HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

IFS INSTITUTE FOR SOFTWARE

Testframework-Migration für COAST

Bachelorarbeit in Informatik
Bachelorstudium
an der Hochschule für Technik Rapperswil
der Fachhochschule Ostschweiz

Autor:
David Tran
Juni 2015

Betreuer:
Mirko Stocker, Prof. Hans Rudin

Gegenleser:
Prof. Dr. Eduard Glatz

Experte:
Daniel Hiltebrand, Crealogix

Erstellung 16. Februar 2015

Letzte Aktualisierung 11. Juni 2015

Autor: David Tran

Betreuer: Mirko Stocker, HSR, IFS/Prof. Hans Rudin, HSR, IFS

Gegenleser: Prof. Dr. Eduard Glatz, HSR

Experte: Daniel Hiltebrand, Crealogix

Weitere Informationen im Web:

http://www.hsr.ch/

Aufgabenstellung Bachelorarbeit für David Tran „Testframework-Migration für COAST“

1. Betreuer und Experte

Diese Bachelorarbeit wird für das Institut für Software (IFS) der HSR durchgeführt.

Betreuer:

- Mirko Stocker, HSR, IFS, m1stocke@hsr.ch
- Prof. Hans Rudin, HSR, IFS hrudin@hsr.ch (verantwortlicher Dozent für Beurteilung)

Experte:

- Daniel Hiltebrand, Crealogix

2. Studierender

Diese Arbeit wird als Bachelorarbeit an der Abteilung Informatik durchgeführt von

- David Tran, d2tran@hsr.ch

3. Einführung

COAST, das C++ Open Application Server Toolkit (http://coast-project.org) ist ein bewährtes Framework, um Web-Applikationen in C++ zu entwickeln. Das Framework besitzt eine sehr hohe Test-Abdeckung durch automatisierte Unit- Tests, allerdings verwenden diese Tests ein selbst entwickeltes Test-Framework und keine Standardlösung. Um die Wartbarkeit von COAST zu verbessern, soll deshalb das Testframework durch CUTE http://cute-test.com ersetzt und die vorhandenen Testfälle nach CUTE migriert werden.

4. Ziele der Arbeit

Das Ziel dieser Arbeit ist nicht, komplett alle Testfälle zu migrieren, sondern eine Migrationsanleitung und Dokumentation zu schreiben und diese exemplarisch an (möglichst vielen) Tests anzuwenden. Eine Automatisierung repetitiver Aufgaben, z.B. durch den Einsatz regulärer Ausdrücke, soll untersucht werden.

5. Aufgabenstellung

Diese Arbeit umfasst die folgenden Schritte:

- Setup von COAST in Eclipse unter Linux (z.B. in einer virtuellen Maschine)
- Analyse des alten COAST Test-Frameworks und Vergleich mit CUTE
- Migrationsanleitung, wie die Testfälle auf CUTE portiert werden können und welche Besonderheiten (z.B. durch Unterschiede in den Test-Frameworks verursacht) beachtet werden müssen.
- Migration von Tests und Dokumentation

Details sowie die Gewichtung der Teilaufgaben werden im Verlauf der Arbeit mit den Betreuern vereinbart.

6. Zur Durchführung

Es finden wöchentliche Besprechungen mit den Betreuern statt, d.h. in der Regel mit Mirko Stocker, zum Teil wird auch Hans Rudin teilnehmen. Zusätzliche Besprechungen sind nach Bedarf durch David Tran zu veranlassen.

Alle Besprechungen, ausser der Kick-off Besprechung, sind von David Tran mit einer Traktandenliste vorzubereiten und zu leiten. Als erstes Traktandum soll immer der Stand des Projektes präsentiert werden (Was wurde gemacht? Was wurde erreicht? Wie viel Zeit wurde in was investiert?). Die Ergebnisse der Besprechungen sind durch David Tran zu protokollieren und an Mirko Stocker und Hans Rudin spätestens am folgenden Tag per Mail zuzustellen.

Für die Durchführung der Arbeit ist ein Projektplan zu erstellen. Dabei ist auf einen kontinuierlichen und sichtbaren Arbeitsfortschritt zu achten. An Meilensteinen (oder auch Zwischenversionen) gemäss Projektplan sind einzelne Arbeitsresultate in vorläufigen Versionen abzugeben. Über die abgegebenen Arbeitsresultate erhält der Studierende ein vorläufiges Feedback. Eine definitive Beurteilung erfolgt auf Grund der am Abgabetermin abgelieferten Resultate.

7. Dokumentation

Über diese Arbeit ist eine Dokumentation gemäss den Richtlinien der Abteilung Informatik zu verfassen (siehe https://www.hsr.ch/Allgemeine-Infos-Diplom-Bach.4418.0.html). Die zu erstellenden Dokumente bzw. Berichtsteile sind im Projektplan festzuhalten. Alle Dokumente sind nachzuführen, d.h. sie sollten den Stand der Arbeit bei der Abgabe in konsistenter Form dokumentieren. Alle Resultate sind vollständig auf CD/DVD in 3 Exemplaren abzugeben.

8. Termine

Siehe auch Terminplan auf https://www.hsr.ch/Termine-Diplom-Bachelor-und.5142.0.html.

Montag, den 16. 2. 2015	**Beginn der Bachelorarbeit, Ausgabe der Aufgabenstellung**
Dienstag, 17.2. 10:00	Kick-off Besprechung im IFS
5.6.2015	Abgabe Kurzbeschreibung und A0-Poster an Betreuer zur Überprüfung Vorlagen stehen unter den allgemeinen Infos Diplom-, Bachelor- und Studienarbeiten zur Verfügung.
12.6.2015, 12:00	**Abgabe der Arbeit an den Betreuer bis 12.00 Uhr Fertigstellung des A0-Posters bis 12.00 Uhr**
12.6.2015	Präsentation der Bachelorarbeiten, 16 bis 20 Uhr
3.8.-21.8.2015	Mündliche Prüfung zur Bachelorarbeit
25.9.2015	Bachelorfeier und Ausstellung der Bachelorarbeiten

9. Beurteilung

Eine erfolgreiche Bachelorarbeit zählt 12 ECTS-Punkte pro Studierenden. Für 1 ECTS Punkt ist eine Arbeitsleistung von 30 Stunden budgetiert (Siehe auch Modulbeschreibung der Bachelorarbeit https://unterricht.hsr.ch/staticWeb/allModules/19419_M_BAI.html.).

Für die Beurteilung ist der verantwortliche Dozent zuständig.

Gesichtspunkt	Gewicht
1. Organisation, Durchführung	1/6
2. Berichte (Abstract, Mgmt Summary, technischer u. persönliche Berichte) sowie Gliederung, Darstellung, Sprache der gesamten Dokumentation	1/6
3. Inhalt*)	3/6
4. Mündliche Prüfung zur Bachelorarbeit	1/6

*) Die Unterteilung und Gewichtung von 3. Inhalt wird im Laufe dieser Arbeit präzisiert.

Im Übrigen gelten die Bestimmungen der Abteilung Informatik für Bachelorarbeiten.

Rapperswil, den 14. Februar 2015

Mirko Stocker

Wissenschaftlicher Mitarbeiter
Institut für Software (IFS)
Hochschule für Technik Rapperswil

Hans Rudin

Professor für Informatik
Institut für Software (IFS)
Hochschule für Technik Rapperswil

Erklärung

Ich erkläre hiermit,

- dass ich die vorliegende Arbeit selber und ohne fremde Hilfe durchgeführt habe, ausser derjenigen, welche explizit in der Aufgabenstellung erwähnt ist oder mit dem Betreuer schriftlich vereinbart wurde,
- dass ich sämtliche verwendeten Quellen erwähnt und gemäss gängigen wissenschaftlichen Zitierregeln korrekt angegeben habe.
- das ich keine durch Copyright geschützten Materialien (z.B. Bilder) in dieser Arbeit in unerlaubter Weise genutzt habe.

Ort, Datum: Rapperswil, 11. Juni 2015

David Tran: 

Abstract

COAST (C++ Open Application Server Toolkit) ist ein Framework für Webapplikationen mit eigenem Testframework. CUTE (C++ Unit Testing Easier) 4.11, Header 2.0.0, ist ein alternatives Testframework um Unit Tests für C++-Code zu schreiben. Beide Frameworks entwickelt das Institut für Software (IFS). CUTE wird Standalone und als Eclipse CDT Plugin veröffentlicht. In dieser Arbeit ist eine Migrationsanleitung zu erstellen. Die COAST Tests sind damit auf CUTE zu migrieren.

Damit die Tests in CUTE ablaufen, sind verschiedene Implementierungen anzupassen: Die Member Funktionen ändern, die „test fixture“ sind in die ctor, dtor und ctor Initialisierung verschoben. Dazu kommt eine neue Testsuite, welche die Tests sammelt. Die CUTE Testsuite startet die Tests nacheinander, worin das CUTE dem COAST Testframework sehr ähnlich ist. Die Implementation wurde angepasst ans CUTE und nachdem die Unit Tests getestet wurden, stellt es die Assertionen mit der Auswertung im Terminal und in einer JUnit XML codiert bereit. Das CUTE Plugin zeigt das Ergebnis in der grafischen Benutzeroberfläche (GUI) an.

Das erste Resultat ist die Migrationsanleitung, da die Testframeworks von COAST und CUTE unterschiedliche Assertionen haben. Die COAST Assertionen wurden spezifisch für COAST gebaut. CUTE hat allgemeinere Assertionen, wobei viele identisch zum COAST Testframework sind und nur leicht unterschiedlich implementiert sind. Das zweite Resultat sind CUTE Assertionen für die COAST-Klasse Anything, diese befinden sich in den CUTE Extensions. Die Extensions Implementationen testen hauptsächlich Funktionen, die mit CUTE Assertionen fehlschlagen und einen zusätzlichen Aufbau um die CUTE Assertion herum benötigen.

