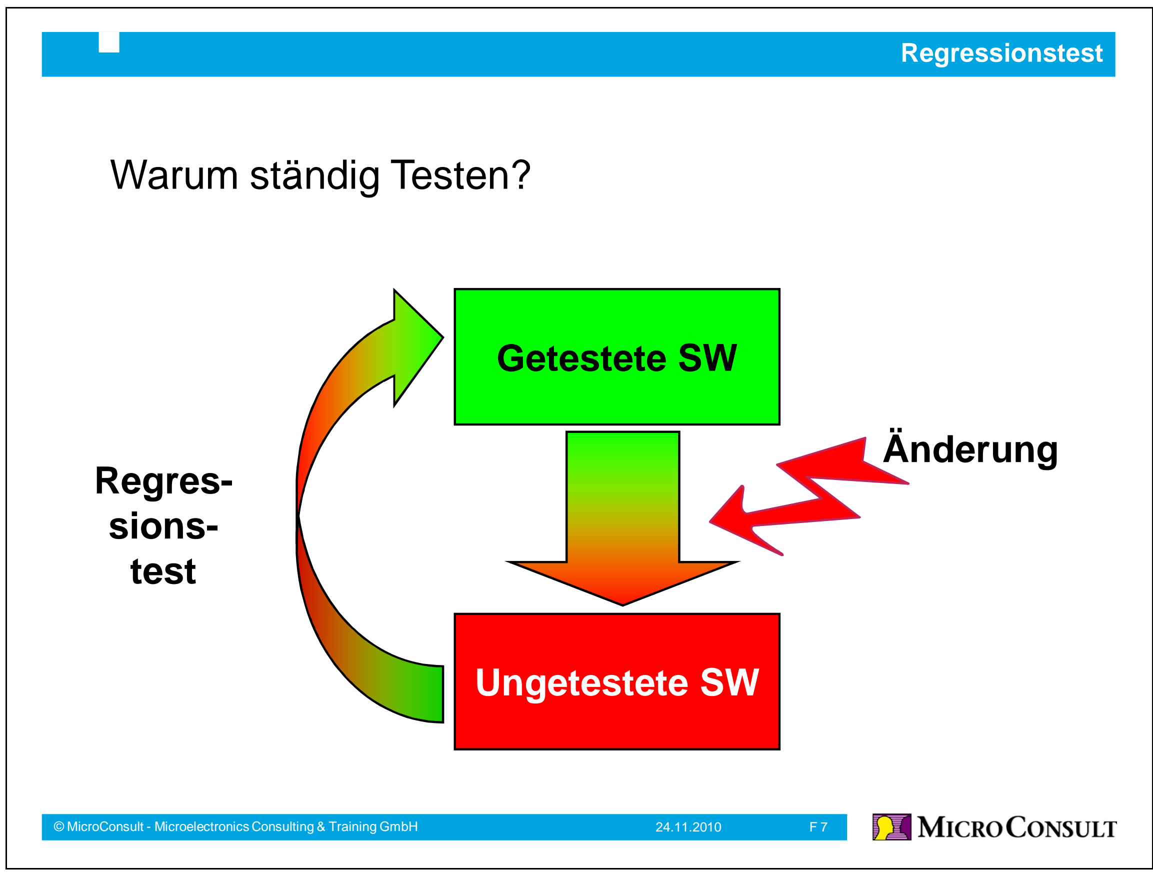
- Das Design wird verbessert, da sich der Entwickler über das benötigte Verhalten der Funktion klar werden muss, bevor er mit der Programmierung beginnt.
- Es wird nur implementiert, was nötig ist – der Test gibt die Funktionalität vor.

In der Praxis ist es oft leichter einen Test vor der Implementierung zu schreiben als umgekehrt.

Grundregel: Pro Fehler ein Test.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 6

2.2 Regressionstest



Ziel: Nachweis, dass alle schon getesteten Funktionen nach einer Codeänderung noch korrekt ablaufen.

Nach einer Codeänderung darf keine Regression auftreten,
d.h., die geänderten Anteile müssen weiterhin der Spezifikation genügen.
Dies wird durch ***Regressionstests*** geprüft.

© MicroConsult - MicroElectronics Consulting & Training GmbH

Regressionstest

Es ist sinnvoll nach jeder Änderung im Code einen Regressionstest ablaufen zu lassen.

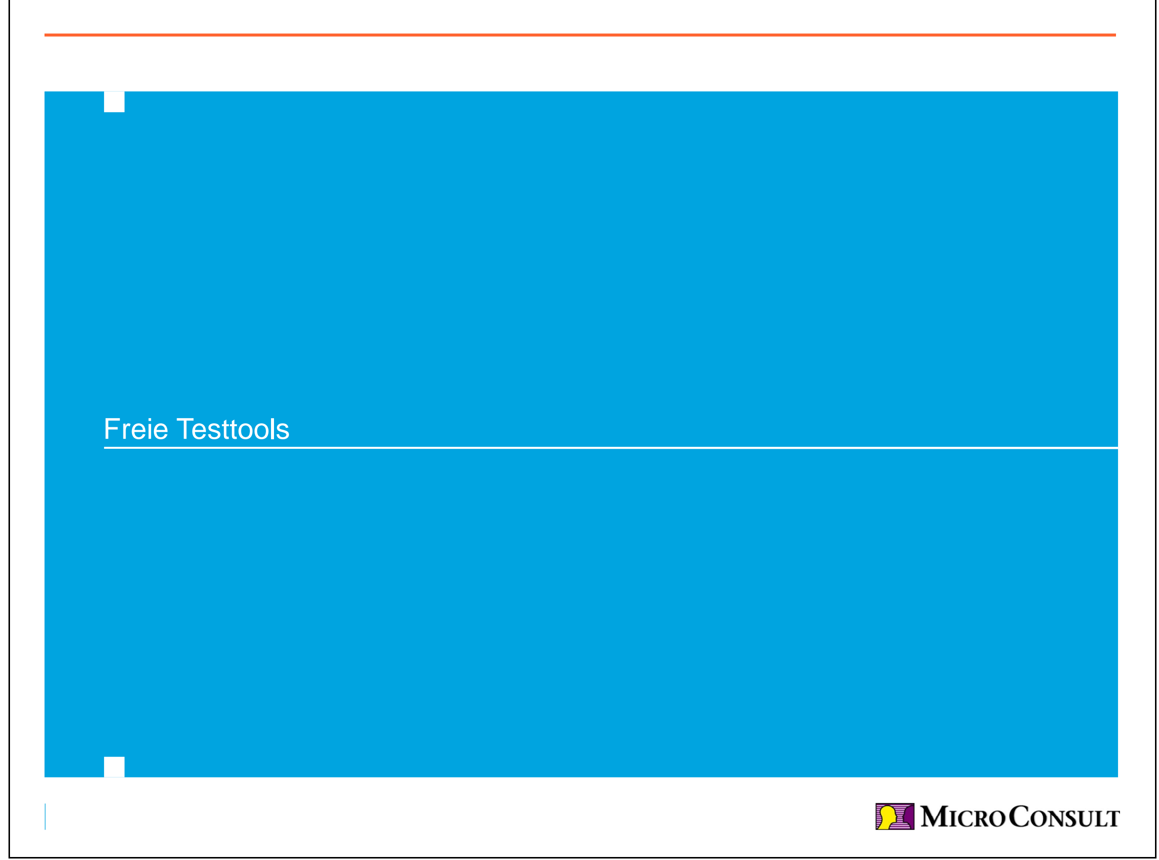
Damit wird sichergestellt, dass mit neuem Code keine oder wenig neue Fehler dazugekommen sind und der vorhandene Code immer noch funktioniert.

Wird erst nach vielen Änderungen getestet, ist es durch den großen Codehub schwieriger, die eventuell entstandenen Fehler zu lokalisieren.

Regressionstests
können nur in einer **automatisierten** Testumgebung
effektiv durchgeführt werden.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 8 MicroConsult

3 Freie Testtools



© MicroConsult - MicroElectronics Consulting & Training GmbH

3.1 Auswahlkriterien

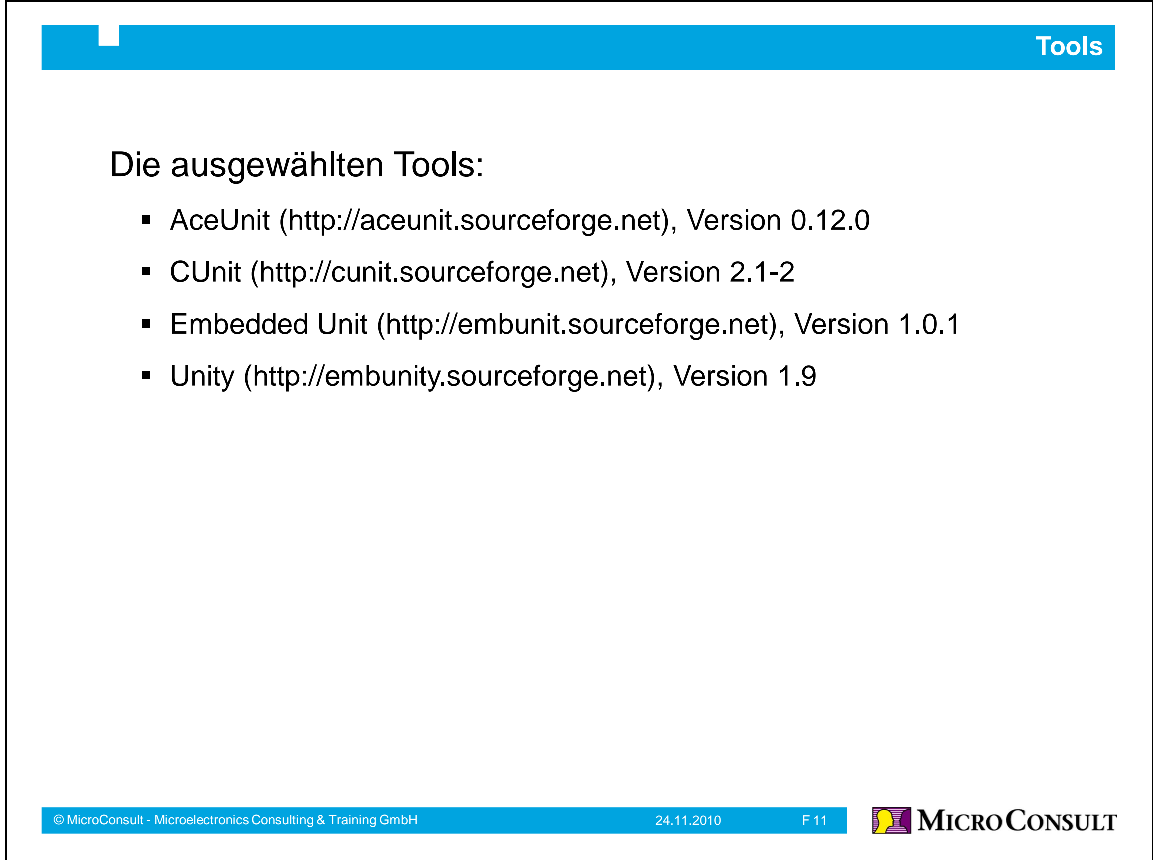
Auswahlkriterien

Folgende Auswahlkriterien wurden für die Auswahl der Tools verwendet (Google-Suche, Auswertung der Informationen auf der Webseite):

- Es testet C-Programme.
- Es kann zum Test von embedded Software verwendet werden.
- Wenig Speicherverbrauch (Code und Daten).
- Es hat eine gewisse Reife erreicht.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 10 MICROCONSULT

3.2 Tools



AceUnit wurde trotz der kleinen Version ausgewählt, da die Beschreibung des Leistungsumfangs eine ganze Menge erwarten ließ.

© MicroConsult - MicroElectronics Consulting & Training GmbH

3.3 Installation der Tools

Installation

Installation

Alle Tools liegen im Quellcode vor. Teilweise werden Hilfsprogramme oder -skripts mitgeliefert.

Die Übersetzung des Quellcodes erfolgt meistens mit Hilfe von Makefiles. Als Referenz wurde *nmake* von Microsoft gewählt. Da *nmake* seit Jahren nicht weiterentwickelt wurde, ist davon auszugehen, dass alles, was mit *nmake* übersetzt werden kann auch mit anderen make-Systemen läuft.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 12 MicroConsult

AceUnit

Download:

- Zip-Datei mit dem Quellcode
- Java-Programm zur Generierung von Testläufen
- Powerpoint-Präsentation über AceUnit

Dokumentation

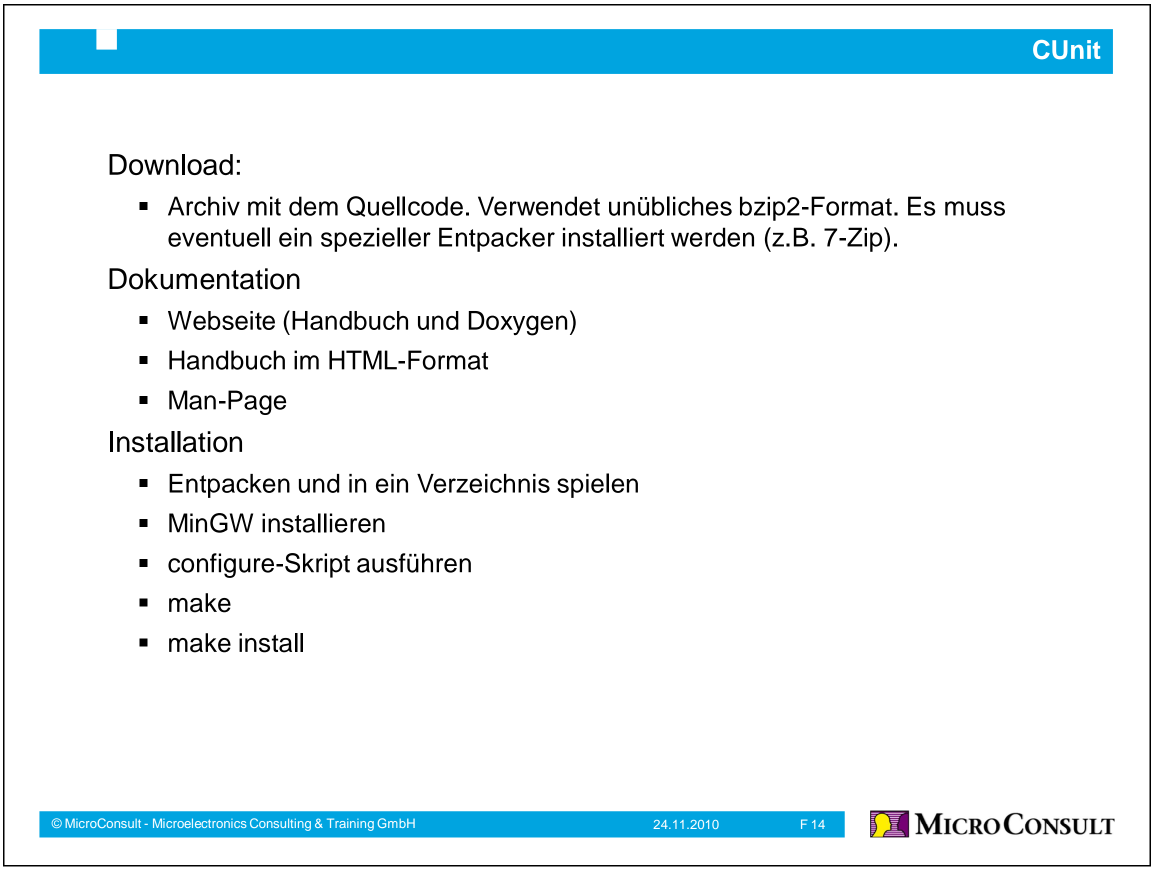
- Webseite
- Powerpoint-Präsentation
- Doxygen-Dokumentation des Quellcodes