Management Summary

Ausgangslage

COAST hat ein Testframework und CUTE ist ein Testframework, die etwas gemeinsam haben. Beide testen C++ mit Unit Tests. Die Frameworks entwickelt das Institut für Software (IFS).

CUTE

Das Testframework CUTE (C++ Unit Testing Easier) testet C++. Das gibt es für Eclipse, Microsoft Visual Studio und Standalone auf der Webpräsenz http://www.cute-test.com. Die Standalone Veröffentlichung ist für die Migration verwendet. Dabei sind CUTE Header, Unit Tests und Beispiele. Für Eclipse CDT gibt es das CUTE Plugin. Einen Test Navigator kommt dazu. Die Eclipse View hat die Bezeichnung: „Test Results“.

- Abbildung 1 Es zeigt das Ergebnis mit einem grünen Balken und bedeutet, dass der Test erfolgreich ist.
- Abbildung 2 Der Test hat einen Fehler gefunden, darum hat es einen roten Balken.
- Abbildung 3 Es ist ein Vergleich der Ergebnisse.

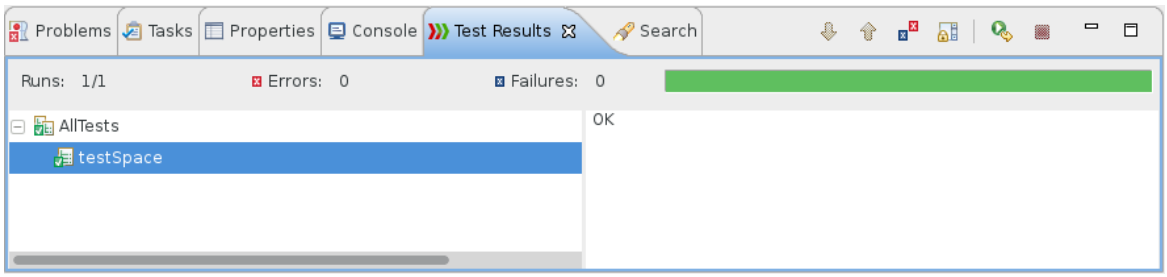


Abbildung 1 Testresultat: OK

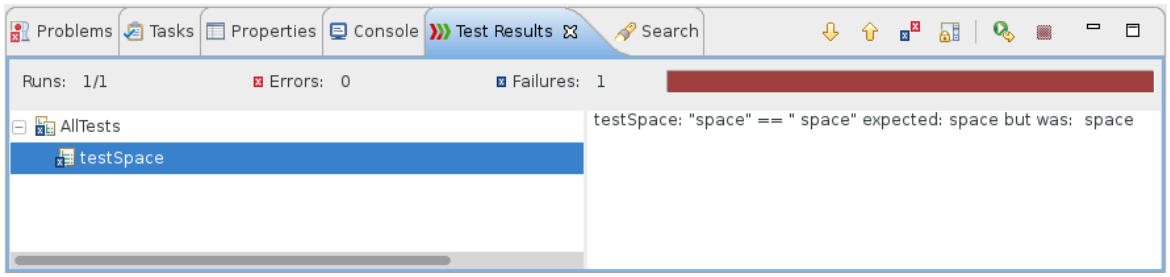


Abbildung 2 Testresultat: Fehler

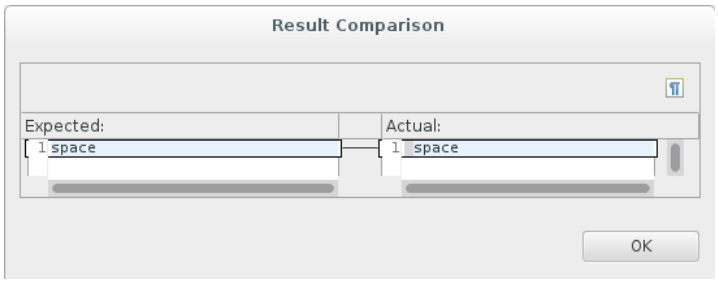


Abbildung 3 Resultate vergleichen

COAST (Testframework)
Das COAST (C++ Open Application Server Toolkit) ist ein Framework für Webapplikationen. Bisher gab es keine Veröffentlichung. Die Quelltexte sind auf der Webpräsenz http://www.coast-project.org und in C++ geschrieben. Das COAST Testframework testet mit mindestens 10000 Tests das Framework. Die Tests sind ebenfalls in C++.

Technologien

CUTE
Für COAST wurde CUTE Standalone gewählt, die Header vorbereitet wurden. Zusätzliche Implementierungen braucht es, damit CUTE Tests kompilieren und testen. Dieses Testsystem erledigt die Durchläufe. Das Setup braucht das CUTE nur am Anfang. Der Verlauf ist mit dem COAST Testframework vergleichbar. Die Auswertung erfolgt im Terminal und in eine JUnit XML codiert.

COAST Testframework
Nähere Erläuterungen zur Migration sind in der Migrationsanleitung. Diese beschreibt den Vorgang und die Ausnahmen. Jede Änderung ist Schritt für Schritt aufgeführt. Die Migrationsanleitung ist für einen Experten verständlich. Am besten zuerst die Testdateien kopieren und in verschiedene Verzeichnisse aufteilen. Das Testframework anpassen ist einfach mit der Migrationsanleitung. Die richtigen Reguläre Ausdrücke (RegEx) auszuwählen, ist schwierig, es geht nur durch ausprobieren und nachkorrigieren. Die neuen Assertionen um die Klasse Anything (ANY) sind nur einzusetzen, wenn die gleichwertige CUTE Assertion versagt. Das passiert nur selten.

- ASSERT_ANY_EQUAL
- ASSERT_ANY_EQUALM
- ASSERT_ANY_COMPARE

Assertionen (ASSERT) sind Member Funktionen, die Werte vergleichen (EQUAL/COMPARE) und entweder richtig oder falsch ergibt. Der eine Wert ist der erwartete Wert und der andere Wert ist oft der aktuelle Wert. Der Kommentar (+M) beschreibt einen Fehler.

Ein Beispiel für eine Migration ohne RegEx (links: COAST, rechts: CUTE)

assertEqual(expect, actual)	ASSERT_EQUAL(expect, actual)
assertEqualm(expect, actual, message)	ASSERT_EQUALM(message, expect, actual)

Ein Beispiel für eine Migration mit RegEx (links: COAST, rechts: CUTE)

ADD_CASE\s*\(\s*(.*\s*,)\s*(.*,.*)\)	s.push_back(CUTE_SMEMFUN(\2))

Abbildung 4 Im Terminal läuft eine Auswertung ab.

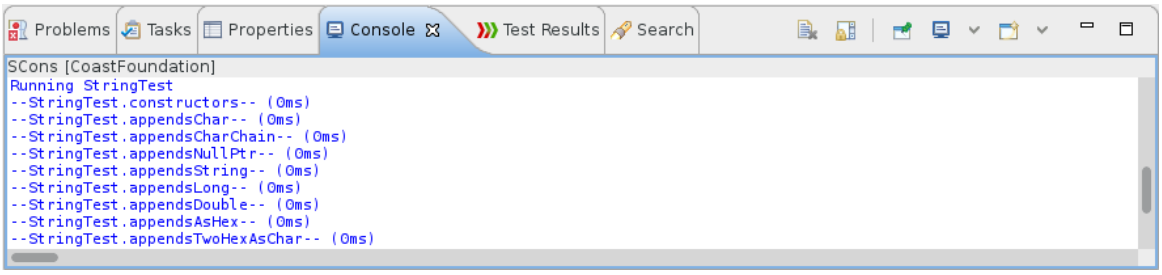


Abbildung 4 Terminal Testresultate

Ergebnisse

Das COAST Framework und das COAST Testframework sind unverändert stehen gelassen. Die migrierten Tests/Testfälle sind 329 in 31 Klassen und 3 Assertionen. Im Vergleich sind die Assertionen häufig sehr ähnlich und inhaltlich gleich. Die CUTE Tests laufen separiert von den COAST Tests. Die Testresultate können in der Results View dargestellt werden. Wenn das Testsystem bereits gestartet wurde, kann mit dem erstellten Skript gearbeitet werden. Ist Eclipse damit eingestellt, dann das CUTE Projekt starten und der Test Navigator stellt das letzte Ergebnis dar.

Mithilfe der Reguläre Ausdrücke (RegEx) vereinfacht es die Migration. Viele herausgefundene Reguläre Ausdrücke finden die richtigen Assertionen und passen diese richtig an. Es gibt ineinander verschachtelte Argumente, die können RegEx nicht ersetzen. Wenn die Migrationsanleitung bei einer Assertion versagt, gibt es keine Lösung und muss separat geregelt werden. Damit alle Tests mit CUTE ablaufen, sind zusätzliche Assertionen implementiert und CUTE Extensions genannt. Eine Installationsanleitung ist dabei, falls mehr Tests migriert werden sollen.

Abbildung 5 Die Auswertung mit CUTE im Terminal.