Installation

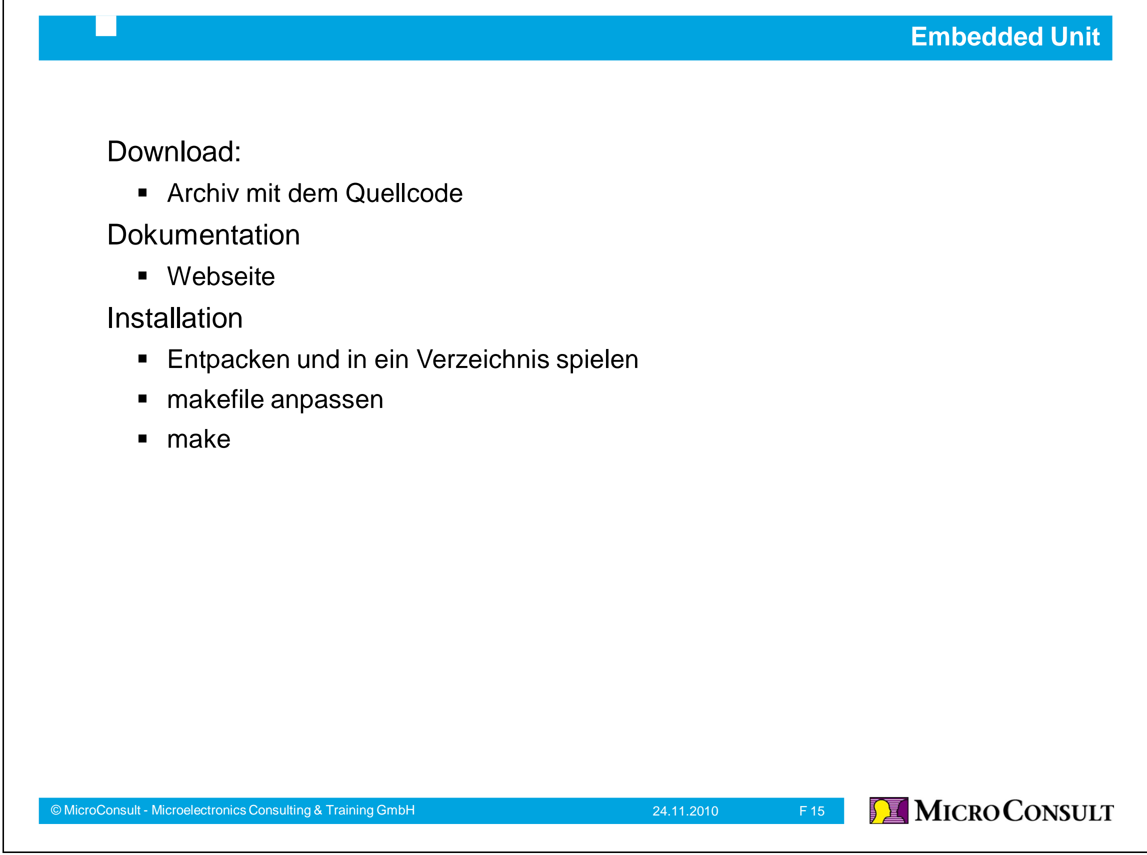
- Entpacken und in ein Verzeichnis spielen
- Für das Java-Programm ist eine Java-Runtime > 1.5 erforderlich
- Da der Quellcode direkt in das Testprogramm eingebunden wird, ist keine separate Übersetzung nötig.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 13 MICROCONSULT

© MicroConsult - MicroElectronics Consulting & Training GmbH



Die Nutzung eines configure-Skripts ist sehr komfortabel, da kein makefile von Hand angepasst werden muss. Allerdings setzt das Skript eine UNIX-Umgebung voraus, d.h. unter Windows ist es ohne MinGW oder Cygwin nicht ablauffähig.



© MicroConsult - MicroElectronics Consulting & Training GmbH

Unity

Download:

- Archiv mit dem Quellcode.

Dokumentation

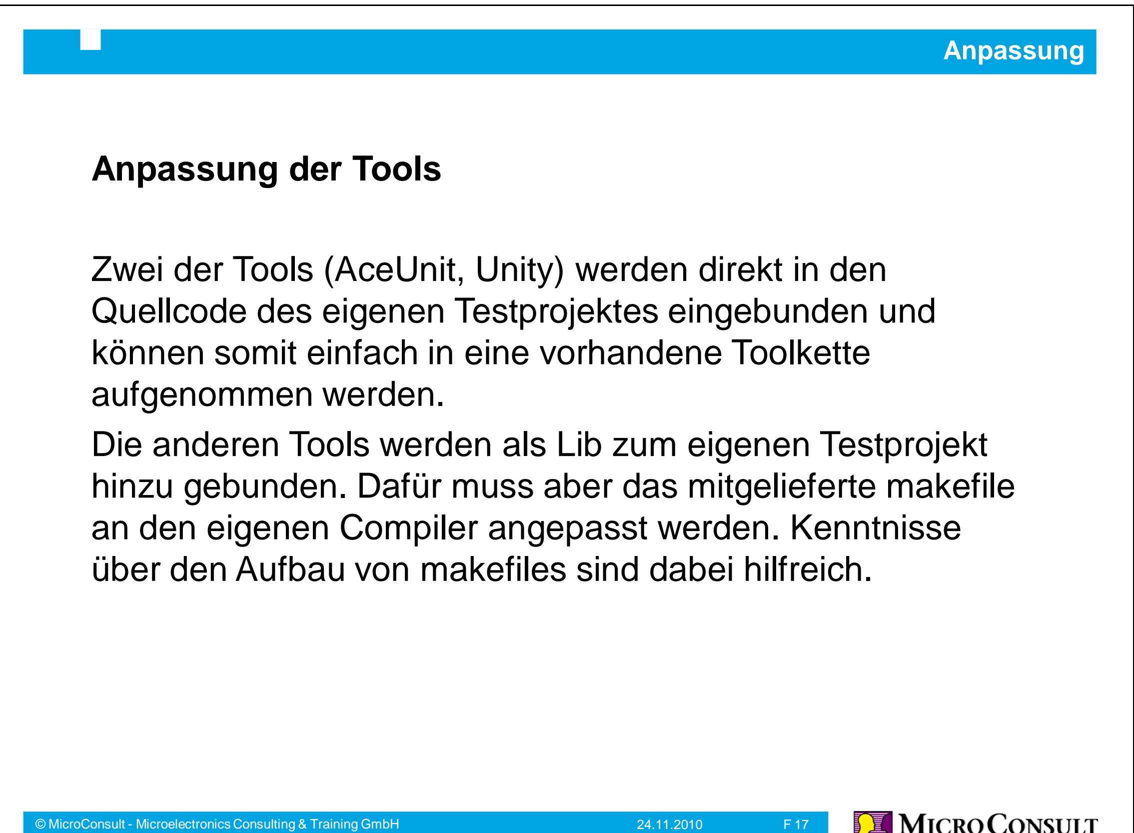
- Webseite
- Handbuch im PDF-Format

Installation

- Entpacken und in ein Verzeichnis spielen
- Da der Quellcode direkt in das Testprogramm eingebunden wird, ist keine separate Übersetzung nötig.
- Optional → mehr Komfort:
  - Ruby installieren
  - eigene *.yml-Datei erstellen oder vorhandene nutzen
  - rakefile.rb anpassen
  - eventuell rakefile_helper.rb anpassen

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 16

3.4 Anpassung der Tools



© MicroConsult - MicroElectronics Consulting & Training GmbH

3.4.1 Makefiles

Makefiles

Die wichtigsten Inhalte von Makefiles sind Variablen, Targets, Regeln und Abhängigkeiten.

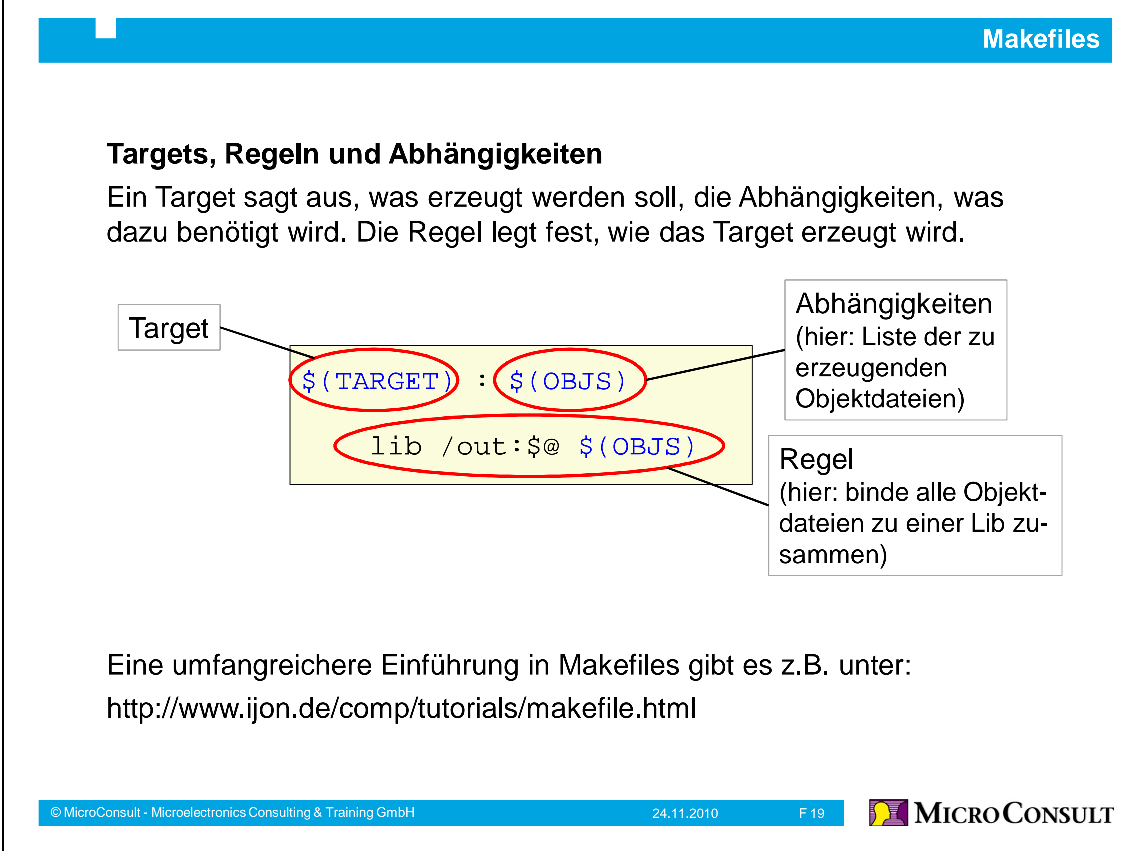
Variablen

Mit Hilfe der Variablen werden an exponierter Stelle im Makefile Voreinstellungen vorgenommen. Damit muss bei einer Anpassung nicht immer das gesamte Makefile geändert werden, meistens werden nur die Werte der Variablen angepasst.

```
RMDIR = rmdir
RMDIROPT = /Q /S
INCLUDES = ..
OUTDIR = out
```

Diese Variablen werden dann im Makefile verwendet. Z.B.:

```
clean:
-$(RMDIR) $(RMDIROPT) "$(OUTDIR)"
```



© MicroConsult - MicroElectronics Consulting & Training GmbH

3.5 Erstellen von Tests

Erstellen von Tests

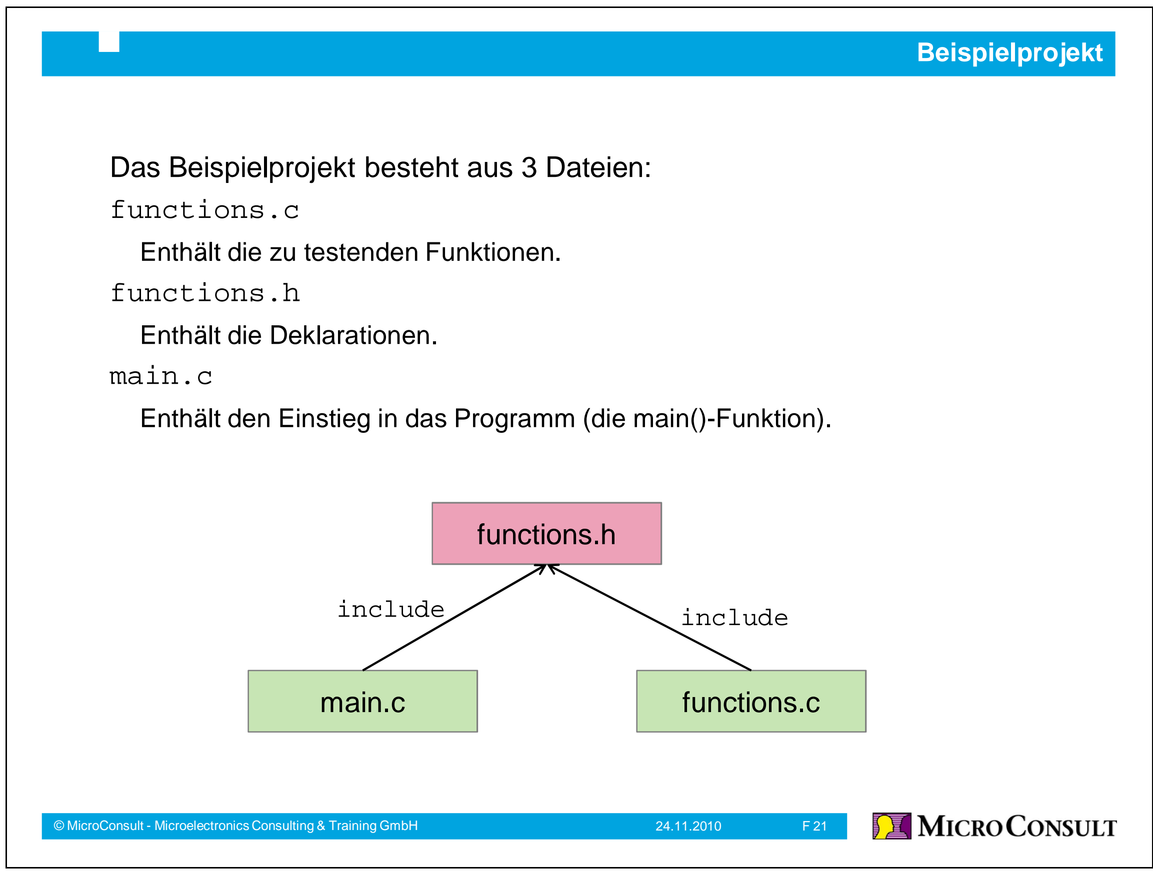
Erstellen von Tests

Die Vorgehensweise bei der Erstellung von Tests ist für alle Tools ähnlich:

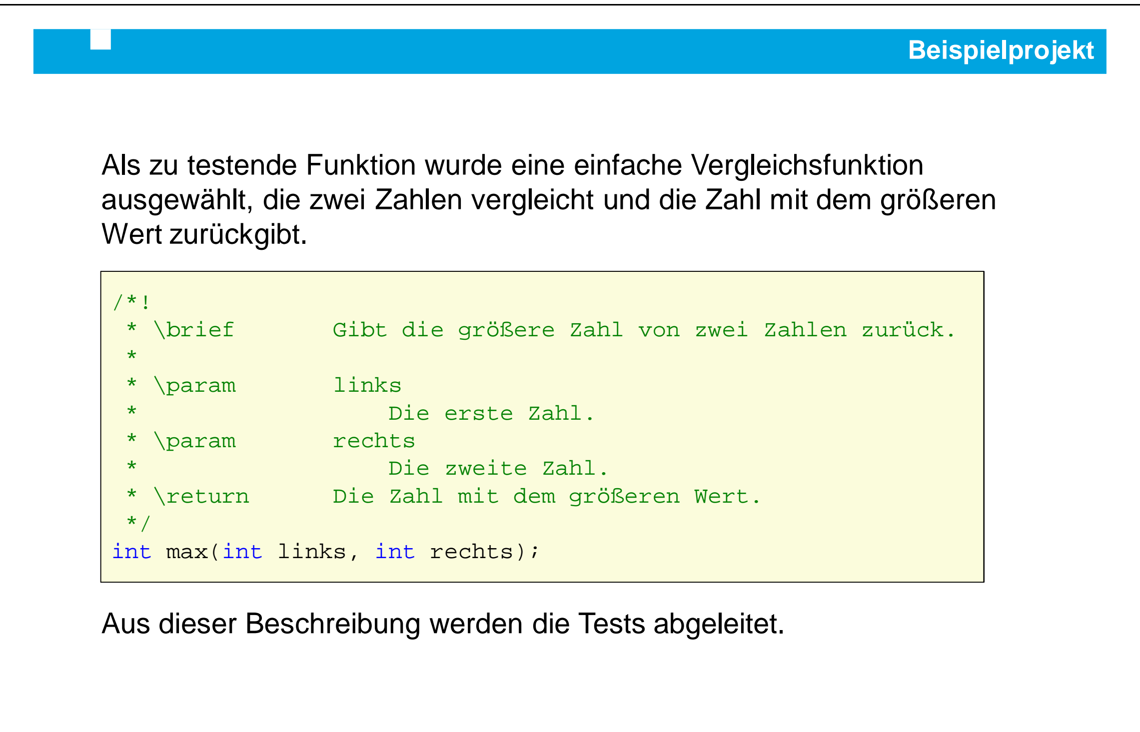
1. Funktionen für die Initialisierung des Tests und für das Aufräumen nach dem Test schreiben.
2. Testfunktionen schreiben.
   1. Eingangswerte bereitstellen.
   2. Zu testende Funktion aufrufen.
   3. Ausgabewerte überprüfen. Dazu werden vom Testtool diverse Assertions bereitgestellt.
3. Eventuell Testsuiten erstellen (toolabhängig).
4. Main-Funktion für das Testprogramm schreiben (wird teilweise bereitgestellt bzw. generiert).
5. Testprogramm kompilieren
6. Test ausführen