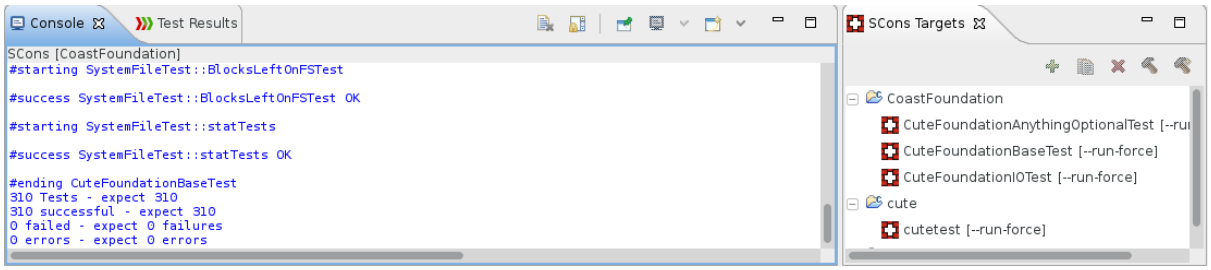


Abbildung 5 CUTE Terminal Testresultate

Inhaltsverzeichnis

Kapitel I Technischer Bericht

1 CUTE

1.1 Ausgangslage

CUTE ist die Abkürzung für C++ Unit Testing Easier. Die Anwendung testet C++ Quelltexte. Das Ziel ist CUTE Tests mit SConsider als Testsystem. Eine Testsuite durchläuft alle Testfälle.

1.1.1 Ziele

- CUTE Testfälle kennenlernen
- Testsystem mit SConsider erstellen

1.1.2 Technologien

- CUTE Headers 2.0.0 (CUTE 4.11)
- SCons 2.3.0
- SConsider

1.2 Vorgehen

1. Setup CUTE und Sconsolidator in Eclipse unter Ubuntu.
2. CUTE Testframework einarbeiten.
3. SCons und SConsider installieren.

1.3 Problembeschreibung

CUTE ist ein Testframework, das Open Source ist. Es gibt es für Eclipse, Visual Studio oder Standalone. Bestehende Eclipse C++ Projekte konvertiert CUTE in ein CUTE Projekt. CUTE besitzt einen Test Navigator, der das Testresultat zusammenfasst und in der Eclipse Konsole darstellt. Der Statusbalken ist rot und steht für fehlerhaft und ist grün, wenn alle Tests erfolgreich abgelaufen sind. Die Tests sind einzeln aufgelistet und auch die Testsuites. Die Fehlermeldung beschreibt den Wert, welcher werden soll und der erreichte Wert während dem Testablauf. Ansonsten steht nur „ok“. Die Eclipse View „Test Results“ gibt die Tests aus, welche unter den Bedingungen (Erfolg/Misserfolg/Fehler) durchlaufen sind.

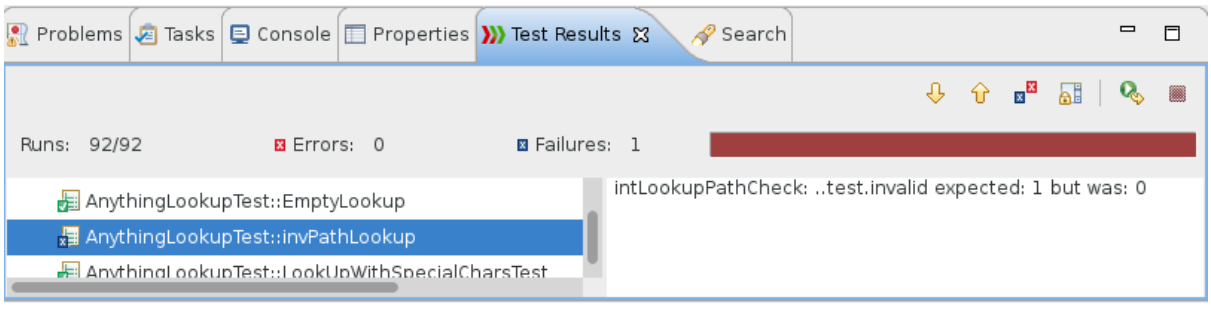


Abbildung 6 Eclipse Test Navigator (Test Results View)

SConsider, die SCons Erweiterung, ist ein Werkzeug, welches das System erstellt und kompiliert. SCons kopiert die zu verarbeitenden Dateien in ein neues Verzeichnis und SConsider kompiliert. SConsider hat eine eigene Konfigurationsdatei und Syntax. Sconsolidator ist das Eclipse Plugin dazu und hat eine grafische Benutzeroberfläche.

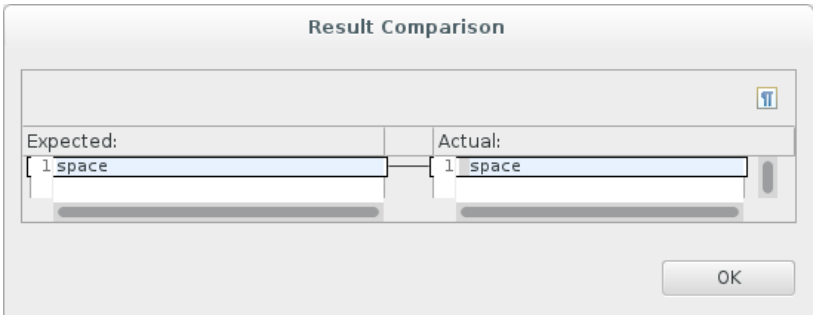


Abbildung 7 Doppelklick auf die Fehlermeldung erscheint ein neuer Dialog

1.4 Lösungskonzept

CUTE Projekt mit einer Test Member Funktion und Assertion, die immer richtig ist:

```
#include "cute.h"
#include "ide_listener.h"
#include "cute_runner.h"

void thisIsATest() {
	ASSERT(true);
}

void runAllTests(int argc, char const *argv[]) {
	cute::suite s;
	s.push_back(CUTE(thisIsATest));
	cute::ide_listener<> lis;
	cute::makeRunner(lis)(s, "AllTests");
}

int main(int argc, char const *argv[]) {
	runAllTests(argc, argv);
	return 0;
}
```

1.5 Umsetzung

- Setup CUTE in Eclipse unter Linux (z.B. in einer virtuellen Maschine)
- CUTE erweitert
- SConsider Konfigurationsdatei für CUTE erstellt

1.6 Resultate: Bewertung und Zielerreichung

Zu den CUTE Header sind neue Quelltexte hinzugekommen. Die Header Datei cute_case.h hat eine Deklaration, die Implementiert werden soll.

```
void setupSuite(cute::suite &s);
```

Die Klasse mit der main Methode besitzt die Testsuite und den Runner.

```
cute::suite s;
setupSuite(s);
cute::ide_listener<> lis;
cute::makeRunner(lis)(s, "AllTests");
```

Die SConsider Dateien sind nur ergänzt, wenn es nötig war. Der Eintrag ist ohne Parameter 'runParams':'', weil der Parameter „-- -all“ implementiert ist. In SConsider sieht es so aus:

```
'runConfig': {
        'runParams':'',
},
```

Die SConsider Target Option „--run-force“ braucht es, damit SConsider unabhängig vom Testresultat durchläuft. Das Sconsolidator beginnt den ersten Testablauf. Die CUTE SConsider Datei ist sehr ähnlich zur COAST Testframework SConsider Datei.

Damit der Test Navigator die Testresultate anzeigt, ist in Eclipse die „Run Configurations…“ manuell einzustellen. Nach dem ersten Testablauf mit SConsider wurde im Ordner „tests“ ein Shell Skript Datei erstellt. Im Kontext ist eine Variable verschieden mit dem Dateinamen (fett markiert). Deshalb kann der Ablauf nicht vereinfacht werden. Einen Ausschnitt aus der Datei: CuteFoundationBaseTest.sh

```
LIBDIR="lib"
BINDIR="bin"
CONFIGDIR="config"
SCRIPTDIR="scripts"
VARIANTDIR="Linux_glibc_2.9-x86_64-32_optimized"
BINARYNAME="CuteFoundationBaseTest"
BASEDIR="`searchBaseDirUp \"${SCRIPTPATH}\" \"${SCRIPTDIR}\"`"
export LIBDIR BINDIR CONFIGDIR SCRIPTDIR VARIANTDIR BINARYNAME BASEDIR
```

1.7 Zusammenfassung und Ausblick

CUTE hat eine Testsuite und dazugehörige Tests. Die Testsuite ist eine Ablage für die Tests. SConsider erweitert SCons und gibt es als Eclipse CDT Plugin Sconsolidator. Das SConsider Target startet den Testablauf. CUTE SConsider braucht eine Klasse mit der main Funktion. Das Setup ist in CUTE als Header erstellt. Das CUTE hatte nur die Testsuite und Runner. CUTE hat einen Test Navigator für die Testresultate. Damit es diese Resultate darstellt, braucht es SConsider und dessen erstelltes Shell Skript. Eclipse CDT startet das Shell Skript. Die Resultate sind die letzten Testergebnisse.

Im Vergleich mit dem COAST Testframework ist CUTE für verschiedene Entwicklungsumgebungen entwickelt. Das machte es einfach das CUTE Testframework für ein anderes Framework anzupassen. Die Assertionen sind ein C++ Standard. Der Test Navigator könnte besser sein. Wenn das Shell Skript Fehler hätte, könnte es das JUnit kompatible XML parsen und die Resultate auflisten.

2 COAST

2.1 Ausgangslage

COAST ist die Abkürzung für C++ Open Application Server Toolkit und es ist ein Framework, um Webapplikationen in C++ zu entwickeln. Das COAST Testframework ist eine Entwicklung vom Institut für Software (IFS).