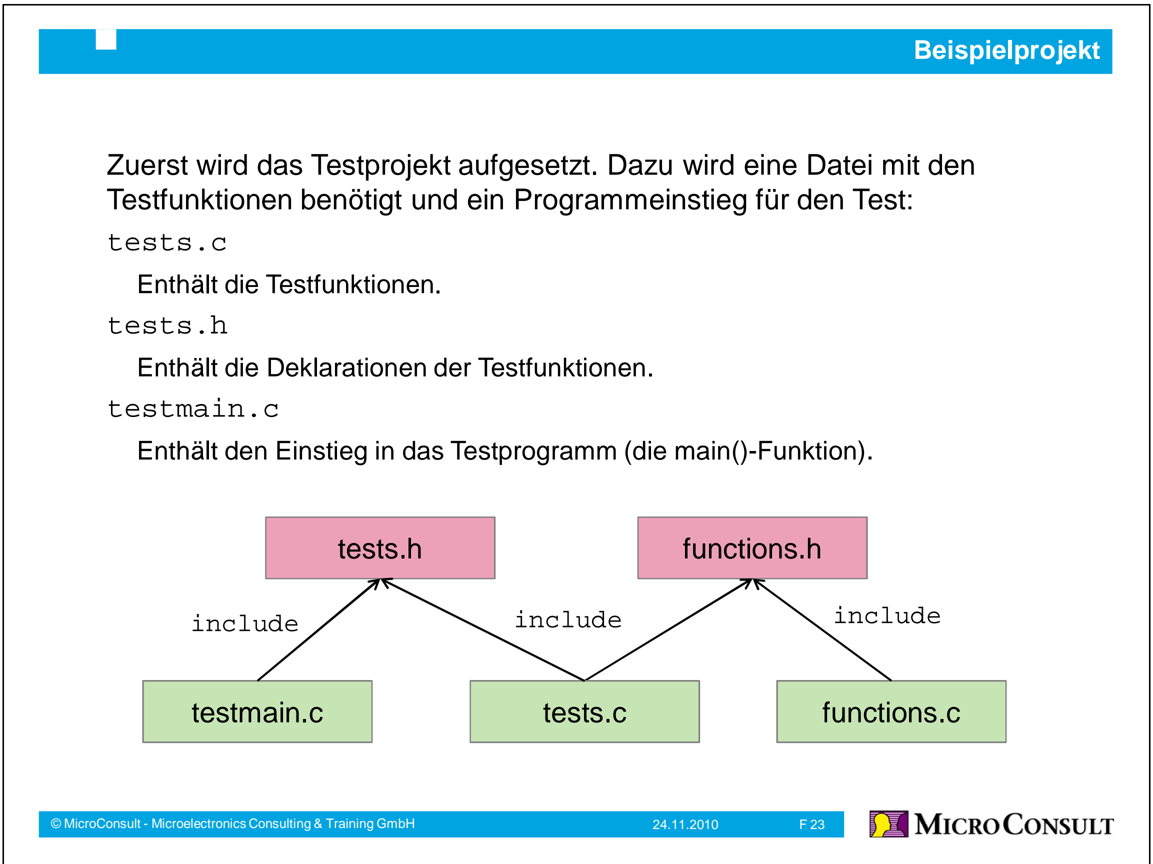
© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 20 MicroConsult

3.5.1 Beispielprojekt



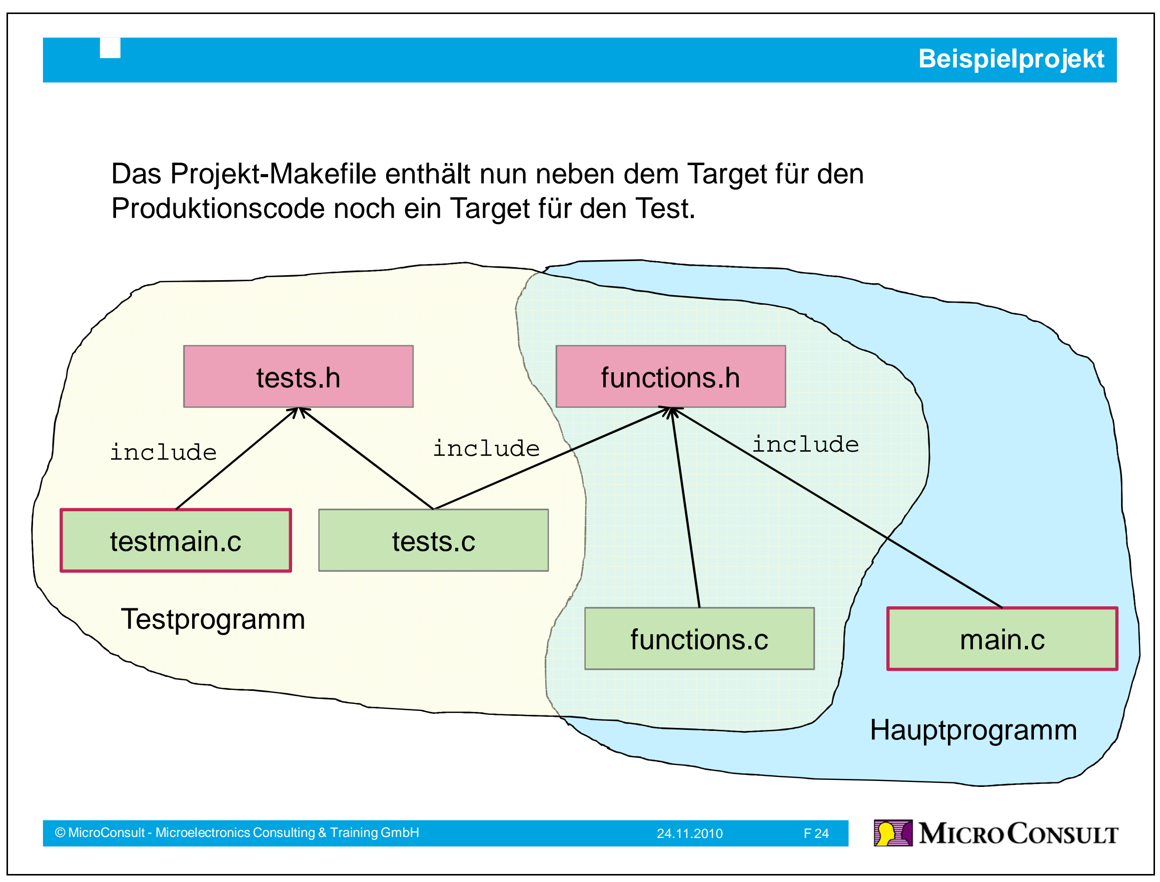
© MicroConsult - MicroElectronics Consulting & Training GmbH


Beispielprojekt
Als zu testende Funktion wurde eine einfache Vergleichsfunktion ausgewählt, die zwei Zahlen vergleicht und die Zahl mit dem größeren Wert zurückgibt.
/*!
 * \brief Gibt die größere Zahl von zwei Zahlen zurück.
 *
 * \param links
 * Die erste Zahl.
 * \param rechts
 * Die zweite Zahl.
 * \return Die Zahl mit dem größeren Wert.
 */
int max(int links, int rechts);
Aus dieser Beschreibung werden die Tests abgeleitet.
© MicroConsult - Microelectronics Consulting & Training GmbH
24.11.2010
F 22
MicroConsult



Je nach verwendetem Tool werden die Dateien mit dem Test-Hauptprogramm und den Test-Deklarationen automatisch generiert.

© MicroConsult - MicroElectronics Consulting & Training GmbH


Beispielprojekt
Das Projekt-Makefile enthält nun neben dem Target für den Produktionscode noch ein Target für den Test.
tests.h
functions.h
include
include
include
testmain.c
tests.c
functions.c
main.c
Testprogramm
Hauptprogramm
© MicroConsult - Microelectronics Consulting & Training GmbH
24.11.2010
F 24
MicroConsult

3.5.2 AceUnit

3.5.2.1 Funktionen für Initialisierung und Aufräumen

AceUnit – Funktionen für Initialisierung und Aufräumen

Die Funktionen für Initialisierung und Aufräumen müssen bei AceUnit keine speziellen Namen haben. Es ist auch nicht notwendig, sie beim Start des Tests anzugeben.

AceUnit stellt Makros bereit, mit denen diese Funktionen gekennzeichnet werden. Der Generator erzeugt anhand dieser Makros den nötigen Code.

`A_Before`

Kennzeichnet eine Funktion, die vor einer Testfunktion ausgeführt wird.

`A_After`

Kennzeichnet eine Funktion, die nach einer Testfunktion ausgeführt wird.

`A_BeforeClass`

Kennzeichnet eine Funktion, die vor einem Testlauf ausgeführt wird.

`A_AfterClass`

Kennzeichnet eine Funktion, die nach einem Testlauf ausgeführt wird.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 25 MicroConsult

© MicroConsult - MicroElectronics Consulting & Training GmbH

AceUnit – Funktionen für Initialisierung und Aufräumen

```
// Dieser Header ist für die Compilierung zwingend notwendig
#include "aceunitdata.h"

A_BeforeClass void initTestrun(void)
{
    ...
}

A_AfterClass void cleanupTestrun(void)
{
    ...
}

A_Before void initTestcase(void)
{
    ...
}

A_After void cleanupTestcase(void)
{
    ...
}
```


Die Makros A_* werden nur vom Generator benutzt. Auf die Kompilierung haben sie keinen Einfluss.

3.5.2.2 Testfunktionen bereitstellen

AceUnit – Testfunktionen bereitstellen

Auch für die Testfunktionen sind bei AceUnit keine speziellen Namen notwendig.

Es werden auch hier Makros zur Kennzeichnung bereitgestellt. Der Generator erzeugt anhand dieser Makros den nötigen Code.

`A_Test`

Kennzeichnet eine Testfunktion.

```
// Dieser Header wird von AceUnit generiert
#include "tests.h"

// Dieser Header ist für die Compilierung zwingend notwendig
#include "aceunitdata.h"

#include "functions.h"
#include <stdio.h>

A_Test void test_max1(void)
{
    assertTrue("Test 1", max(2, 4) == 4);
}
```

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 27

© MicroConsult - MicroElectronics Consulting & Training GmbH

AceUnit – Testfunktionen bereitstellen

Folgende Assertions werden bereitgestellt:

- `fail(message)`
  Erzeugt einen Fehler.
- `assertTrue(message, condition)`
  Prüft, ob die Bedingung wahr ist.
- `assertFalse(message, condition)`
  Prüft, ob die Bedingung falsch ist.
- `assertEquals(message, expected, actual)`
  Prüft, ob beide Variablen denselben Wert haben.
- `assertNotEquals(message, unexpected, actual)`
  Prüft, ob beide Variablen unterschiedliche Werte haben.
- `assertNotNull(message, ptr)`
  Prüft, ob der Zeiger ungleich NULL ist.
- `assertNull(message, ptr)`
  Prüft, ob der Zeiger gleich NULL ist.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 28

3.5.2.3 Generierung der Tests

AceUnit – Generierung der Tests

Es gibt zwei Varianten Tests zu generieren:

1. Ohne Testsuite
2. Mit Testsuite

In beiden Varianten müssen drei Dateien zum Build hinzugefügt werden:

- AceUnit.c
- AceUnitData.c
- Logger (z.B. FullPlainLogger.c)

Problem:

Die Dokumentation beschreibt die nötigen Schritte nicht ausreichend. Die Informationen müssen aus den Beispielen herausgesucht werden.

Auch der Aufruf des Generators wird nicht verständlich beschrieben.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 29 MicroConsult

© MicroConsult - MicroElectronics Consulting & Training GmbH

AceUnit – Generierung der Tests

1. Testcode ohne Testsuite

Aufruf Generator:

```
java -jar AceUnit.jar <Name der Testdatei>
```

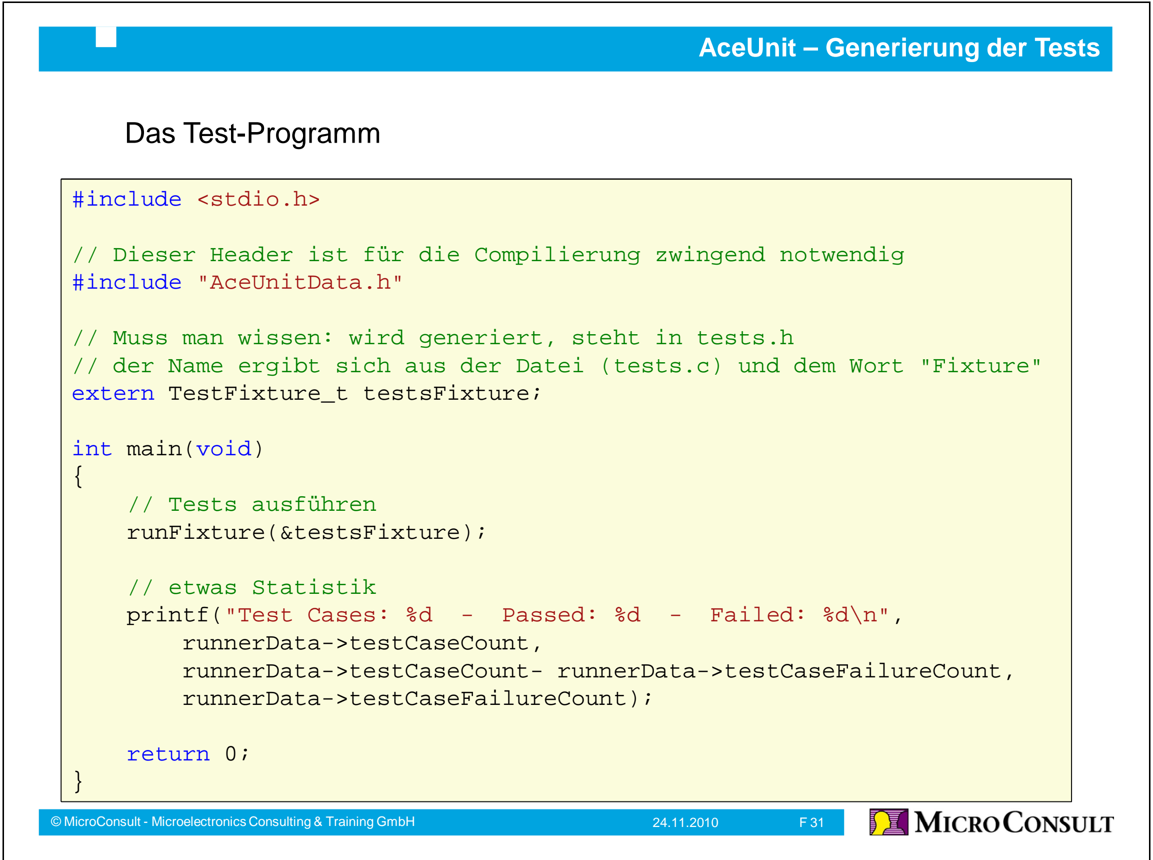
z.B.: `java -jar AceUnit-0.12.0.jar tests`

Es wird nur die zur Testdatei passende Headerdatei erzeugt.

Die Headerdatei muss in der Datei mit dem Testcode includiert werden.

Die Datei mit der Main()-Funktion muss selbst geschrieben werden. Dabei werden globale Variablen verwendet, die in der Dokumentation nicht beschrieben werden.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 30



Die Variable *runnerData* wurde in der Datei AceUnitData.h bereitgestellt.

© MicroConsult - MicroElectronics Consulting & Training GmbH

AceUnit – Generierung der Tests

2. Testcode mit Testsuite

Aufruf Generator:

```
java -jar AceUnit.jar <Name des Testverzeichnisses>
```

z.B.: `java -jar AceUnit-0.12.0.jar .`

Es wird die zur Testdatei passende Headerdatei erzeugt und eine Codedatei mit dem Code für die Testsuite.

Die Headerdatei muss in der Datei mit dem Testcode includiert werden.

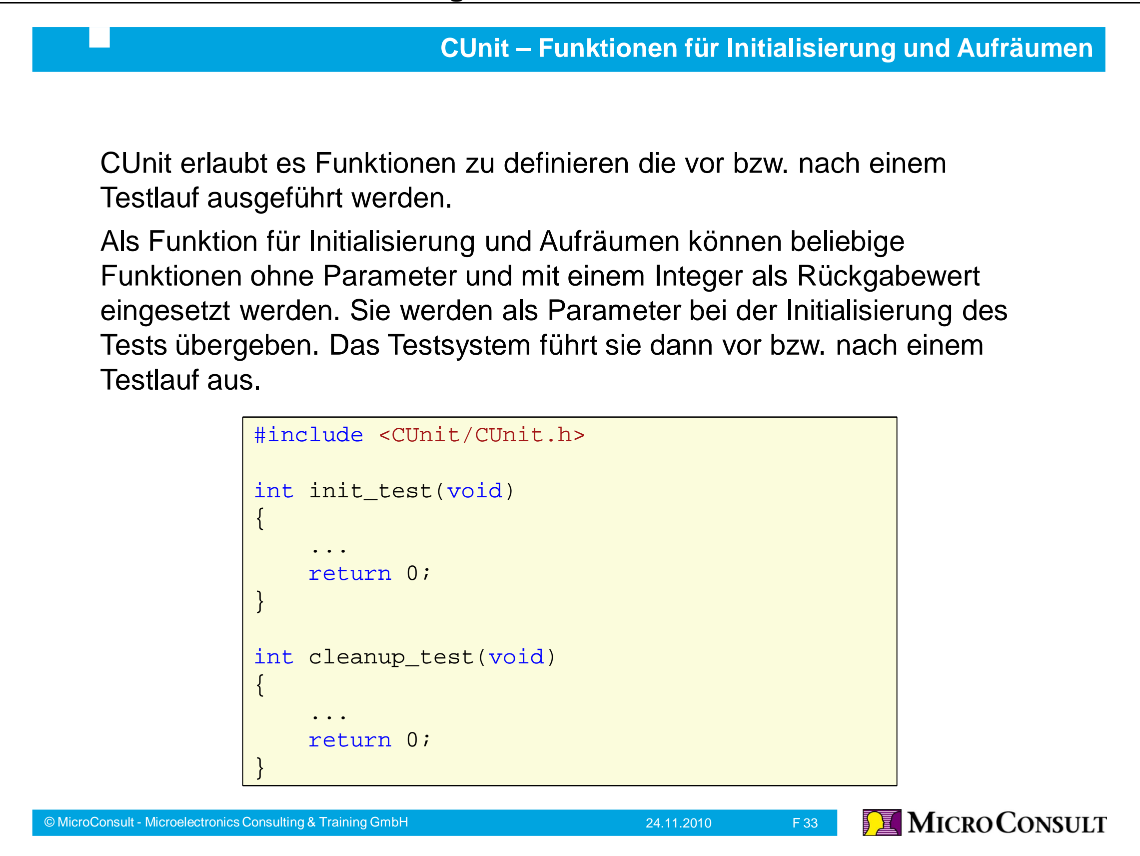
Weiterhin muss die Testsuite-Datei zum Build hinzugefügt werden.

Wird nun noch die globale Definition ACEUNIT_SUITES gesetzt und die Datei AceUnitMain.c zum Build hinzugefügt, braucht keine eigene main()-Funktion bereitgestellt werden.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 32

3.5.3 CUnit

3.5.3.1 Funktionen für Initialisierung und Aufräumen



© MicroConsult - MicroElectronics Consulting & Training GmbH

3.5.3.2 Testfunktionen bereitstellen

CUnit – Testfunktionen bereitstellen

Da die Testfunktionen den Tests manuell hinzugefügt werden müssen, sind auch hier keine speziellen Namen notwendig.

Es kann jede beliebige Funktion ohne Parameter und ohne Rückgabewert verwendet werden.

```
#include <CUnit/CUnit.h>

#include "functions.h"

void test_max1(void)
{
    CU_ASSERT(max(2, 4) == 4);
}
```

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 34 MicroConsult

CUnit – Testfunktionen bereitstellen

Folgende Assertions werden bereitgestellt:

- `CU_ASSERT(int expression)`
  `CU_ASSERT_FATAL(int expression)`
  `CU_TEST(int expression)`
  `CU_TEST_FATAL(int expression)`
  Prüft, ob die Bedingung wahr ist.
- `CU_ASSERT_TRUE(value)`
  `CU_ASSERT_TRUE_FATAL(value)`
  Prüft, ob der Wert wahr ist.
- `CU_ASSERT_FALSE(value)`
  `CU_ASSERT_FALSE_FATAL(value)`
  Prüft, ob der Wert falsch ist.
- `CU_ASSERT_EQUAL(actual, expected)`
  `CU_ASSERT_EQUAL_FATAL(actual, expected)`
  Prüft, ob beide Variablen denselben Wert haben.
- `CU_ASSERT_NOT_EQUAL(actual, expected))`
  `CU_ASSERT_NOT_EQUAL_FATAL(actual, expected)`
  Prüft, ob beide Variablen unterschiedliche Werte haben.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 35 MICROCONSULT

© MicroConsult - MicroElectronics Consulting & Training GmbH

CUnit – Testfunktionen bereitstellen

- `CU_ASSERT_PTR_EQUAL(actual, expected)`
  `CU_ASSERT_PTR_EQUAL_FATAL(actual, expected)`
  Prüft, ob beide Zeiger gleiche Werte haben.
- `CU_ASSERT_PTR_NOT_EQUAL(actual, expected)`
  `CU_ASSERT_PTR_NOT_EQUAL_FATAL(actual, expected)`
  Prüft, ob beide Zeiger unterschiedliche Werte haben.
- `CU_ASSERT_PTR_NULL(value)`
  `CU_ASSERT_PTR_NULL_FATAL(value)`
  Prüft, ob der Zeiger gleich NULL ist.
- `CU_ASSERT_PTR_NOT_NULL(value)`
  `CU_ASSERT_PTR_NOT_NULL_FATAL(value)`
  Prüft, ob der Zeiger ungleich NULL ist.
- `CU_ASSERT_STRING_EQUAL(actual, expected)`
  `CU_ASSERT_STRING_EQUAL_FATAL(actual, expected)`
  Prüft, ob beide Strings gleich sind.
- `CU_ASSERT_STRING_NOT_EQUAL(actual, expected)`
  `CU_ASSERT_STRING_NOT_EQUAL_FATAL(actual, expected)`
  Prüft, ob beide Strings unterschiedlich sind.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 36

CUnit – Testfunktionen bereitstellen

- `CU_ASSERT_NSTRING_EQUAL(actual, expected, count)`
  `CU_ASSERT_NSTRING_EQUAL_FATAL(actual, expected, count)`
  Prüft, ob die ersten *count* Zeichen beider Strings gleich sind.
- `CU_ASSERT_NSTRING_NOT_EQUAL(actual, expected, count)`
  `CU_ASSERT_NSTRING_NOT_EQUAL_FATAL(actual, expected, count)`
  Prüft, ob die ersten *count* Zeichen beider Strings unterschiedlich sind.
- `CU_ASSERT_DOUBLE_EQUAL(actual, expected, granularity)`
  `CU_ASSERT_DOUBLE_EQUAL_FATAL(actual, expected, granularity)`
  Prüft, ob die Differenz `(actual - expected) <= granularity` ist. Hierfür muss die mathematische Bibliothek zum Projekt hinzugebunden werden.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 37

© MicroConsult - MicroElectronics Consulting & Training GmbH

CUnit – Testfunktionen bereitstellen

- `CU_ASSERT_DOUBLE_NOT_EQUAL(actual, expected, granularity)`
  `CU_ASSERT_DOUBLE_NOT_EQUAL_FATAL(actual, expected, granularity)`
  Prüft, ob die Differenz *(actual - expected) > granularity* ist. Hierfür muss die mathematische Bibliothek zum Projekt hinzugebunden werden.
- `CU_PASS(message)`
  Erzeugt eine Passed-Meldung.
- `CU_FAIL(message)`
  `CU_FAIL_FATAL(message)`
  Erzeugt einen Fehler.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 38

3.5.3.3 Bereitstellung der Tests

CUnit – Bereitstellung der Tests

CUnit stellt kein Tool zur Verfügung, mit dem die Tests zusammengefasst und in einen Testablauf eingefügt werden.

Eine main()-Funktion für den Test wird bei CUnit manuell erzeugt.

Der Ablauf dafür ist folgender:

1. Test-Registry initialisieren.
2. Testsuite anlegen.
3. Testfunktionen zur Testsuite hinzufügen.
4. Test starten.
5. Test-Registry aufräumen.

Zur fehlerfreien Übersetzung des Testprogramms muss die Bibliothek *cunit* zum Projekt hinzu gebunden werden.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 39 MicroConsult

© MicroConsult - MicroElectronics Consulting & Training GmbH

CUnit – Bereitstellung der Tests

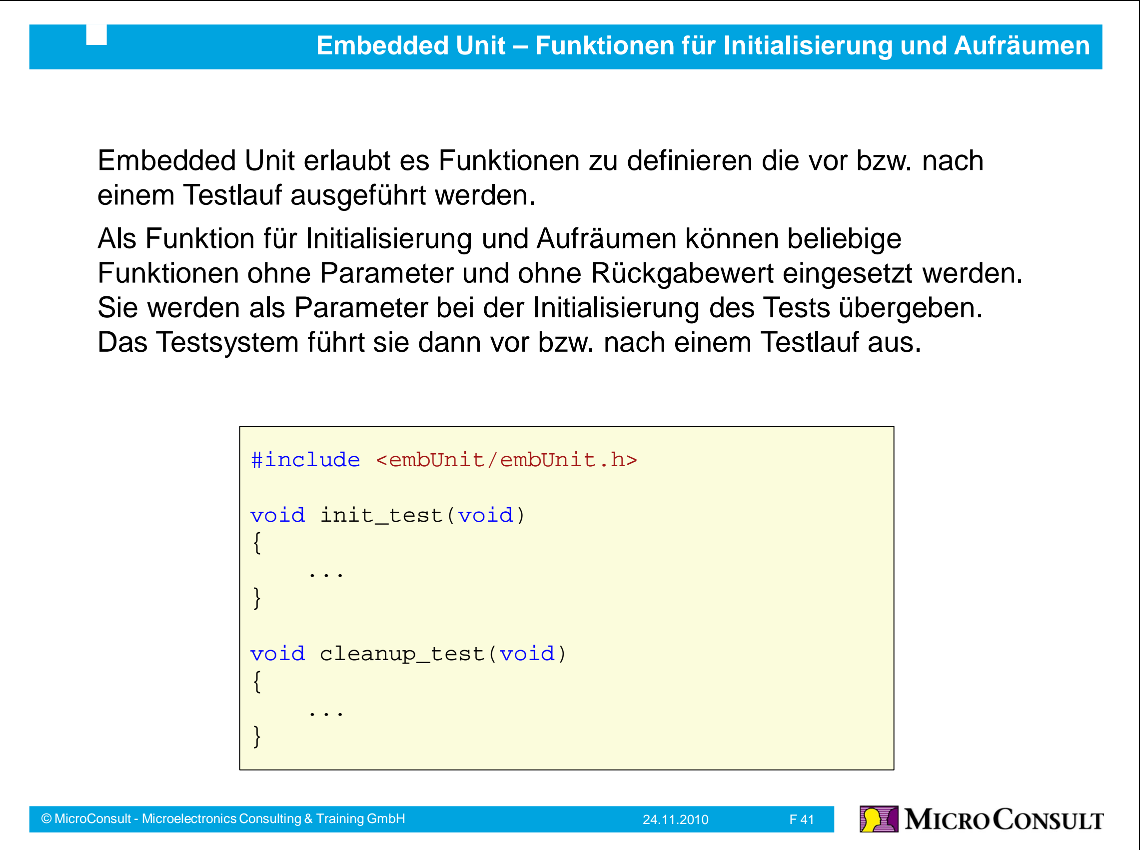
Das Test-Programm

```
#include <CUnit/CUnit.h>
#include <CUnit/Basic.h>
#include "tests.h"

int main(void)
{
    CU_pSuite suite;

    CU_initialize_registry(); // Initialisierung des Testsystems

    // Testsuite erzeugen
    suite= CU_add_suite("cfgReadline", init_test, cleanup_test);

    // Tests hinzufügen
    CU_ADD_TEST(suite, test_max1);
    CU_ADD_TEST(suite, test_max2);

    // Tests ausführen
    CU_basic_run_tests();

    CU_cleanup_registry(); // Testsytem aufräumen
    return 0;
}
```

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 40 MicroConsult

3.5.4 Embedded Unit

3.5.4.1 Funktionen für Initialisierung und Aufräumen



© MicroConsult - MicroElectronics Consulting & Training GmbH

3.5.4.2 Testfunktionen bereitstellen

Embedded Unit – Testfunktionen bereitstellen

Da die Testfunktionen in eine Liste eingetragen werden, sind auch hier keine speziellen Namen notwendig.

Es kann jede beliebige Funktion ohne Parameter und ohne Rückgabewert verwendet werden.

```
#include <embUnit/embUnit.h>

#include "functions.h"

void test_max1(void)
{
    TEST_ASSERT(max(2, 4) == 4);
}
```

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 42

Embedded Unit – Testfunktionen bereitstellen

Folgende Assertions werden bereitgestellt:

- `TEST_ASSERT(condition)`
  `TEST_ASSERT_MESSAGE(condition, message)`
  Prüft, ob die Bedingung wahr ist.
- `TEST_ASSERT_EQUAL_INT(expected, actual)`
  Prüft, ob beide Integer-Variablen denselben Wert haben.
- `TEST_ASSERT_EQUAL_STRING(expected, actual)`
  Prüft, ob beide Strings gleich sind.
- `TEST_ASSERT_NULL(pointer)`
  Prüft, ob der Zeiger gleich NULL ist.
- `TEST_FAIL(message)`
  Erzeugt einen Fehler.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 43 MICROCONSULT

© MicroConsult - MicroElectronics Consulting & Training GmbH

3.5.4.3 Bereitstellung der Tests

Embedded Unit – Bereitstellung der Tests

Die einzelnen Tests werden in einer Testsuite zusammengefasst.

Bei Embedded Unit ist es üblich pro Suite eine Funktion zur Verfügung zu stellen, welche die Datenstruktur mit den Suite-Informationen liefert.

```
TestRef Test_tests(void)
{
    EMB_UNIT_TESTFIXTURES(fixtures)
    {
        new_TestFixture("test_max1", test_max1),
        new_TestFixture("test_max2", test_max2),
        new_TestFixture("test_max3", test_max3),
        new_TestFixture("test_max4", test_max4),
    };

    EMB_UNIT_TESTCALLER(TestsTest, "TestsTest",
        init_test, cleanup_test, fixtures);

    return (TestRef)&TestsTest;
}
```

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 44 MicroConsult

Embedded Unit – Bereitstellung der Tests

Der Inhalt der main()-Funktion für den Test ist nicht sehr komplex.

Der Ablauf dafür ist folgender:

1. Testrunner starten.
2. Test starten.
3. Testrunner beenden.

Zur fehlerfreien Übersetzung des Testprogramms muss die Bibliothek *embUnit.lib* zum Projekt hinzu gebunden werden.

```
int main (int argc, const char* argv[])
{
    TestRunner_start();
    TestRunner_runTest(Test_tests());
    TestRunner_end();

    return 0;
}
```


© MicroConsult - MicroElectronics Consulting & Training GmbH

Embedded Unit – Bereitstellung der Tests

Embedded Unit bietet die Möglichkeit, eine umfangreichere Ausgabe (Text oder XML) zu Erzeugen. Dafür ist in der main()-Funktion noch ein Ausgabemedium anzugeben.

Zur fehlerfreien Übersetzung des Testprogramms muss neben der Bibliothek *embUnit.lib* noch die Bibliothek *libtextui.lib* zum Projekt hinzu gebunden werden.

```
int main (int argc, const char* argv[])
{
    // XML-Ausgabe aktivieren
    TextUIRunner_setOutputter(XMLOutputter_outputter());


    TextUIRunner_start();
    TextUIRunner_runTest(Test_tests());
    TextUIRunner_end();

    return 0;
}
```

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 46 MICROCONSULT

Embedded Unit – Bereitstellung der Tests

Zur Unterstützung der Testerzeugung gibt es vier kleine Tools, die bei der Erzeugung der Tests helfen.

Es wird allerdings kein Quellcode analysiert. Nur der Rahmen für die Tests kann bereitgestellt werden.