2.2 Aufgabenstellung

Die Aufgabe ist die vorhandenen COAST Testfälle mit CUTE zu migrieren. Die COAST Quelltexte bleiben unverändert. Das COAST Testframework bleibt bis CUTE vollständig übernimmt. Die Wartbarkeit verbessert sich mit CUTE. Die Quelltexte sind auf dem Git Server abgelegt.

2.2.1 Kann-Ziele

- Alle Testfälle migrieren.

2.2.2 Muss-Ziele

- Mit Linux arbeiten (ev. Virtuelle Maschine).
- Tests migrieren.
- Testfälle migrieren.
- Untersuchung einer Automatisierung repetitiver Aufgaben.
- Migrationsanleitung schreiben.

2.3 Rahmenbedingungen

- Abgabe der Kurzfassung an das Abteilungssekretariat ist am 12. Juni 2015.
- Abgabe des Berichts ist spätestens am 12. Juni 2015 um 12:00 Uhr.
- A0-Poster bis am 12. Juni 2015 um 10:00 Uhr.

2.3.1 Software Engineering

Es wird nach dem Rational Unified Process (RUP) gearbeitet. Die Iterationen:

- Business Modeling
- Requirements
- Analysis & Design
- Implementation
- Test
- Deployment

Die Phasen:

- Inception
- Elaboration
- Construction
- Transition

2.3.2 Technologien

- Cevelop 1.1.1 (Eclipse 4.4.1)
    - CUTE Headers 2.0.0 (CUTE 4.11)
    - SConsider
- GNU Toolchain (gcc/g++-4.9, gcc/g++-4.9-multilib)

- COAST (Revision c90af723)
- SCons 2.3.0
- Virtualenv 12.0

2.4 Vorgehen

1. Setup COAST in Eclipse unter Ubuntu.
2. Analyse des alten COAST Testframeworks und mit CUTE vergleichen.
3. Tests migrieren.
4. Migrationsanleitung schreiben.
5. Dokumentation.

2.5 Problembeschreibung

COAST ist ein C++ Framework, das Open Source ist und entstand im Institut für Software (IFS). Damit sind Webapplikationen in C++ entwickelt. COAST hat ein Testframework, das Unit Tests hat. SCons und SConsider erstellen das Testsystem. Einen COAST Testframework Test mit Assertionen ist wie folgt aufgebaut. Die Header Inhalte:

```
#ifndef _SS1Test_h_
#define _SS1Test_h_

#include "TestCase.h"

class SS1Test : public testframework::TestCase {
public:
        SS1Test(TString tstrName);
        virtual ~SS1Test();

        static Test *suite ();

        void SimpleTest();
};

#endif
```

Der Quelltext mit Implementation ist:

```
#include "SS1Test.h"
#include "StringStream.h"
#include "TestSuite.h"

SS1Test::SS1Test(TString tname) : TestCaseType(tname){}

SS1Test::~SS1Test() {}

Test *SS1Test::suite() {
        TestSuite *testSuite = new TestSuite;
        ADD_CASE(testSuite, SS1Test, SimpleTest);
        return testSuite;
}

void SS1Test::SimpleTest() {
        String out("something very small");
        assertEqual("something very small", out);
        assertEqual(21, out.Capacity());
        assertEqual(20, out.Length());
}
```

2.6 Lösungskonzept

Folgende Tabelle vergleicht die Testframeworks COAST und CUTE:

COAST Testframework	**CUTE**
Alle Tests sind in einer Testsuite	= (gleichwertig)
Eine Liste hat alle Tests	=
Setup führt alle Tests aus	- (nichts vergleichbares)
Die Testresultate sind in der Konsole dargestellt	=
-	Test Navigator („green/red bar“)
-	Setup um alle Tests zu sammeln
COAST Implementationen	**CUTE Implementationen**
TestSuite (mit Rückgabewert)	cute::suite
TestRunner	cute::makeRunner
SetupRunner	setupSuite
setUp	Initialisierungsliste/Konstruktor
tearDown	Destruktor

Das COAST Testframework hat eine Main Klasse. In dieser Klasse ruft es ein Setup auf, das alle Tests sammelt und in eine Testsuite/-liste kopiert. Danach werden die Tests ausgewertet. Das Testresultat ist in der Eclipse Konsole angezeigt. Jeder Test ist auf einer neuen Zeile. Das Testframework ist wenig verallgemeinert. Es ist nur für COAST geschrieben. Das CUTE ist auch ein Testframework. Es hat vergleichbare Member Funktionen wie COAST. Die meisten sind verschieden und in eine Assertion zusammengefasst. CUTE hat kein Setup. Die Tests sind einzeln in eine Testsuite/-liste hinzugefügt. Die Testauswertung ist in der Eclipse Konsole und als Test Navigator, das bei Erfolg einen grünen Balken oder bei mindestens einem Misserfolg einen roten Balken anzeigt.

Reguläre Ausdrücke (RegEx) vereinfachen und übernehmen, falls die Textsuche nichts ersetzen kann. In Eclipse ist es mit „Find/Replace“ und wenn die Option „Regular

expressions“ aktiviert ist. Um einen Test auf eine Zeile zu bringen, lösche zuerst die Zeilenumbrüche. Der reguläre Ausdruck „\R“ sucht die neue Zeile. Wenn die Assertion mehrere Werte hat und nur vertauscht sind, suche mit „(.*)“ den Kontext und verändere die Sortierung mit „\x“. Jeder Schritt wiederholen bis der Reguläre Ausdruck durch ist. Teilweise sind Assertionen zu komplex für RegEx. Eine Assertion, die mit RegEx versagt und eine Korrektur benötigt:

```
assertComparem( system::eExists, equal_to, system::MakeDirectory(strTmpDir, 0755,
false), "expected creation of directory to fail");
```

RegEx versagen:

```
ASSERT_EQUALM(0755, false), "expected creation of directory to fail", system::eExists,
system::MakeDirectory(strTmpDir);
```

Die richtige Lösung ist:

```
ASSERT_EQUALM("expected creation of directory to fail", system::eExists,
system::MakeDirectory(strTmpDir, 0755, false));
```

Die RegEx versagt auch bei diesem Beispiel:

```
Test *ClassName::suite() {
        StartTrace(ClassName.suite);
        TestSuite *testSuite = new TestSuite;
        return testSuite;
}
```

RegEx versagen:

```
Test *ClassName::suite() {
        TestSuite *testSuite = new TestSuite;
}
```

Die richtige Lösung ist:

```
Test *ClassName::suite() {
        StartTrace(ClassName.suite);
}
```

2.7 Umsetzung

- Setup von COAST in Cevelop unter Linux (z.B. in einer virtuellen Maschine)
- Analyse des alten COAST Testframeworks und Vergleich mit CUTE
- Migrationsanleitung, wie die Testfälle auf CUTE portiert werden können und welche Besonderheiten (z.B. durch Unterschiede in den Testframeworks verursacht) beachtet werden müssen.
- Migration von Tests und Dokumentation

2.8 Resultate: Bewertung und Zielerreichung

Im COAST Testframework sind Implementationen zu löschen. Die CUTE Implementationen sind dann zu verwenden. Die Migrationsanleitung zeigt das Vorgehen. Die Assertionen mit dem COAST Testframework (Reihenfolge beachten):

```
t_assert(extremelyLongString.Length()>0);
t_assertm(coast::typetraits::TypeTraits<TestType>::isConst == false, "expected non-const type");
assertEqual(trueString.length(), extremelyLongString.Length());
assertEqualm(false, retVal, NotNull(path));
ADD_CASE(testSuite, SystemFileTest, IStreamTest);
```

Dieselben Assertionen mit CUTE:

```
ASSERT(extremelyLongString.Length()>0);
ASSERTM("expected non-const type", coast::typetraits::TypeTraits<TestType>::isConst == false);
ASSERT_EQUAL(trueString.length(), extremelyLongString.Length());
ASSERT_EQUALM(NotNull(path), false, retVal);
s.push_back(CUTE_SMEMFUN(SystemFileTest, IStreamTest));
```

2.9 Zusammenfassung und Ausblick

In den Tests ist die Klasse TString mit std::string ersetzt. Es benutzt diese Klasse String für Fehlermeldungen. Im COAST Testframework hat es eine Member Funktion für den Funktionsnamen und ist mit dem Makro __FUNCTION__ ausgetauscht. Je nach dem ist die Member Funktion ganz löschbar. In manchen Fällen ist der Rückgabewert von der Member Funktion AsCharPtr() einen char *. Die CUTE Assertionen haben nur const char *. Eine Typumwandlung: char * zu const char *. Reguläre Ausdrücke können nicht alles. Wenn es versagt, ist die Korrektur unvollständig und eine Nachkorrektur notwendig.

Das Testframework CUTE hat bereits vielen Assertionen. Die COAST Klassen konnten damit nicht alle befriedigt werden. Mit Erweiterungen zu CUTE sind zusätzliche Assertionen implementiert. CUTE Assertionen haben keinen Rückgabewert, so dass es individuell gelöst sein musste. Das Verhalten überspringt den Test bei der Member Funktion, welche die Assertion fehlerhaft war. Das COAST Testframework markiert solche Assertionen als Fehler. Eine Überprüfung vor der Assertion löst das Manko. Die Reguläre Ausdrücke sind in Zwei Varianten geteilt. Beides zusammenfassen würde es vereinfachen die Übersicht zu behalten.