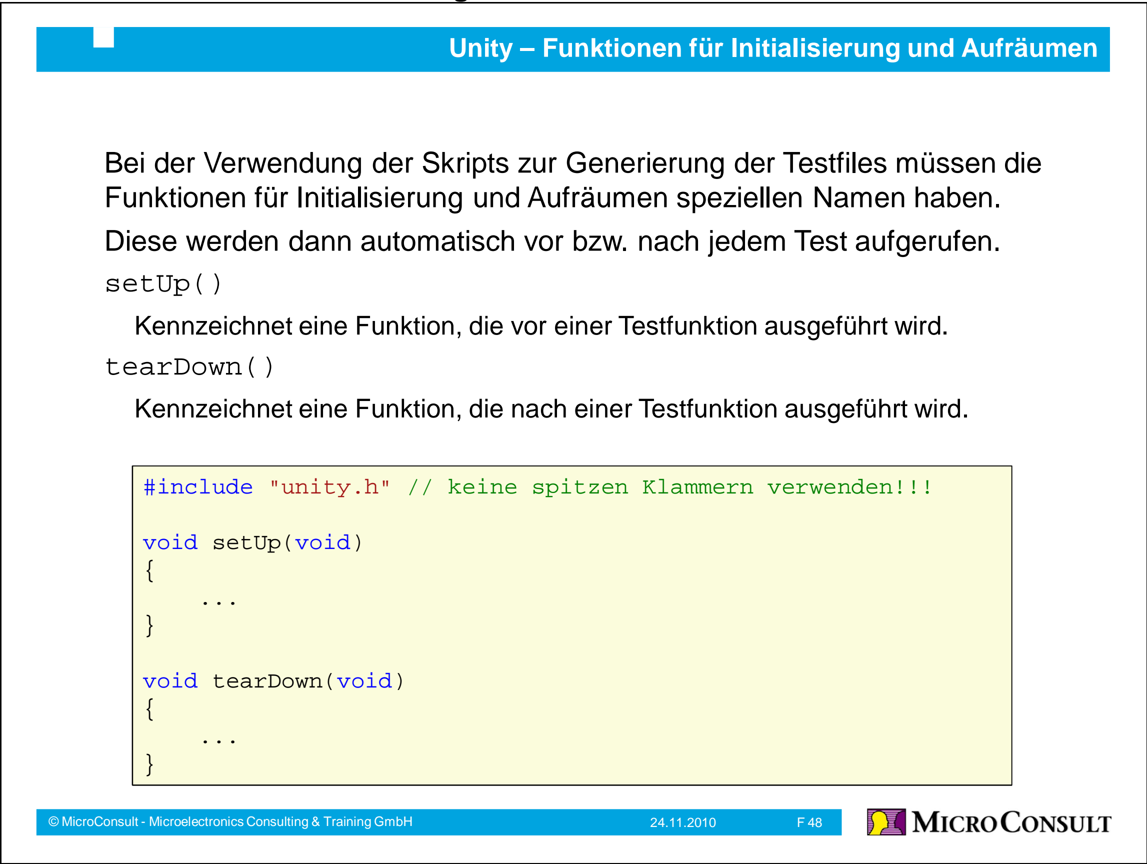
- `tcuppa.exe`
  Generiert eine Datei mit Templates für Testfunktionen und der Funktion zur Initialisierung der Testsuite.
- `tuma.exe`
  Fügt eine Testfunktion in eine durch tcuppa.exe generierte Datei ein.
- `bcuppa.exe`
  Generiert die Datei mit der main()-Funktion.
- `buma.exe`
  Fügt eine weitere Testsuite in die main()-Funktion ein.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 47

© MicroConsult - MicroElectronics Consulting & Training GmbH

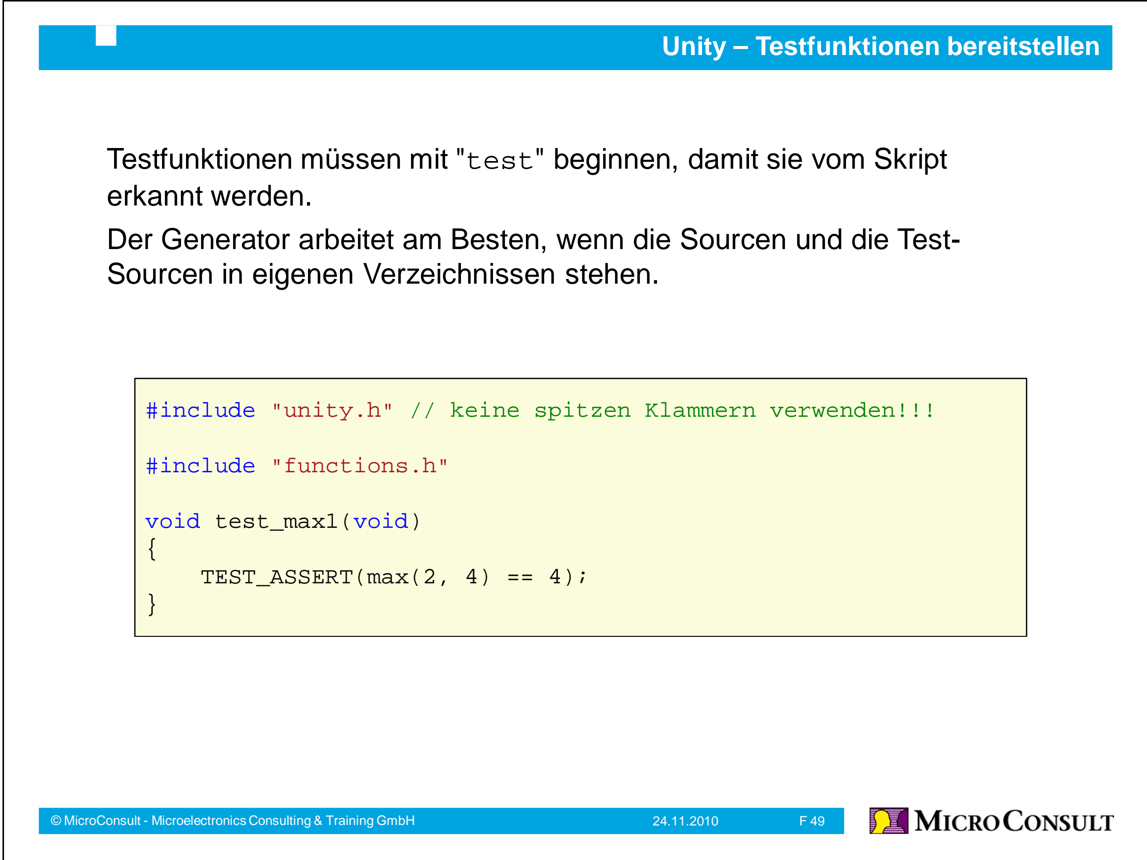
3.5.5 Unity

3.5.5.1 Funktionen für Initialisierung und Aufräumen



ACHTUNG: Wenn `unity.h` beim `#include` in spitze Klammern gesetzt wird, funktioniert das Skript zur Generierung des Testprogrammes nicht.

3.5.5.2 Testfunktionen bereitstellen



ACHTUNG: Wenn `unity.h` beim `#include` in spitze Klammern gesetzt wird, funktioniert das Skript zur Generierung des Testprogrammes nicht.

© MicroConsult - MicroElectronics Consulting & Training GmbH

Unity – Testfunktionen bereitstellen

Folgende Assertions werden bereitgestellt:

- `TEST_ASSERT(condition)`
  `TEST_ASSERT_TRUE(condition)`
  Prüft, ob die Bedingung wahr ist.
- `TEST_ASSERT_FALSE(condition)`
  `TEST_ASSERT_UNLESS(condition)`
  Prüft, ob die Bedingung falsch ist.
- `TEST_ASSERT_EQUAL(expected, actual)`
  `TEST_ASSERT_EQUAL_INT(expected, actual)`
  Prüft, ob beide Integer-Variablen denselben Wert haben.
  Zeigt Fehler als Integer an.
- `TEST_ASSERT_EQUAL_UINT(expected, actual)`
  Prüft, ob beide Integer-Variablen denselben Wert haben.
  Zeigt Fehler als unsigned Integer an.
- `TEST_ASSERT_EQUAL_HEX(expected, actual)`
  `TEST_ASSERT_EQUAL_HEX32(expected, actual)`
  Prüft, ob beide Integer-Variablen denselben Wert haben.
  Zeigt Fehler als 32-Bit Hex-Wert an.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 50

Unity – Testfunktionen bereitstellen

- `TEST_ASSERT_EQUAL_HEX16(expected, actual)`
  Prüft, ob beide Integer-Variablen denselben Wert haben.
  Zeigt Fehler als 16-Bit Hex-Wert an.
- `TEST_ASSERT_EQUAL_HEX8(expected, actual)`
  Prüft, ob beide Integer-Variablen denselben Wert haben.
  Zeigt Fehler als 8-Bit Hex-Wert an.
- `TEST_ASSERT_EQUAL_xxx_MESSAGE(expected, actual, message)`
  Entspricht den obigen Makros, gibt zusätzlich eine benutzerdefinierte Nachricht aus.
- `TEST_ASSERT_EQUAL_INT_ARRAY(mask, expected, num_elem, actual)`
  `TEST_ASSERT_EQUAL_UINT_ARRAY(mask, expected, num_elem, actual)`
  `TEST_ASSERT_EQUAL_HEX32_ARRAY(mask, expected, num_elem, actual)`
  `TEST_ASSERT_EQUAL_HEX_ARRAY(mask, expected, num_elem, actual)`
  Vergleicht `num_elem` Elemente des angegebenen Arrays.

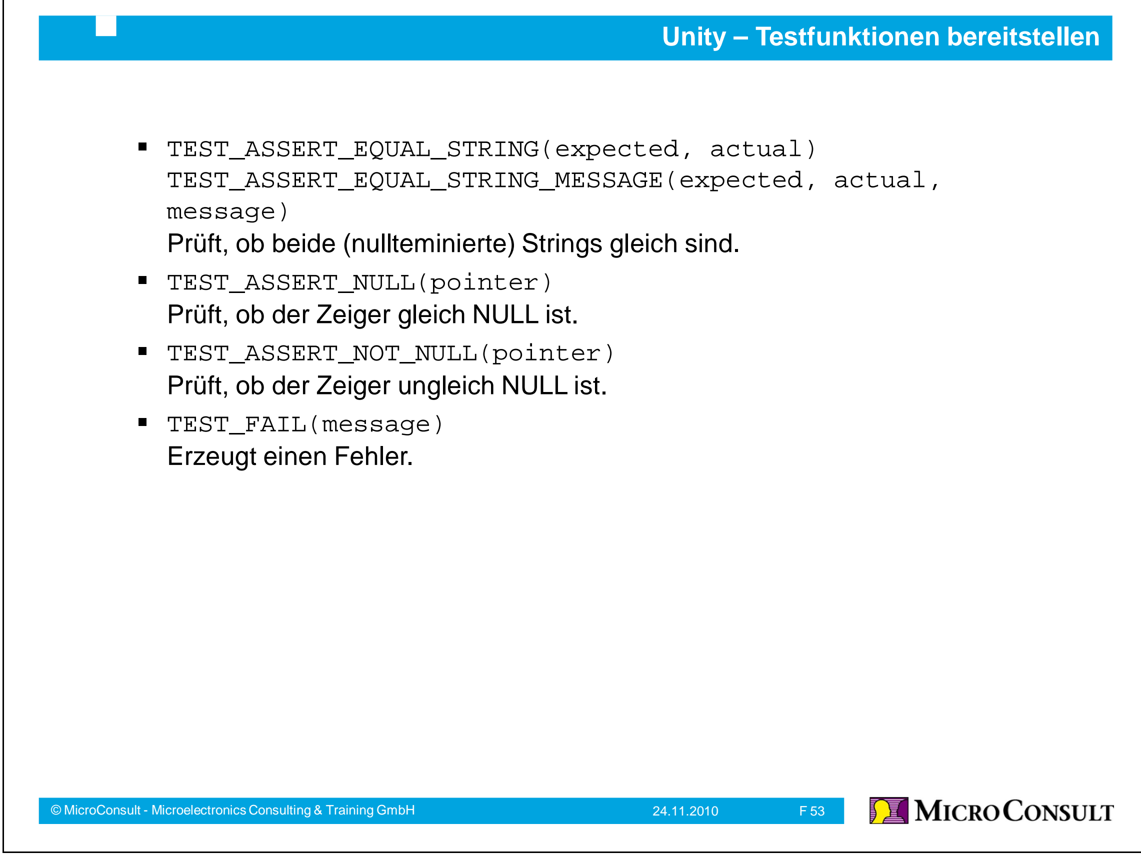
© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 51

© MicroConsult - MicroElectronics Consulting & Training GmbH

Unity – Testfunktionen bereitstellen

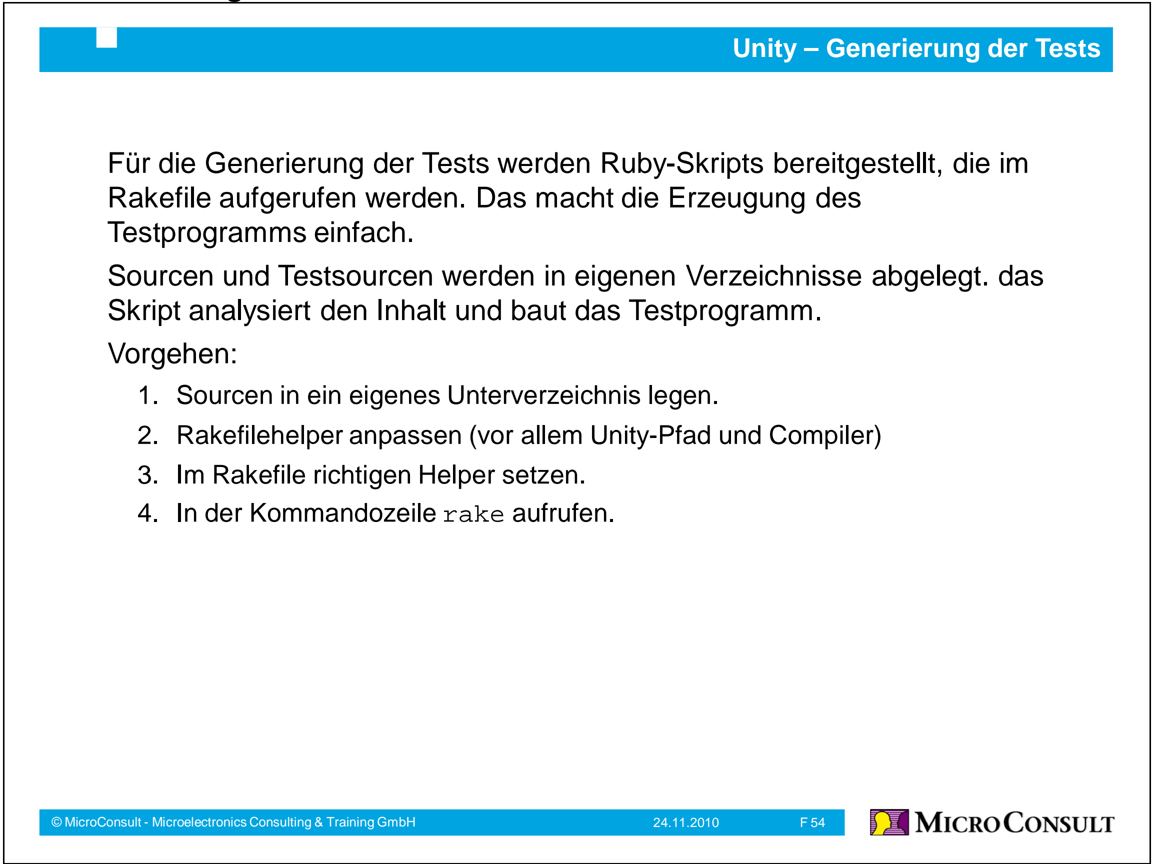
- `TEST_ASSERT_BITS(mask, expected, actual)`
  Vergleicht die in der Maske gesetzten Bits.
- `TEST_ASSERT_BITS_HIGH(mask, actual)`
  Prüft, ob die in der Maske gesetzten Bits den Wert `high` haben.
- `TEST_ASSERT_BITS_LOW(mask, actual)`
  Prüft, ob die in der Maske gesetzten Bits den Wert `low` haben.
- `TEST_ASSERT_BIT_HIGH(bit, actual)`
  Prüft, ob das angegebene Bit den Wert `high` hat.
- `TEST_ASSERT_BIT_LOW(bit, actual)`
  Prüft, ob das angegebene Bit den Wert `low` hat.
- `TEST_ASSERT_INT_WITHIN(delta, expected, actual)`
  Prüft, ob der aktuelle Integer-Wert im Bereich +-`delta` vom erwarteten Wert liegt.
- `TEST_ASSERT_FLOAT_WITHIN(delta, expected, actual)`
  Prüft, ob der aktuelle Float-Wert im Bereich +-`delta` vom erwarteten Wert liegt.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 52 MICROCONSULT



© MicroConsult - MicroElectronics Consulting & Training GmbH

3.5.5.3 Generierung der Tests



Damit die Skripts ausgeführt werden können, muss auf dem Rechner Ruby installiert sein.

Durch den einfachen Aufbau von Unity und weil der Unity-Programmcode dem Testprogramm als Source hinzugefügt wird, kann natürlich auch ein make für die Kompilierung verwendet werden. Die main()-Funktion muss dann manuell erstellt werden.

3.6 Testauswertung

Testauswertung

Die Informationen über den Testablauf sind in den meisten Fällen sehr knapp gehalten.

Im Erfolgsfall wird nur ausgegeben, dass alles in Ordnung ist.

Im Fehlerfall stehen in der Ausgabe meistens Codedatei und Zeilennummer und zusätzlich der Inhalt der fehlgeschlagenen Assertion. Teilweise wird auch ein Kommentar ausgegeben, wenn er im Quellcode als Parameter der Assertion bereitgestellt wurde (nicht bei jedem Tool verfügbar).