Kapitel II SW-Engineering Dokumentation

3 CUTE

3.1 Anforderungsanalyse

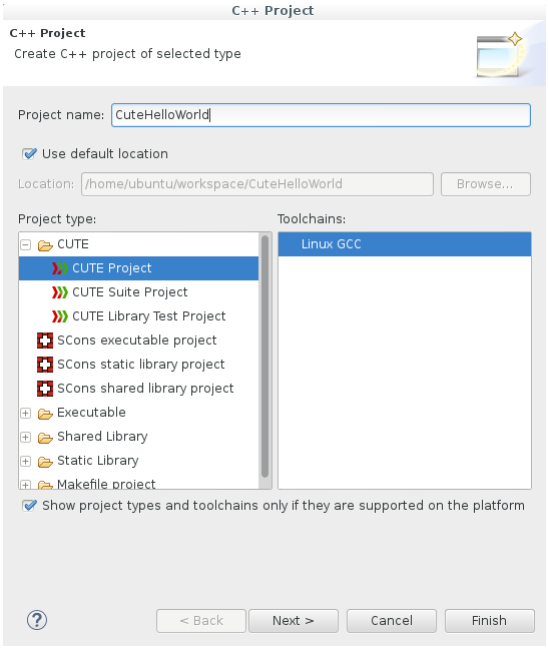
3.1.1 Allgemeine Beschreibung

3.1.1.1 Produkt Perspektive

Das CUTE vom Institut für Software (IFS) ist ein Testframework und steht für C++ Unit Testing Easier. Die Webpräsenz http://www.cute-test.com/ bietet ein Eclipse Plugin an, die per Eclipse Update-Site installiert. Das Plugin ist für Eclipse CDT. Die bestehenden C++ Projekte konvertiert CUTE zu einem CUTE Projekt. Auf der Webpräsenz ist eine Standalone. Neben die CUTE Header sind Tests, die CUTE testen und einige Beispiele.

3.1.1.2 Produkt Funktion

CUTE ist ein Testframework um C++ Quelltexte mit Unit Tests zu testen. Die Quelltexte kompilieren wie ein normales C++ Projekt. Die Konsolenausgabe und Test Navigator stellen die Testresultate dar. Der Test Navigator zeigt die Testsuite und Tests an und hat einen Balken für den aktuellen Status. Das CUTE braucht die C++ Bibliothek Boost.



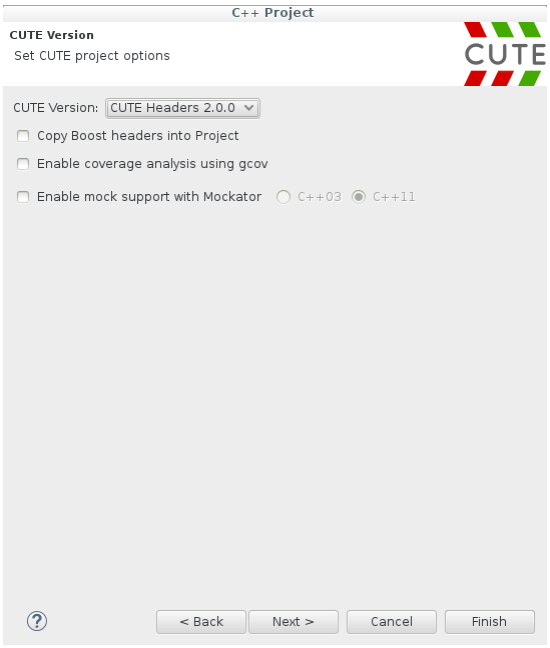


Abbildung 8 Eclipse CDT, CUTE Plugin: CUTE Project

Abbildung 9 Eclipse CDT, CUTE Plugin: CUTE Version

3.1.1.3 Benutzer Charakteristik

Die C++ Entwickler, welche Eclipse CDT als Entwicklungsplattform benutzen. Das Plugin bietet Unterstützung mit dem Test Navigator. Die Standalone ist für eigene Frameworks integrierbar.

3.1.1.4 Einschränkungen

CUTE Test Navigator gibt es nur im Plugin für Eclipse CDT.

3.1.1.5 Annahmen

- Quelltexte sind nur erweitert.

- Kompilieren mit GNU Toolchain.
- Linux Betriebssystem ist Voraussetzung.

3.1.1.6 Abhängigkeiten
Frameworks für die Entwicklung:

- Cevelop 4.4.1 (Eclipse 1.1.1)
- SCons 2.3.0

3.1.2 Weitere Anforderungen

3.1.2.1 Schnittstellen

3.1.2.1.1 SCons
Die Datei SConstruct liest SCons ein. Damit kontrolliert und kompiliert es das System.

3.1.2.1.2 SConsider
Eine SCons Erweiterung. Die Datei SConsider erzeugt das System.

3.1.2.1.3 Sconsolidator
Das SConsider Plugin für Eclipse CDT.

3.1.2.2 Randbedingungen
CUTE ist ein separates Testframework, das eigenständig läuft.

3.2 Domain Analyse

3.2.1 Domain Modell

3.2.1.1 Strukturdiagramm

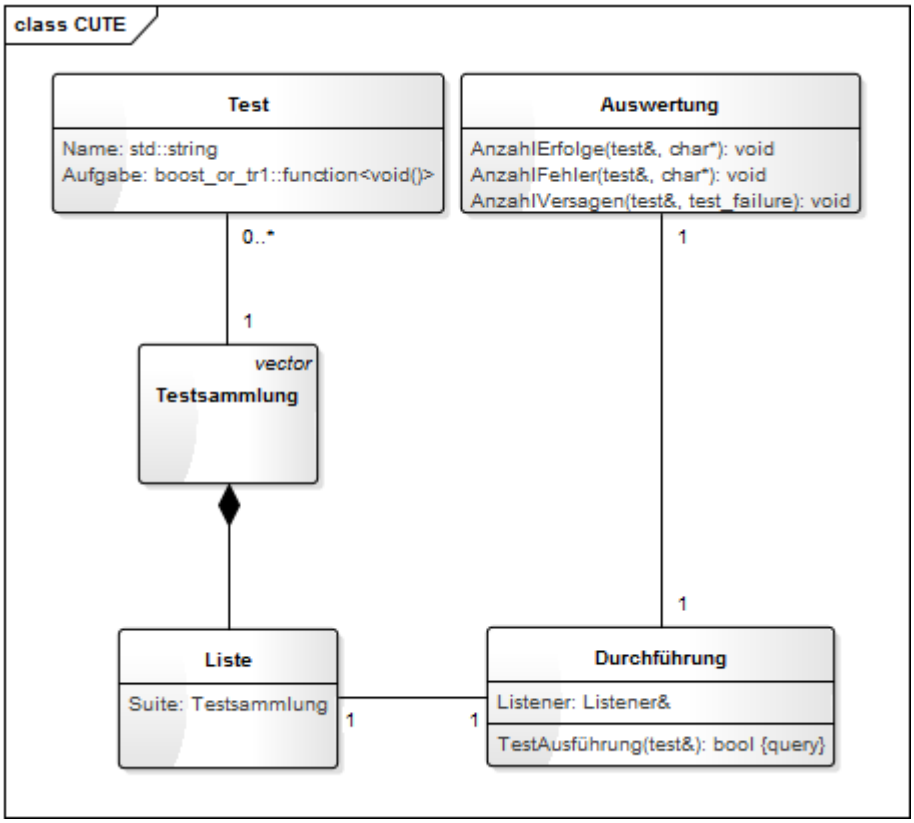


Abbildung 10

3.2.1.2 Wichtige Konzepte

3.2.1.2.1 Test
Der Test erhält einen Namen, Inhalte hinzufügen.

3.2.1.2.2 Testsammlung
Erweitert die Liste mit Operatoren.

3.2.1.2.3 Liste
Die Liste hat alle Tests als Testsammlung.

3.2.1.2.4 Durchführung
Die Durchführung mit einer Testausführung.

3.2.1.2.5 Auswertung
Die Statistiken vom Test anzeigen. Die Erfolg(e)/Misserfolg(e)/Fehler sind aufgezeichnet.

3.3 Software Architektur

3.3.1 Systemübersicht

Beschreibt die Softwarearchitektur eines Systems und wie sie sich präsentiert.

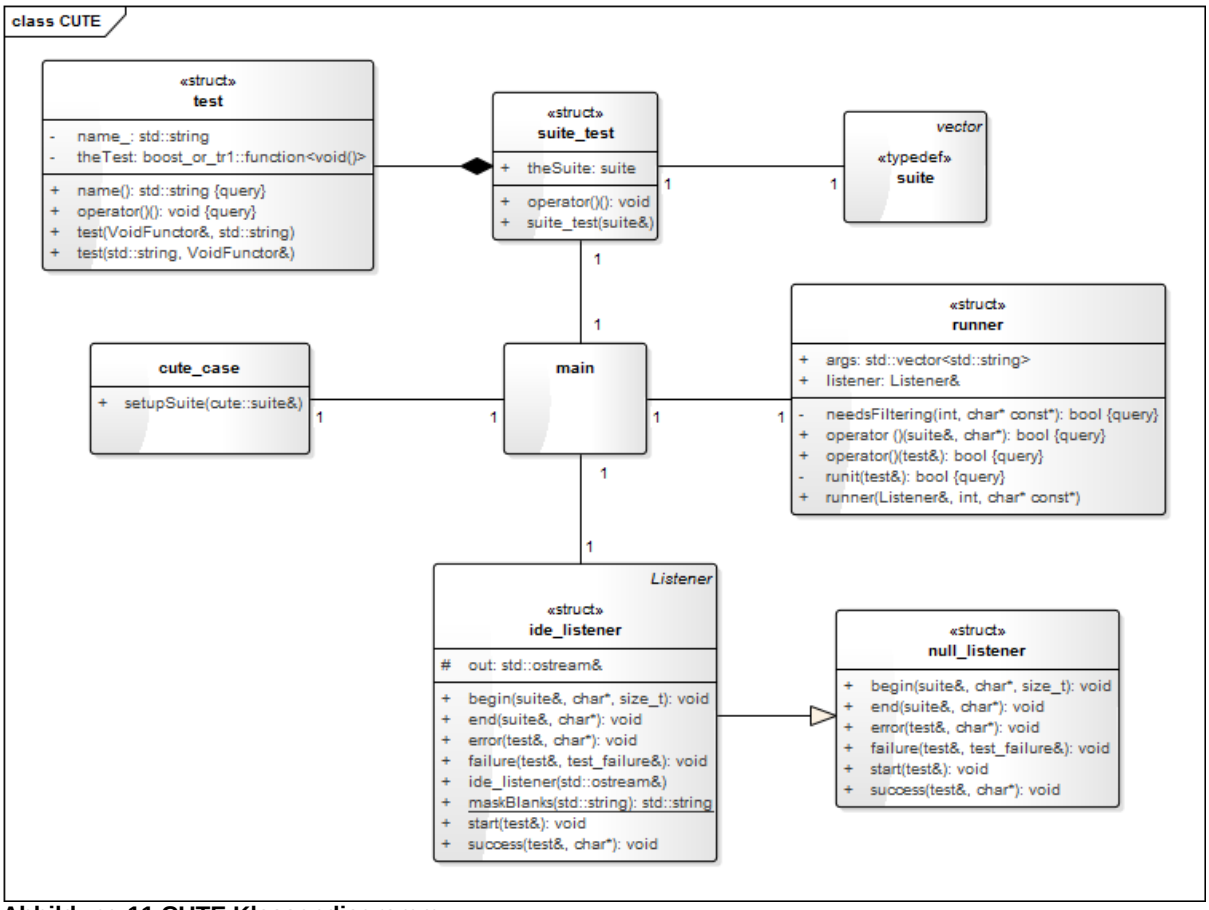


Abbildung 11 CUTE Klassendiagramm

3.3.2 Architektonische Ziele & Einschränkungen

3.3.2.1 Distribution

- CUTE Headers 2.0.0 (CUTE 4.11)

3.3.2.2 Implementationsstrategie

- Klasse mit einer main Funktion implementieren
- SConsider Datei hinzufügen
- Header Datei einsetzen

3.3.2.3 Design

- CUTE

3.3.2.4 Entwicklungstools

- Cevelop 4.4.1 (Eclipse 1.1.1)
- GNU Toolchain (gcc/g++-4.9, gcc/g++-4.9-multilib)
- SCons 2.3.0

3.3.3 Logische Architektur

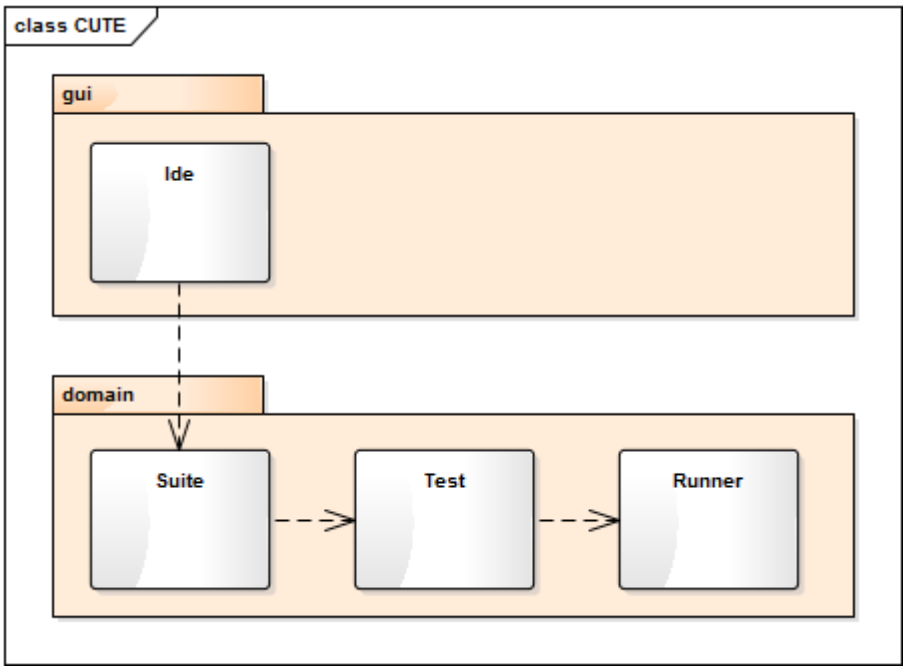


Abbildung 12

3.3.3.1 Gui/Ide

3.3.3.1.1 Klassenstruktur

Klasse	Beschreibung
ide_listener.h	Die Member Funktionen gibt das Resultat in der Konsole aus. Es ist eine Zeilendarstellung. Dasselbe Resultat zeigt es im Test Navigator in der Eclipse IDE.
ostream_listener.h	Ein Beispiel um einen eigenen Listener zu implementieren.
xml_listener.h	Die Member Funktionen gibt das Resultat als XML Darstellung aus. Dasselbe Resultat zeigt es im Test Navigator.
cute_listener.h	Ein Null Listener, die Deklarationen bereitstellt.

3.3.3.2 Domain/Suite

3.3.3.2.1 Klassenstruktur

Klasse	Beschreibung
cute_suite.h	Es hat einen Vektor. Die überladenen Operatoren sind nicht für das Eclipse Plugin gedacht.
cute_suite_test.h	Macht aus der Suite einen Test.

3.3.3.2.2 Schnittstellen

- Domain/Test

3.3.3.3 Domain/Runner

3.3.3.3.1 Klassenstruktur

Klasse	Beschreibung
cute_runner.h	Durchläuft die Suite.

3.3.3.3.2 Schnittstellen

- Gui/Ide
- Domain/Suite
- Domain/Test

3.3.3.4 Domain/Test

3.3.3.4.1 Klassenstruktur

Klasse	Beschreibung
cute_test.h	Gibt dem Test ein Name.

3.3.3.5 Wichtige Abläufe

Dieses CUTE hat eine Klasse mit einer main Funktion, die beginnt mit der Suite. Die Klasse cute_case.h ist implementiert, dann sind alle Tests bekannt.

3.3.4 Deployment

Die CUTE Header, Quelltexte und Tests.

3.4 Implementation

3.4.1 Die Konfigurationsdatei: cute.sconsider

```
import pkg_resources
pkg_resources.require(["SConsider"])
import SConsider

Import('*')

_sconsider_dist = pkg_resources.get_distribution('SConsider').parsed_version
if _sconsider_dist < pkg_resources.parse_version("0.5.dev"):
    buildSettings = {
        packagename: {
            'sourceFiles': SConsider.listFiles(['*.cpp']),
            'targetType': 'LibraryShared',
            'public': {
                'includes': SConsider.listFiles(['*.h']),
            },
        },
    }
    SConsider.createTargets(packagename, buildSettings)
```

Package definition[1]	Beschreibung
import pkg_resources import SConsider	Das import System lädt die Module pkg_resources und SConsider
Import('*')	Exportiert Variablen, die importiert sind

[1] https://redmine.coast-project.org/projects/sconsider/wiki (27.5.15)

packagename	Die main Target
sourceFiles	Liste von Quelltexte
targetType	Eine shared library aufbauen
includes	Public Headers

3.4.2 Beschreibung von Klassen und Member Funktionen

```
#include "ide_listener.h"
#include "xml_listener.h"
#include "cute_counting_listener.h"
#include "cute_runner.h"
#include "cute.h"
#include "cute_case.h"
#include <iostream>

const char * setupSuite(cute::suite &s) {
	return "AllTests";
}

int main(int argc, char const *argv[]) {
	cute::suite s;
	const char *test = setupSuite(s);
	cute::xml_file_opener xmlfile(argc, argv);
	cute::xml_listener<cute::counting_listener<cute::ide_listener<> > > lis(xmlfile.out);
	cute::makeRunner(lis)(s, test);
	std::cerr << lis.numberOfTests << " Tests - expect " << s.size() << std::endl;
	std::cerr << lis.successfulTests << " successful - expect " << s.size() << std::endl;
	std::cerr << lis.failedTests << " failed - expect 0 failures" << std::endl;
	std::cerr << lis.errors << " errors - expect 0 errors" << std::endl;
	return lis.failedTests;
}
```

Die Klasse cute_main hat mehrere CUTE Headers. Es hat eine main Member Funktion. Die Member Funktion setupSuite(cute::suite &s) hat nur ein Rückgabewert. Die Implementation von cute_case.cpp überschreibt diese Member Funktion.

```
#ifndef CUTE_CASE_H_
#define CUTE_CASE_H_

#include "cute.h"

void setupSuite(cute::suite &s);

#endif /* CUTE_CASE_H_ */
```

Die Header hat nur eine Deklaration setupSuite(cute::suite &s). Die Implementation ist zu gestalten mit den Tests Header Dateien und die Member Funktion runAllTests(cute::suite &s).

3.5 Test

CUTE hat eigene Tests, die CUTE testen. Den Ordner cute in Eclipse importieren. Dieses SCons Target erstellen:

- cutetest

Die SCons Option „--run-force" hinzufügen. Die Tests starten mit „Build target". Die Auswertung ist in der Eclipse Konsole.