Die Ausgabeformate schwanken je nach Tool von reinem Text bis hin zu XML.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 55

© MicroConsult - MicroElectronics Consulting & Training GmbH

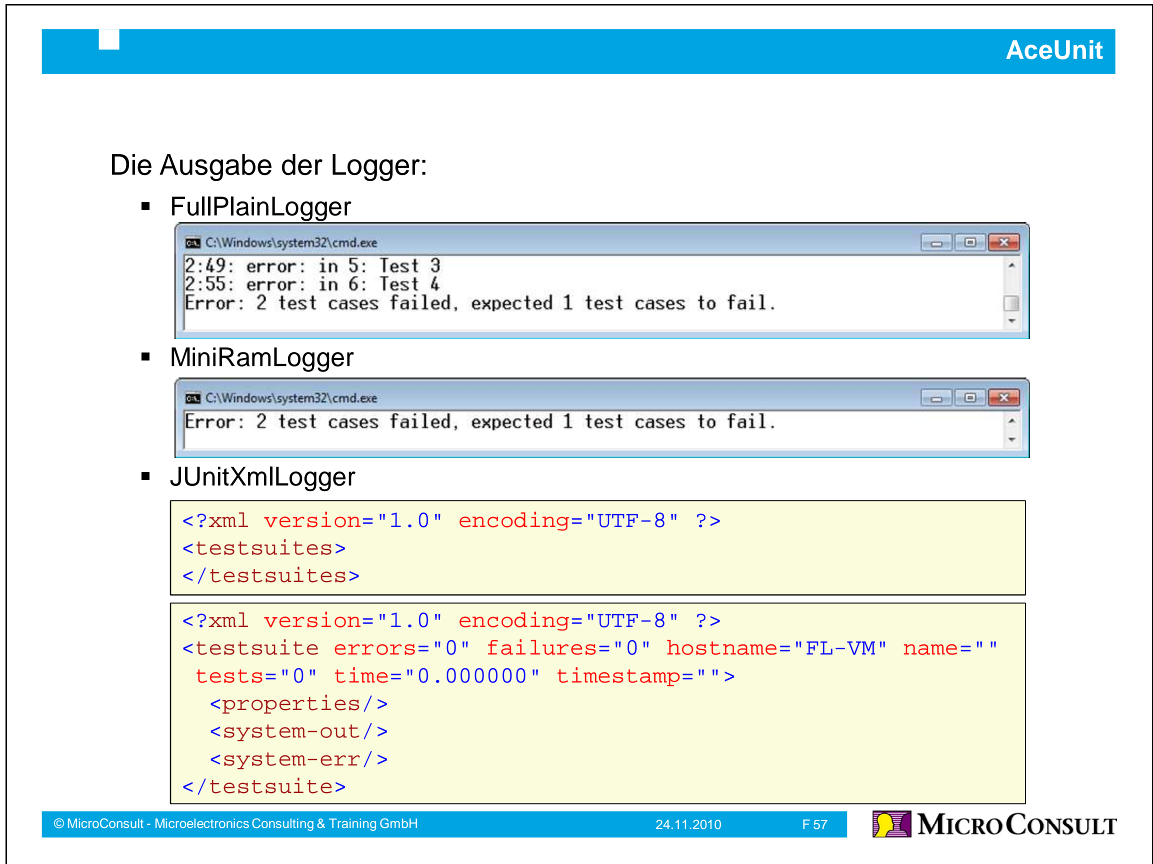
3.6.1 AceUnit

AceUnit

Zur Darstellung der Testergebnisse stellt AceUnit drei Logger bereit:

- *FullPlainLogger*
  Schreibt alle Informationen auf die Konsole.
  Fehlgeschlagene Assertions werden angezeigt.
- *MiniRamLogger*
  Nur verfügbar im embedded Mode (ACEUNIT_EMBEDDED muss definiert sein)
  Schreibt alle Informationen auf die Konsole.
  Es wird nur angezeigt, wie viele Tests fehlgeschlagen sind.
- *JUnitXmlLogger*
  Für Bastler, dieser Logger ist leider nicht fertig. Kann bei Bedarf selbst erweitert werden.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 56 MicroConsult



Der XML-Logger liefert leider keine verwertbaren Ergebnisse.

© MicroConsult - MicroElectronics Consulting & Training GmbH

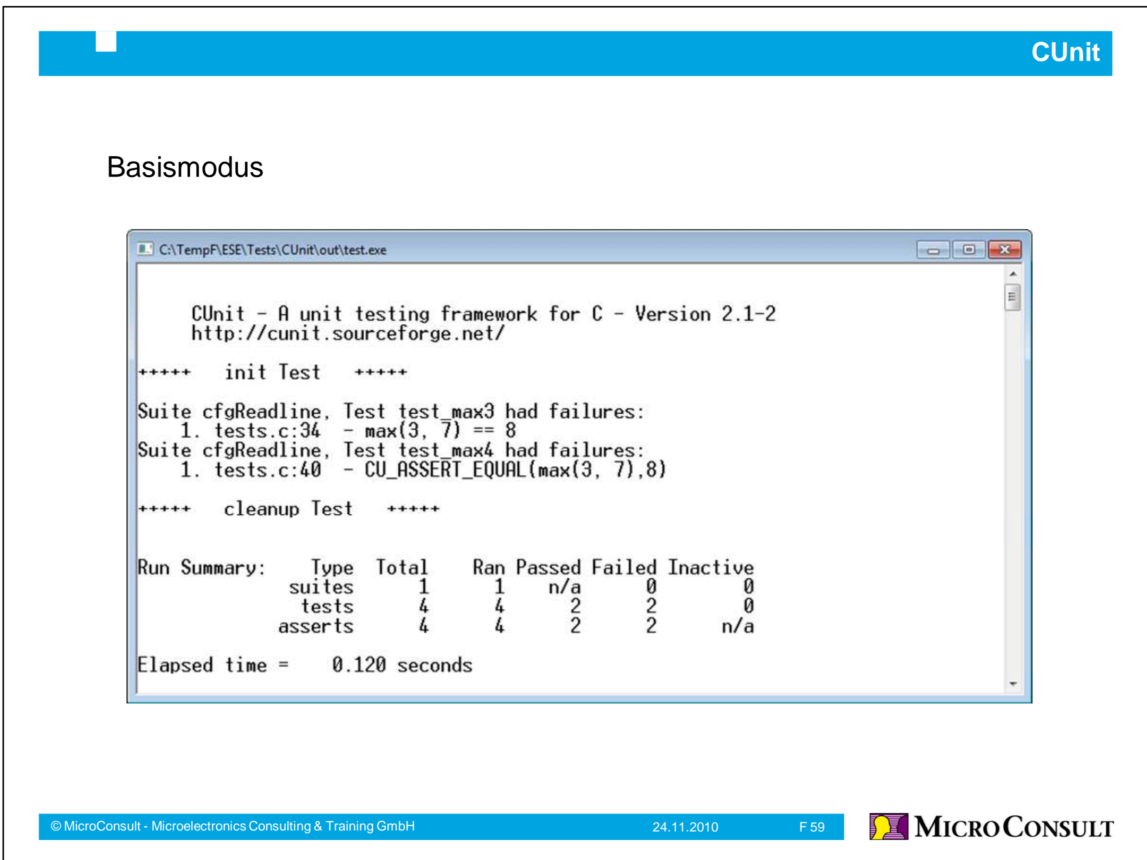
3.6.2 CUnit

CUnit

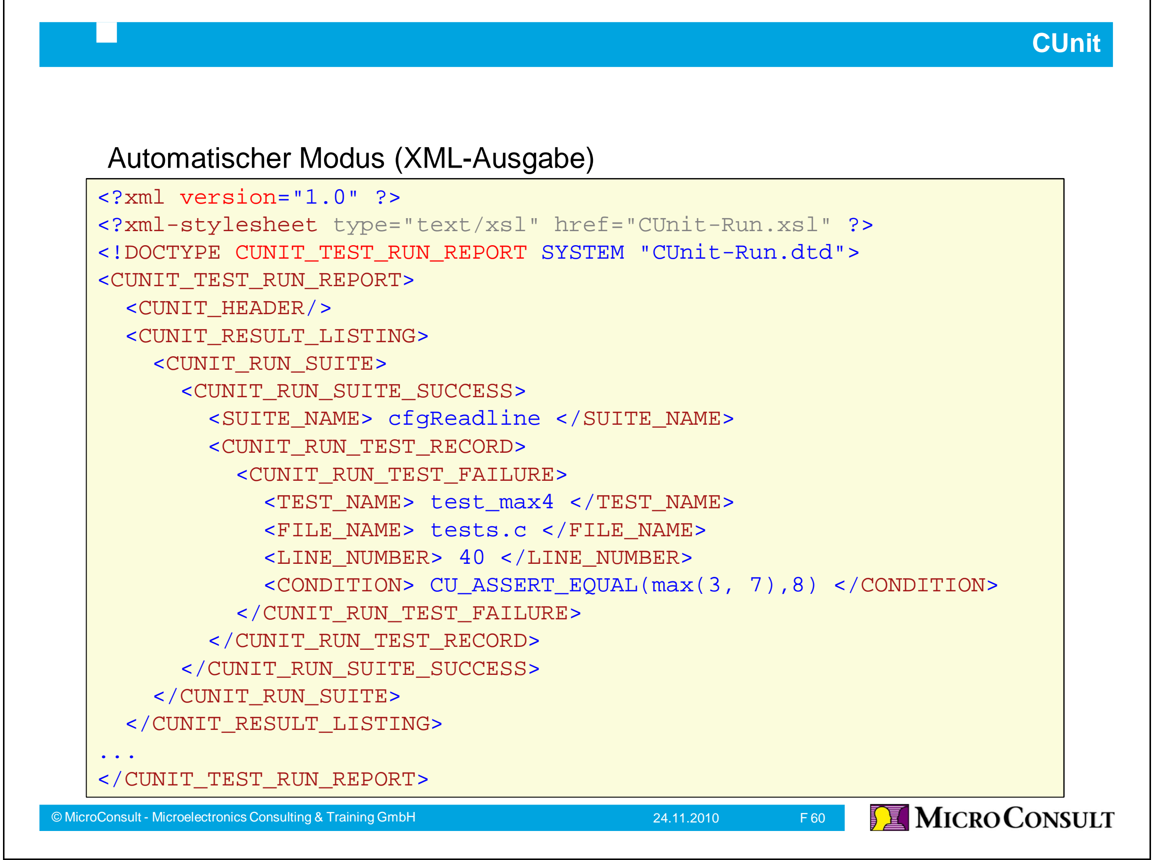
CUnit stellt drei Möglichkeiten zur Ausführung bereit:

- *Basismodus*
  Schreibt alle Informationen auf die Konsole.
  Fehlgeschlagene Assertions und eine Zusammenfassung des Tests werden angezeigt.
- *Automatischer Modus*
  Alle Informationen werden in eine XML-Datei geschrieben.
- *Interaktiver Modus*
  Hier kann zwischen normaler Konsole und Curses (nur unter Linux verfügbar) gewählt werden.
  Der Benutzer bekommt ein Menü zur Auswahl der durchzuführenden Aktion (z.B. Suite auswählen, Test starten, Fehler anzeigen, …).

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 58 MICROCONSULT

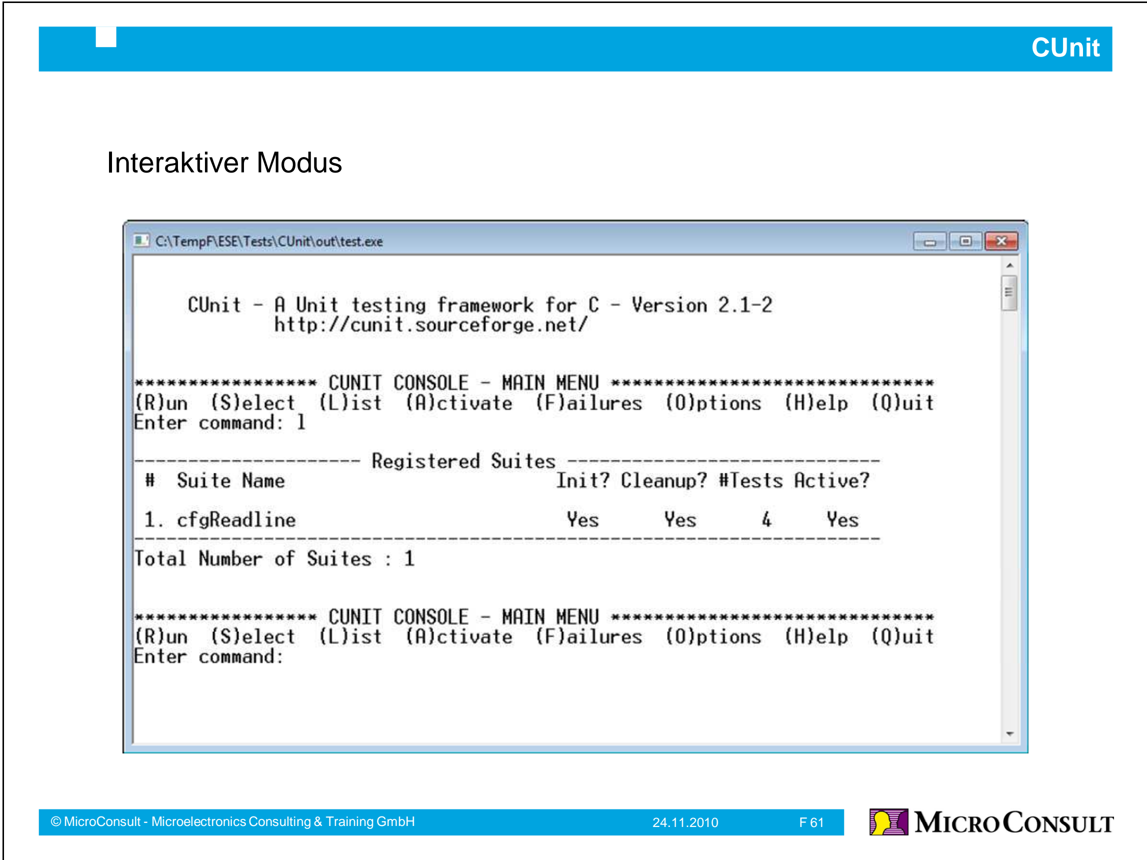


© MicroConsult - MicroElectronics Consulting & Training GmbH


CUnit
Automatischer Modus (XML-Ausgabe)
<?xml version="1.0" ?>
<?xml-stylesheet type="text/xsl" href="CUnit-Run.xsl" ?>
<!DOCTYPE CUNIT_TEST_RUN_REPORT SYSTEM "CUnit-Run.dtd">
<CUNIT_TEST_RUN_REPORT>
<CUNIT_HEADER/>
<CUNIT_RESULT_LISTING>
<CUNIT_RUN_SUITE>
<CUNIT_RUN_SUITE_SUCCESS>
<SUITE_NAME> cfgReadline </SUITE_NAME>
<CUNIT_RUN_TEST_RECORD>
<CUNIT_RUN_TEST_FAILURE>
<TEST_NAME> test_max4 </TEST_NAME>
<FILE_NAME> tests.c </FILE_NAME>
<LINE_NUMBER> 40 </LINE_NUMBER>
<CONDITION> CU_ASSERT_EQUAL(max(3, 7),8) </CONDITION>
</CUNIT_RUN_TEST_FAILURE>
</CUNIT_RUN_TEST_RECORD>
</CUNIT_RUN_SUITE_SUCCESS>
</CUNIT_RUN_SUITE>
</CUNIT_RESULT_LISTING>
...
</CUNIT_TEST_RUN_REPORT>
© MicroConsult - Microelectronics Consulting & Training GmbH
24.11.2010
F 60
MICROCONSULT

CUnit

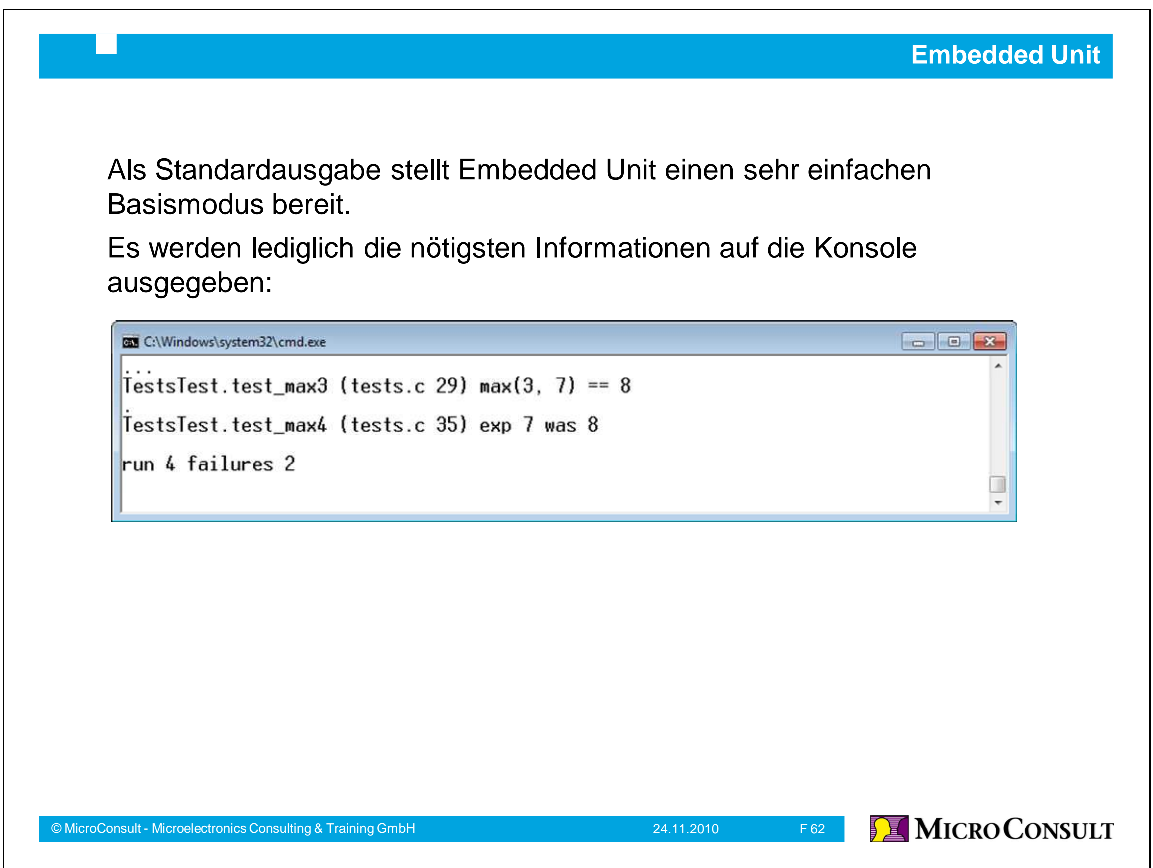
Interaktiver Modus



© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 61 MicroConsult

© MicroConsult - MicroElectronics Consulting & Training GmbH

3.6.3 Embedded Unit



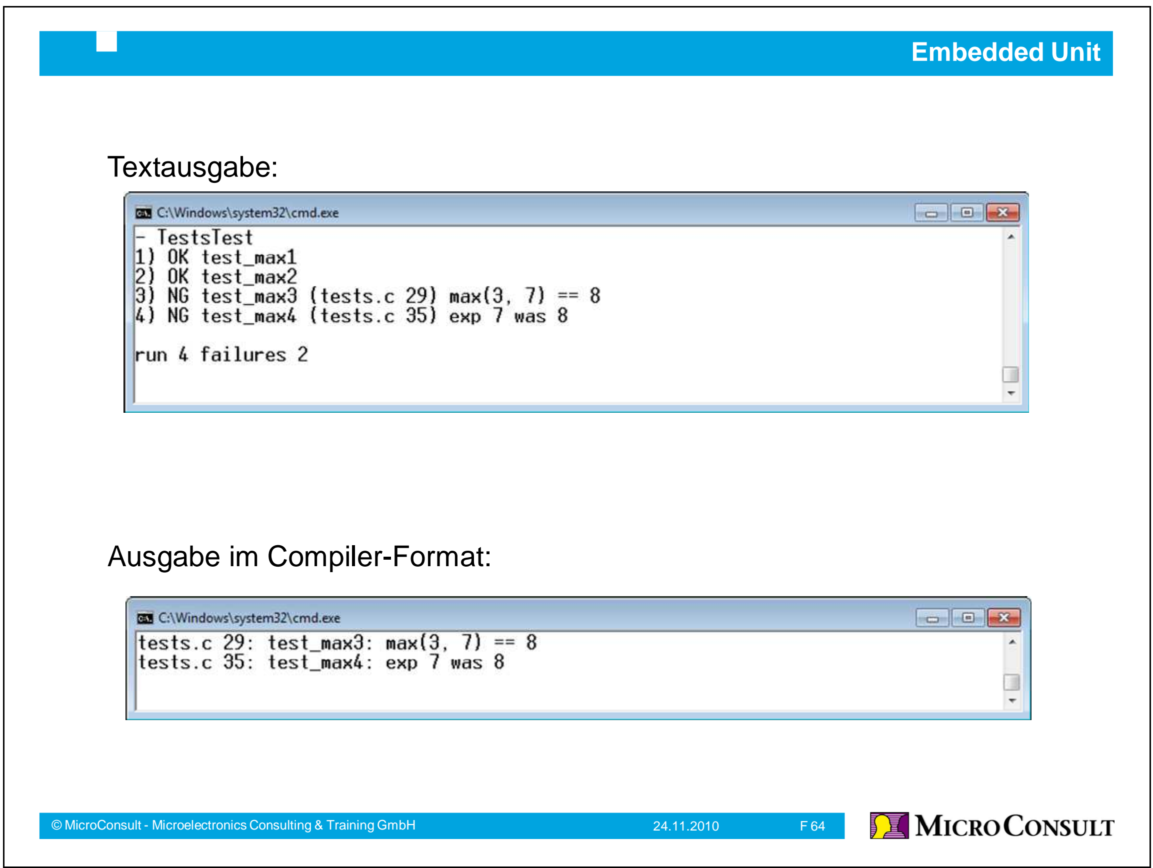
Embedded Unit

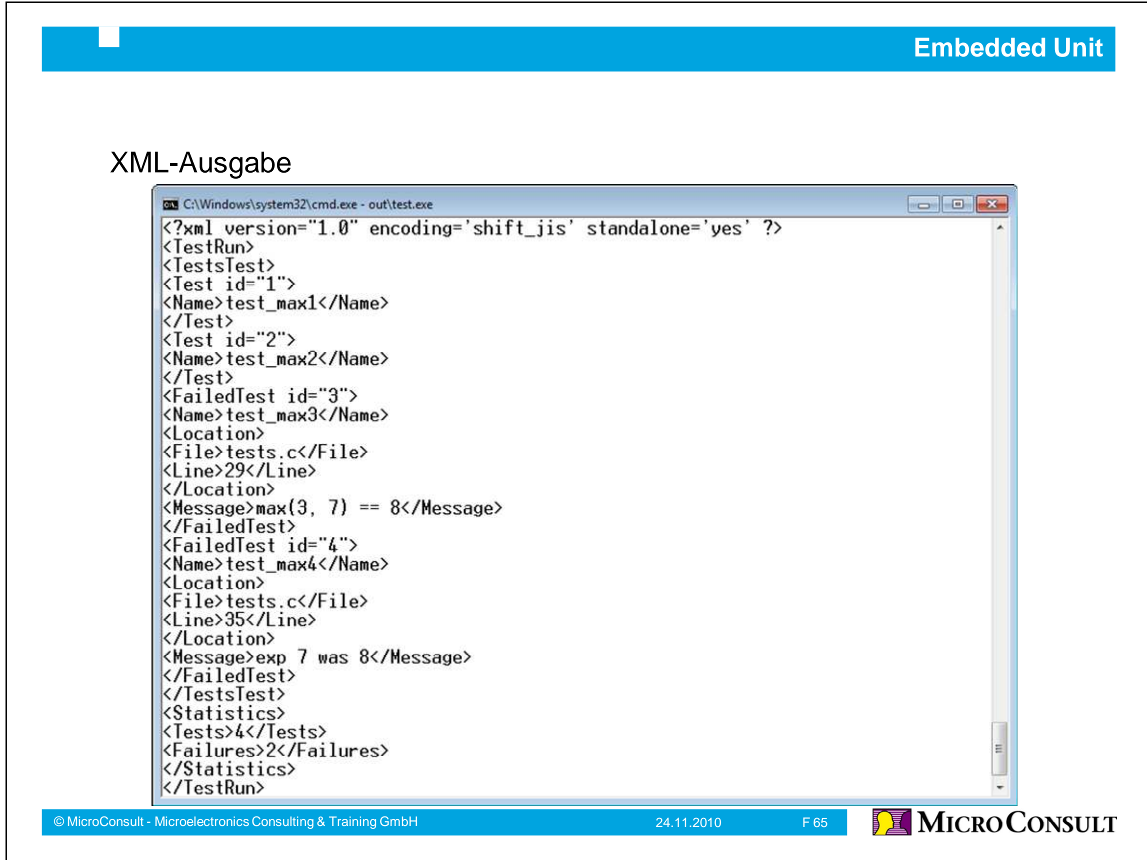
Wird die Bibliothek `libtextui.lib` zum Projekt hinzu gebunden, kommen drei weitere Ausgabeformate hinzu:

- *Textausgabe*
  Schreibt alle Informationen auf die Konsole.
  Fehlgeschlagene Assertions und eine Zusammenfassung des Tests werden angezeigt.
- *Ausgabe im Compiler-Format*
  Schreibt alle Informationen auf die Konsole.
  Das Ausgabeformat ähnelt den Fehlermeldungen eines Compilers.
- *XML-Ausgabe*
  Alle Informationen werden im XML-Format auf die Konsole geschrieben.
  Falls eine Datei mit den Informationen benötigt wird, muss die Ausgabe per Ausgabeumlenkung auf der Kommandozeile in eine Datei umgelenkt werden.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 63

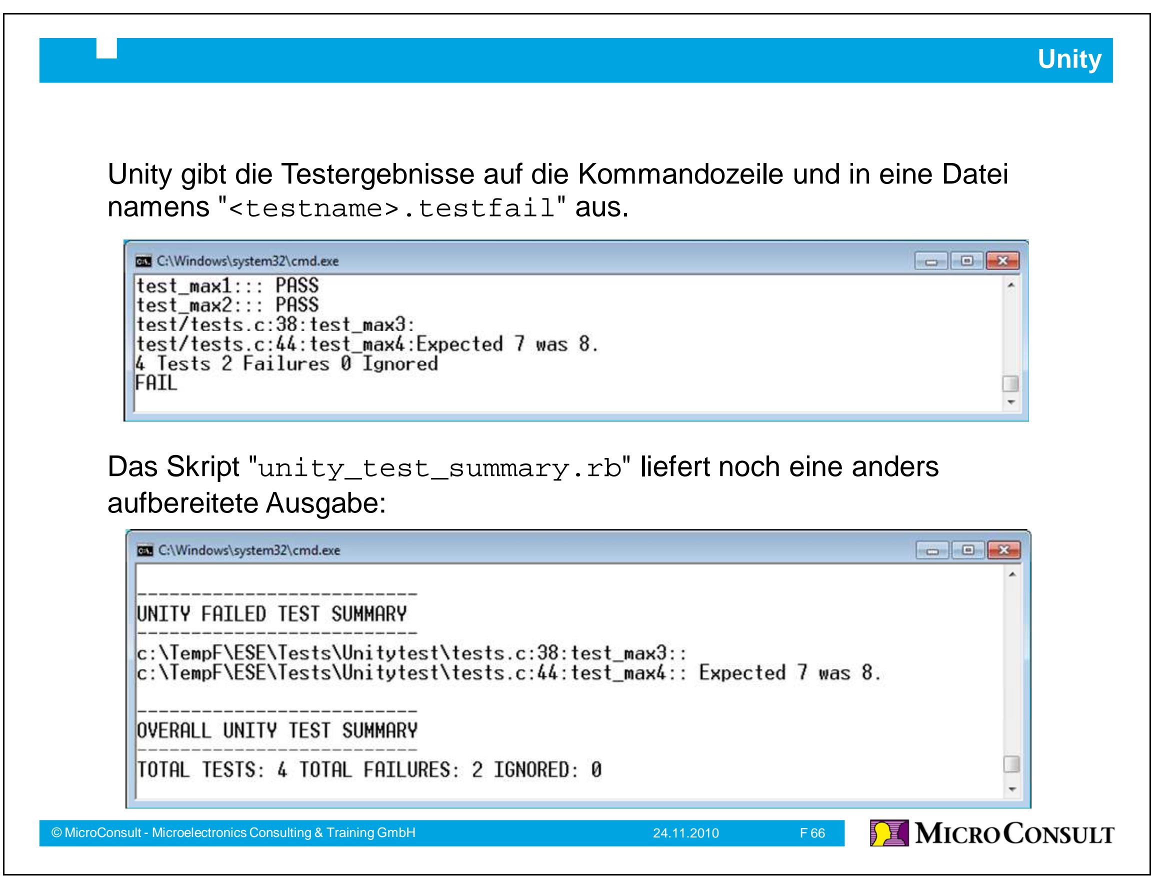
© MicroConsult - MicroElectronics Consulting & Training GmbH


Embedded Unit
Textausgabe:
C:\Windows\system32\cmd.exe
- TestsTest
1) OK test_max1
2) OK test_max2
3) NG test_max3 (tests.c 29) max(3, 7) == 8
4) NG test_max4 (tests.c 35) exp 7 was 8
run 4 failures 2
Ausgabe im Compiler-Format:
C:\Windows\system32\cmd.exe
tests.c 29: test_max3: max(3, 7) == 8
tests.c 35: test_max4: exp 7 was 8
© MicroConsult - Microelectronics Consulting & Training GmbH
24.11.2010
F 64
MicroConsult