3.5.1 Die Konfigurationsdatei: cutetest.sconsider

```
import pkg_resources
pkg_resources.require(["SConsider"])
import SConsider

Import('*')

_sconsider_dist = pkg_resources.get_distribution('SConsider').parsed_version
if _sconsider_dist < pkg_resources.parse_version("0.5.dev"):
    buildSettings = {
        packagename: {
            'linkDependencies': ['cute'],
            'sourceFiles': SConsider.listFiles(['*.cpp']),
            'targetType': 'ProgramTest',
            'public': {
                'includes': SConsider.listFiles(['*.h']),
            },
            'runConfig': {
                'runParams':'',
            },
        },
    }
    SConsider.createTargets(packagename, buildSettings)
```

3.5.2 Beschreibung von Klassen und Member Funktionen

```
#include "cute.h"
#include <iostream>

#include "cute_case.h"
#include "test_any.h"
#include "test_cute_equals.h"
#include "test_cute_expect.h"
#include "test_repeated_test.h"
#include "test_cute_runner.h"
#include "test_cute_suite_test.h"
#include "test_cute_suite.h"
#include "test_cute_test_incarnate.h"
#include "test_cute_test.h"
#include "test_cute_testmember.h"
#include "test_cute.h"
#include "test_cute_to_string.h"
#include "test_cute_to_string_embedded.h"
#include "test_xml_listener.h"
#include "test_xml_file_opener.h"
#include "test_cute_filter_runner.h"
#include "test_cute_relops.h"
#include "test_cute_data_driven.h"
```

Die Tests Header.

```
using namespace cute;
// some brain dead test cases to find out my bug using function
static int simpleTestfunctionCalled=0;
void simpleTestFunction(){
	++simpleTestfunctionCalled;
	std::cerr << "simpleTestFunction run no:"<< simpleTestfunctionCalled << std::endl;
	ASSERT(true);
	throw std::exception();
}
struct SimpleTestFunctionCalledTest {
	void operator()() const{
		ASSERT_EQUALM("look at cute::test ctor
overload",1,simpleTestfunctionCalled);
	}
};
void shouldFailButNotThrowStdException(){
	ASSERT(false);
	throw std::exception();
}
// demonstrates how to write equality tests
void test2(){
	ASSERT_EQUAL(1,1);
	ASSERT_EQUAL(1,2);
}
// demonstrates how to write test functors
struct test3{
	void operator()() const{
		throw std::exception();
	}
};
// demonstrates how test objects are incarnated
struct to_incarnate{
	std::ostream &out;
	to_incarnate(std::ostream &os):out(os){
		out << "born" << std::endl;
	}
	to_incarnate(to_incarnate &other):out(other.out){} // copy ctor would be deleted and
is required by std::function ctor
	~to_incarnate() {
		out << "killed" << std::endl;
	}
	void operator()(){
		out << "tested" << std::endl;
	}
};
struct to_incarnate_without : to_incarnate {
	to_incarnate_without():to_incarnate(std::cout){}
};
void test_SimpleTestFunctionThrows(){
	ASSERT_THROWS(simpleTestFunction(),std::exception);
}
void test_shouldFailThrowsFailure(){
	ASSERT_THROWS(shouldFailButNotThrowStdException(),cute::test_failure);
}
```

```
const char * setupSuite(cute::suite &s) {
	using namespace std;
	TestAny::runAllTests(s);
	s += make_suite_test_cute_data_driven();
	s += make_suite_test_cute_relops();
	s += make_suite_test_cute_filter_runner();
	s += make_suite_test_xml_listener();
	s += make_suite_test_xml_file_opener();
	s += make_suite_test_cute_to_string_embedded();
	s += test_cute_to_string();
	s += test_cute_equals();
	// the following test produces one of the 2 expected errors, since it throws
	//s += CUTE(simpleTestFunction);
	s += CUTE(test_SimpleTestFunctionThrows);
	s += SimpleTestFunctionCalledTest();
	s += CUTE(test_shouldFailThrowsFailure);
	suite throwing=test_cute_expect();
	s.insert(s.end(),throwing.begin(),throwing.end());
	s += CUTE_SUITE_TEST(test_repeated_test());
	s += CUTE(test_cute_runner);
	s += CUTE_SUITE_TEST(test_cute_suite_test());
	s += CUTE(test_cute_suite);
	s += CUTE(test_lambda_suite);
	s += CUTE_SUITE_TEST(test_cute_test_incarnate());
	s += CUTE_SUITE_TEST(test_cute_test());
	s += CUTE_SUITE_TEST(test_cute_testmember());
	s += CUTE_SUITE_TEST(test_cute());
	//---
	s += CUTE_INCARNATE(to_incarnate_without);
	s +=
CUTE_INCARNATE_WITH_CONTEXT(to_incarnate,boost_or_tr1::ref(std::cout));
	s +=
CUTE_CONTEXT_MEMFUN(boost_or_tr1::ref(std::cerr),to_incarnate,operator());
	return "cutetest";
}
```

Die Klasse ist refaktorisiert für den Ablauf mit SConsider. Die CUTE Tests entsprechen der Vorgabe und ergänzt mit einem Anythingtest. Damit testet CUTE alle Tests mit SConsider.

```
#ifndef TEST_ANY_H_
#define TEST_ANY_H_

#include "cute_case.h"
#include "AssertionAnything.h"

struct TestAny : public Assertion {
	void dummyTest();
	void anyTest();
	static void runAllTests(cute::suite &s);
};


#endif /* TEST_ANY_H_ */
```

Die Klasse Anythingtest mit Deklarationen.

```
#include "test_any.h"

void TestAny::dummyTest() {
	ASSERT(1);
	ASSERT_EQUAL(1, 1);
	ASSERT_EQUAL_DELTA(1.2, 1.2000000001, 1E-5);
	ASSERT_EQUAL("Test", "Test");
	ASSERT_EQUAL(0, false);
	ASSERT_EQUAL(1, true);
}

void TestAny::anyTest() {
	Anything a, b;
	ASSERT_ANY_EQUAL(a, b);
	ASSERT_ANY_EQUALM("ANY", a, b);
	a.Append("Test");
	b[0L] = "Test";
	ASSERT_ANY_EQUAL(a, b);
	ASSERT_ANY_EQUALM("ANY Test", a, b);
}

void TestAny::runAllTests(cute::suite &s) {
	s.push_back(CUTE_SMEMFUN(TestAny, dummyTest));
	s.push_back(CUTE_SMEMFUN(TestAny, anyTest));
}
```

Die Member Funktion dummyTest() und anyTest() testet CUTE (Extensions) Assertionen. Der Test ist mit der Member Funktion runAllTests(cute::suite &s) in die Testsuite hinzugefügt.

3.6 Schlussbericht

3.6.1 Zielerreichung

Die CUTE Header Dateien sind unverändert. CUTE ist mit SConsider erweitert, damit ist es mit dem COAST Testframework fast identisch. Die Konfigurationsdatei ist erstellt und kompiliert CUTE als Bibliothek. Neu ist die Klasse cute_main mit der main Member Funktion zum Start von CUTE. Ebenso ist neu die Header cute_case. Diese Deklaration muss implementiert werden. Die Tests haben eine statische Member Funktion, welche ins Setup muss. Nebenbei erzeugt das System eine XML Datei, die zum JUnit XML Format kompatibel ist. Diese Datei ist für die Build Automation gedacht, zum Beispiel Jenkins. Nach dem kompilieren macht Jenkins eine Auswertung und stellt es im WUI dar. In der Konsole zeigt es das Ergebnis als Zusammenfassung an.

4 COAST

4.1 Anforderungsanalyse

4.1.1 Allgemeine Beschreibung

4.1.1.1 Produkt Perspektive

Das Framework ist COAST (C++ Open Application Server Toolkit). Auf der Webpräsenz http://coast-project.org/ ist nichts Stabiles veröffentlicht, aber COAST gilt schon als stabil. Es sind nur noch ausstehende Fehler zu beseitigen. Die Idee ist eine eigene Webapplikation Framework. Das COAST Testframework entwickelt das Institut für Software (IFS). Es ist nur für COAST.

4.1.1.2 Produkt Funktion

COAST dient als Grundlage um Webapplikationen mit C++ zu entwickeln. Das COAST Testframework testet mit über 10000 Unit Tests das COAST. Die Tests starten mit dem Build System SCons und SConsider, zum Beispiel als Eclipse CDT Plugin. Die Resultate stellt es in der Konsole dar. Es kann ein XML exportieren, welches mit dem JUnit XML Format kompatibel ist. Jenkins parst die XML nach dem kompilieren und zeigt es im WUI.

4.1.1.3 Benutzer Charakteristik

Die Benutzer, welche C++ gelernt haben und Webapplikationen programmieren.

4.1.1.4 Einschränkungen

Das COAST Testframework ist eine eigene Entwicklung vom Institut für Software (IFS).

4.1.1.5 Annahmen

- COAST Quelltexte sind nicht geändert.
- COAST Testframework weglassen.
- Nur CUTE benutzen.
- Kompilieren mit GNU Toolchain.
- Linux Betriebssystem ist Voraussetzung.

4.1.1.6 Abhängigkeiten

Frameworks für die Entwicklung:

- Cevelop 4.4.1 (Eclipse 1.1.1)
- GNU Toolchain (gcc/g++-4.9, gcc/g++-4.9-multilib)
- SCons 2.3.0
- Virtualenv 12.0

4.1.2 Weitere Anforderungen

4.1.2.1 Randbedingungen

Das COAST läuft eigenständig und ist in C++ geschrieben. Das Testframework ist nur für COAST zu geschnitten. Es sind bereits Tests in C++ geschrieben. CUTE soll das COAST Testframework übernehmen. Die COAST Testframework Testfälle sind für CUTE zu migrieren. Die Tests sind in separate Verzeichnisse, beginnend mit „cute“, kopiert.