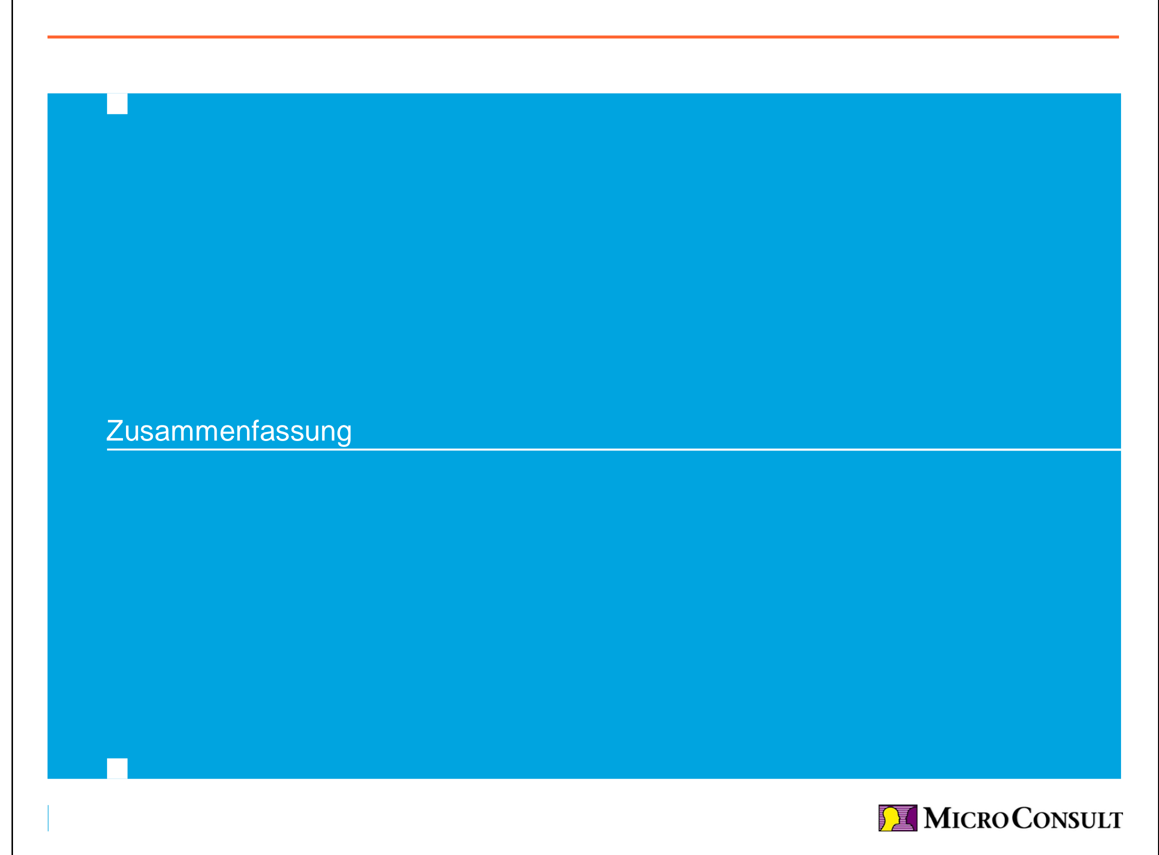


© MicroConsult - MicroElectronics Consulting & Training GmbH

3.6.4 Unity



4 Zusammenfassung



© MicroConsult - MicroElectronics Consulting & Training GmbH

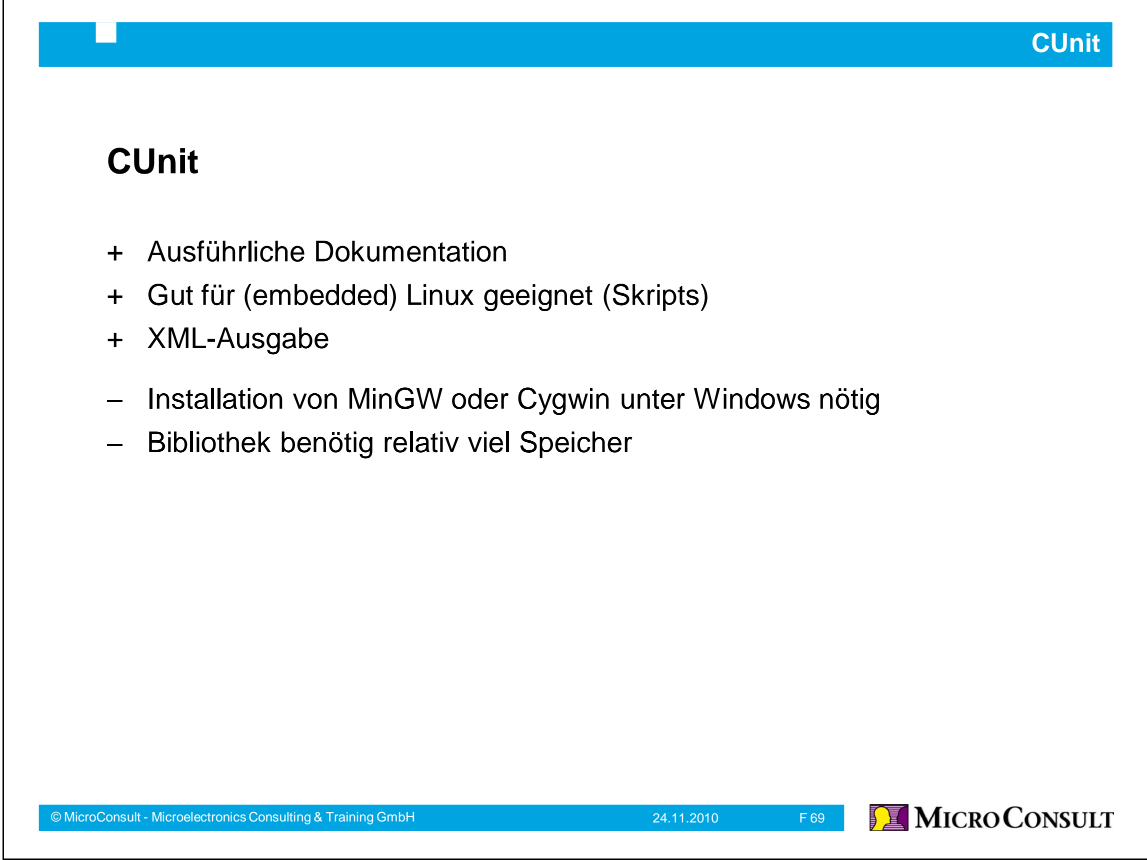
AceUnit

AceUnit

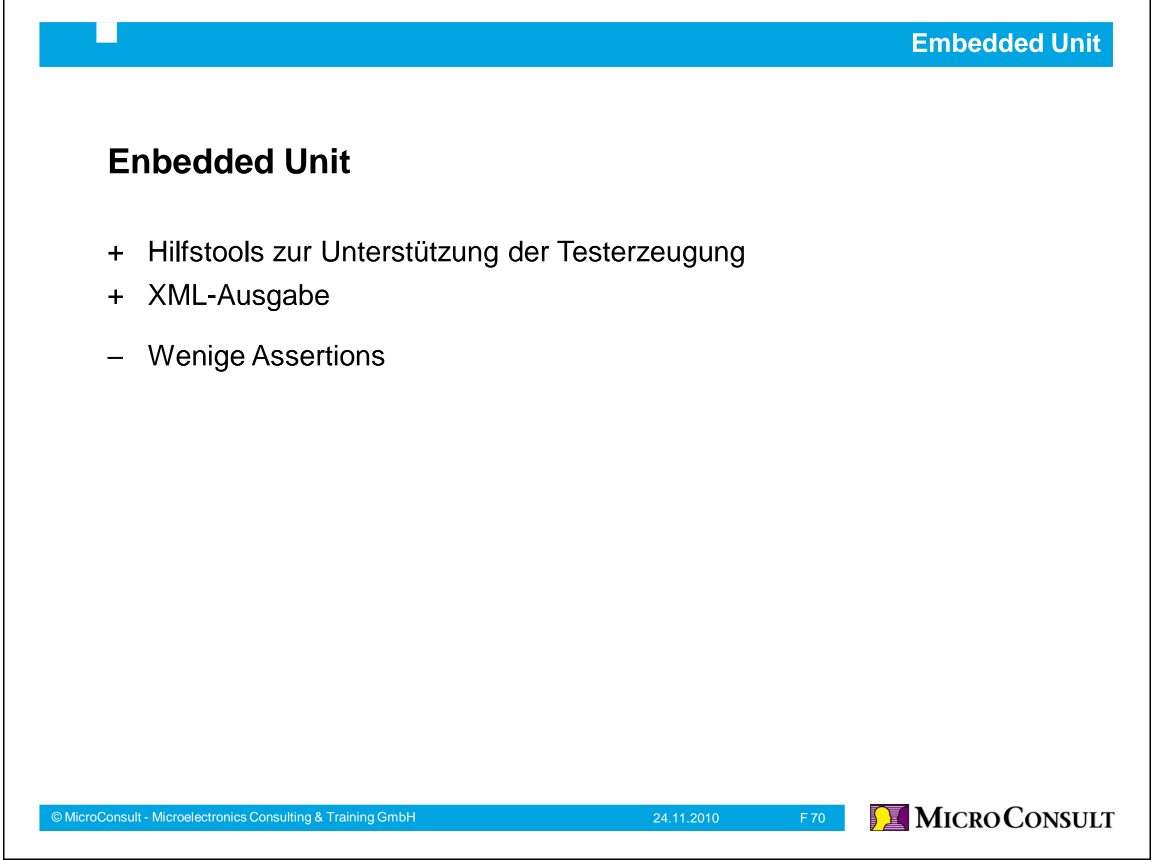
+ Generator zur Unterstützung der Testerzeugung
+ Vorgefertigte main()-Funktion
+ Speichersparender embedded Modus

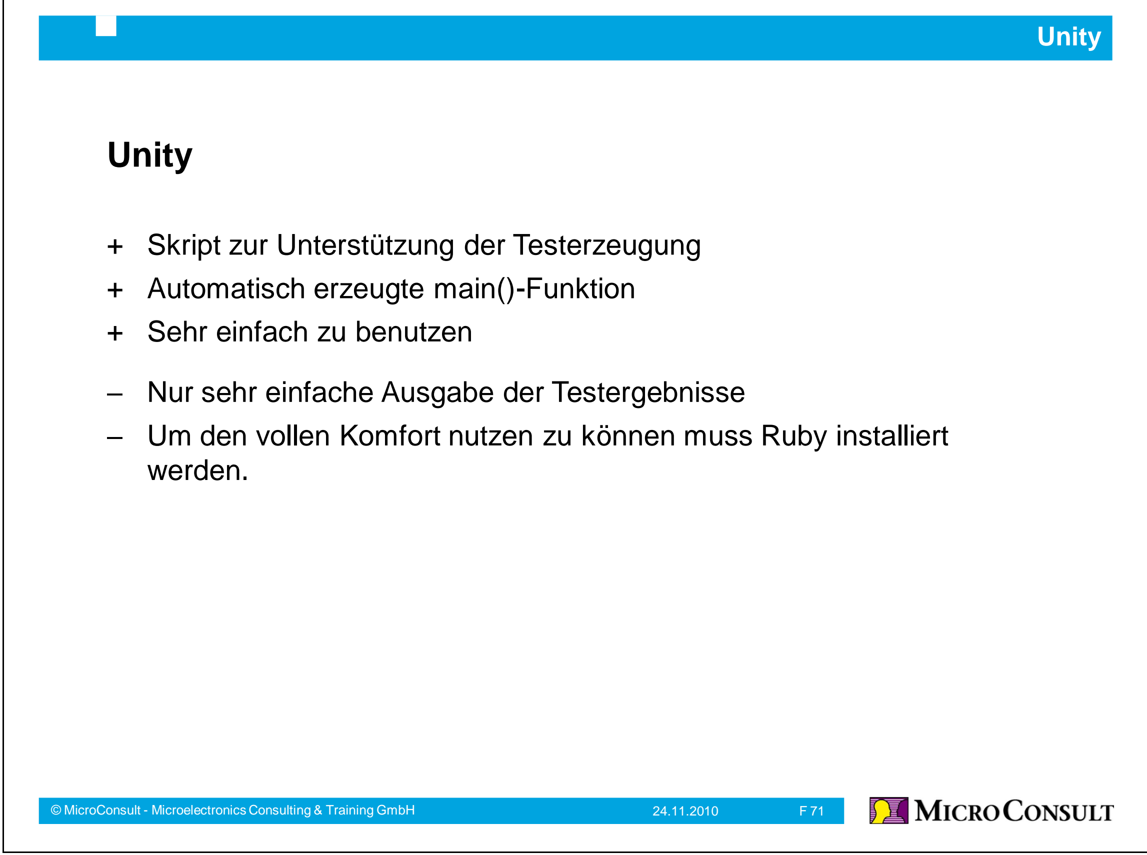
– Sehr spärliche Dokumentation
– XML-Ausgabe unfertig

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 68 MICROCONSULT



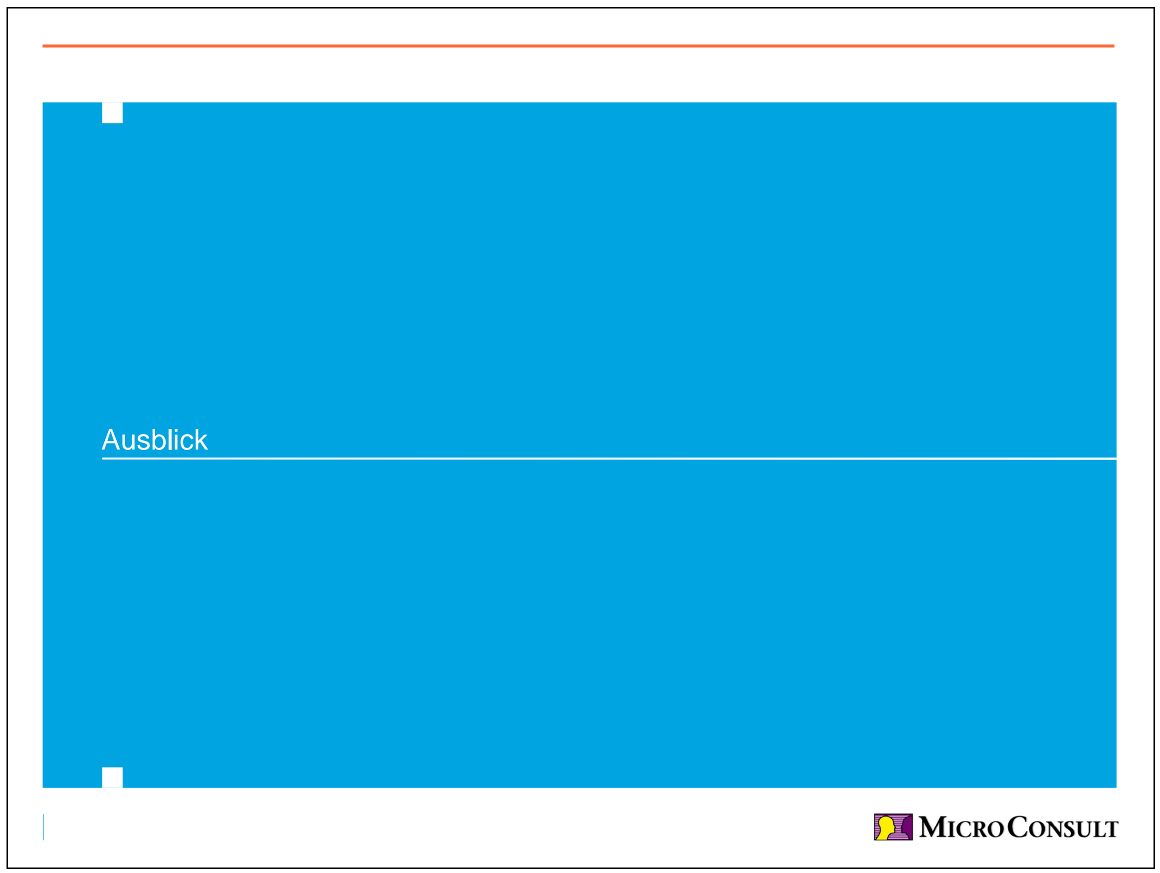
© MicroConsult - MicroElectronics Consulting & Training GmbH


Embedded Unit
Enbedded Unit
+ Hilfstools zur Unterstützung der Testerzeugung
+ XML-Ausgabe
– Wenige Assertions
© MicroConsult - Microelectronics Consulting & Training GmbH
24.11.2010
F 70
MicroConsult



© MicroConsult - MicroElectronics Consulting & Training GmbH

5 Ausblick



Ausblick

Was leisten professionelle Tools?

Neben dem großen Unterschied in der Anschaffung haben professionelle Tools noch einige Merkmale zu bieten:

- Eigene Test-Entwicklungsumgebung mit grafischer Oberfläche und Hilfesystem (z.B. Tessy).
- Integration in die Entwicklungsumgebung (z.B. Cantata++)
- Einfache Testerstellung ohne Skriptsprache oder Programmierung.
- Automatische Generierung von Testtreibern und -stubs.
- Messung der Codeabdeckung der Tests.
- Vielfältige Ausgabeformate für die Testergebnisse. (z.B. XML, HTML, Word, Excel)
- Hilfswerkzeuge für die Testfindung. Z.B. Classification Tree Editor (CTE) bei Tessy.
- Professioneller Support.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 73

© MicroConsult - MicroElectronics Consulting & Training GmbH

Ausblick

Nutzung von C++ anstatt C

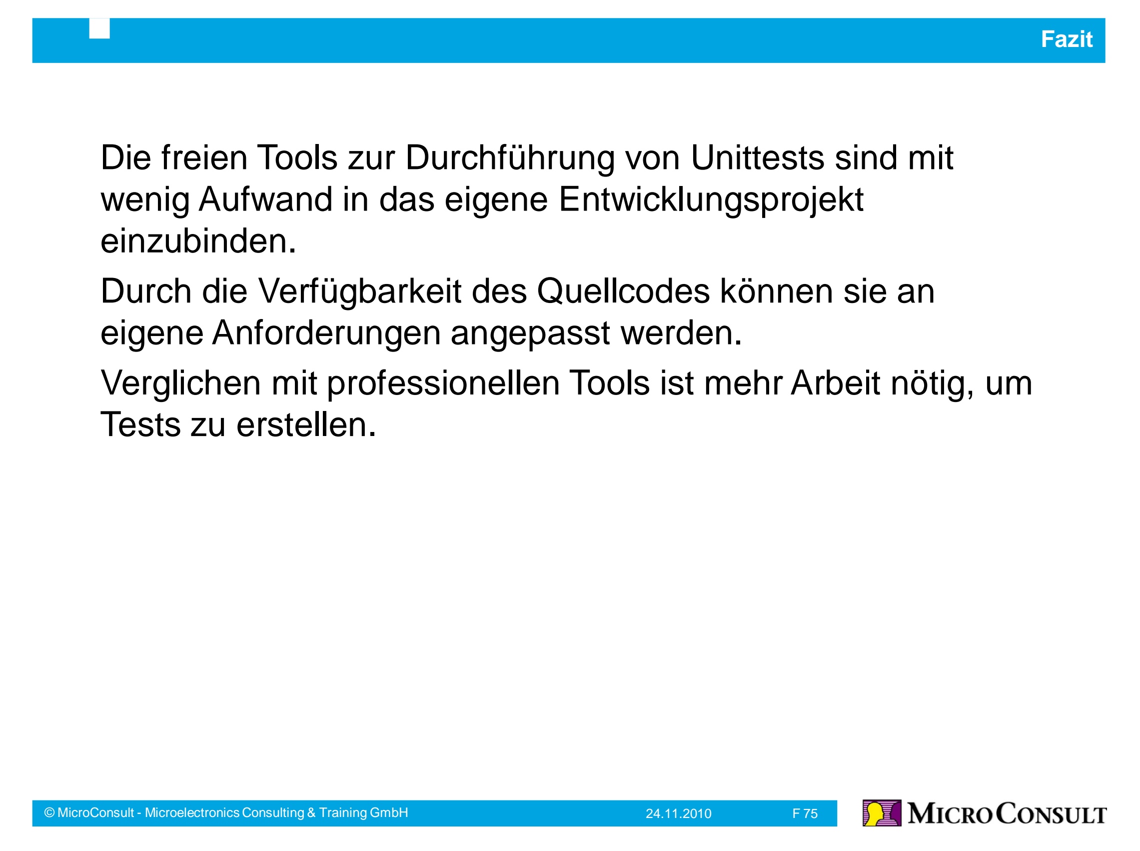
Für C++ ist das Angebot an freien Testtools wesentlich größer als für C. Vor allem was die Leistungsfähigkeit und den Komfort angeht.

Ein prominentes Beispiel ist CppUnit.

Ein weiteres Beispiel ist das Programm CUTE (C++ Unit Testing Easier) vom Institut für Software der Hochschule für Technik Rapperswil (Schweiz):

Es integriert sich in Eclipse und bietet Assistenten für die Testerstellung und Testnavigatoren für schnellen Überblick über Tests und Testergebnisse.

© MicroConsult - Microelectronics Consulting & Training GmbH 24.11.2010 F 74 MicroConsult



© MicroConsult - MicroElectronics Consulting & Training GmbH