4.2 Domain Analyse

4.2.1 Domain Modell

4.2.1.1 Strukturdiagramm

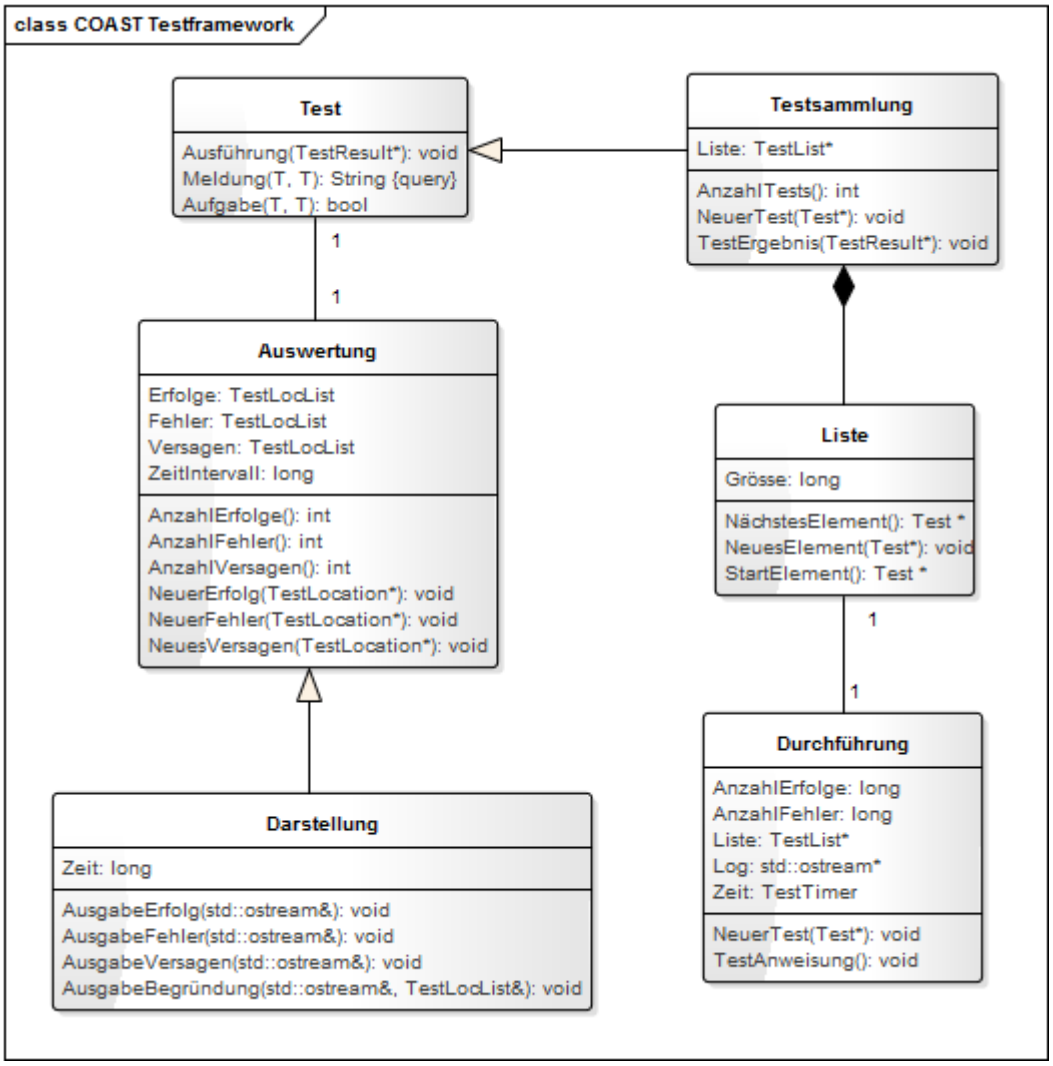


Abbildung 13

4.2.1.2 Wichtige Konzepte

4.2.1.2.1 Test

Es hat eine Ausführung, eine Meldung, einen Inhalt.

4.2.1.2.2 Auswertung

Die Tests auswerten. Die Erfolg(e)/Misserfolg(e)/Fehler/Zeit sind als Statistik aufgelistet.

4.2.1.2.3 Testsammlung

Eine Sammlung für Tests und nur hinzufügen ist gestattet.

4.2.1.2.4 Liste

Eine Liste, die alle Tests hat.

4.2.1.2.5 Durchführung

Die Tests starten damit. Die Erfolg(e)/Misserfolg(e)/Fehler sind gezählt.

4.2.1.2.6 Darstellung

Die Ausgabe zu den Tests. Die Erfolg(e)/Misserfolg(e)/Fehler und die Zeit sind aufgeteilt dargestellt.

4.3 Software Architektur

4.3.1 Systemübersicht

Beschreibt die Softwarearchitektur eines Systems und wie sie sich präsentiert.

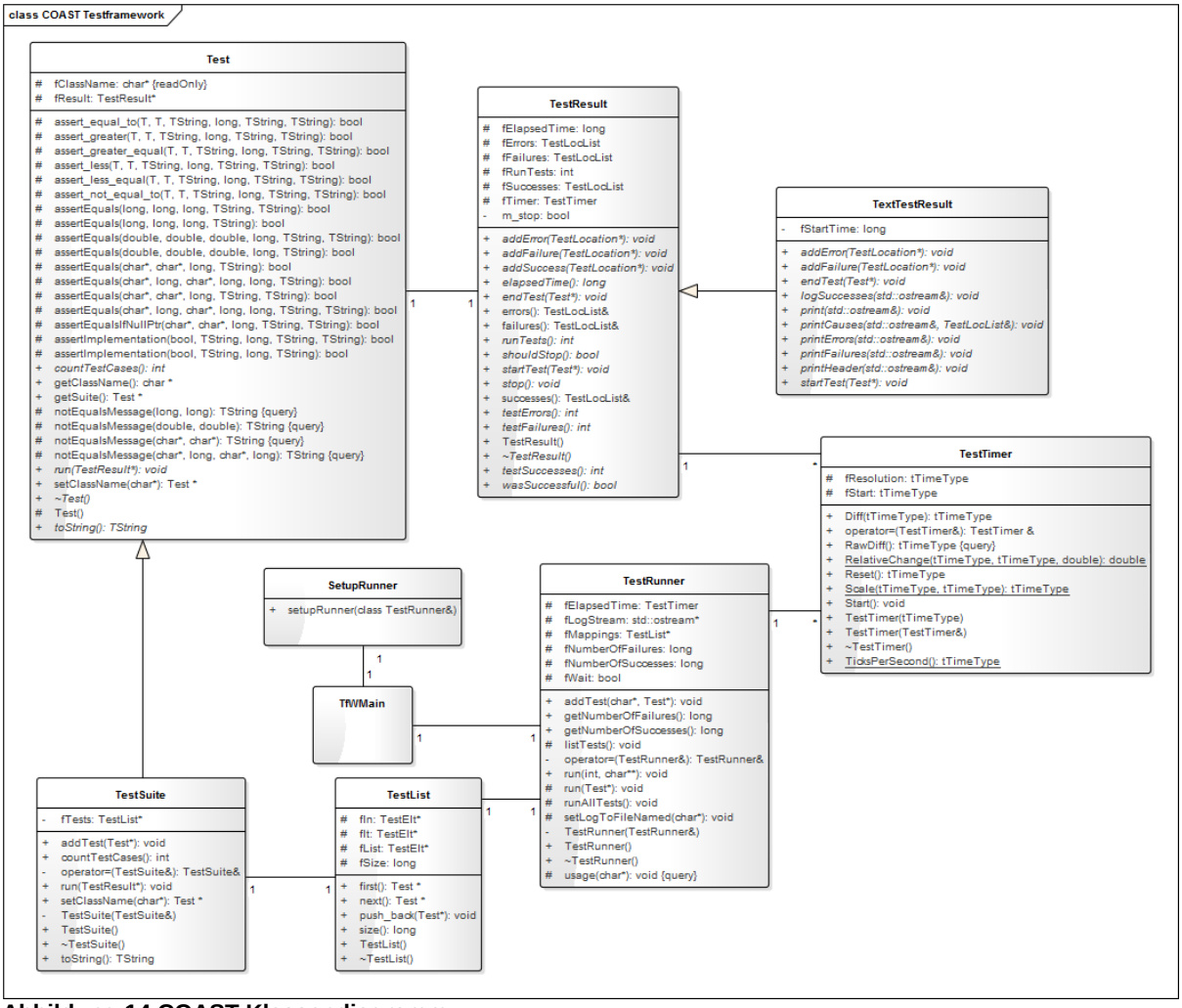


Abbildung 14 COAST Klassendiagramm

4.3.2 Architektonische Ziele & Einschränkungen

4.3.2.1 Distribution

- COAST (Revision c90af723)
- CUTE Headers 2.0.0 (CUTE 4.11)

4.3.2.2 Implementationsstrategie

- CUTE hinzufügen
- Tests migrieren
- Neue CUTE Testfälle implementieren

- Testfälle migrieren

4.3.2.3 Design

- COAST Testframework
- CUTE

4.3.2.4 Entwicklungstools

- Cevelop 4.4.1 (Eclipse 1.1.1)
  - CUTE Headers 2.0.0 (CUTE 4.11)
  - SConsider
- GNU Toolchain (gcc/g++-4.9, gcc/g++-4.9-multilib)
- SCons 2.3.0
- Virtualenv 12.0

4.3.3 Logische Architektur

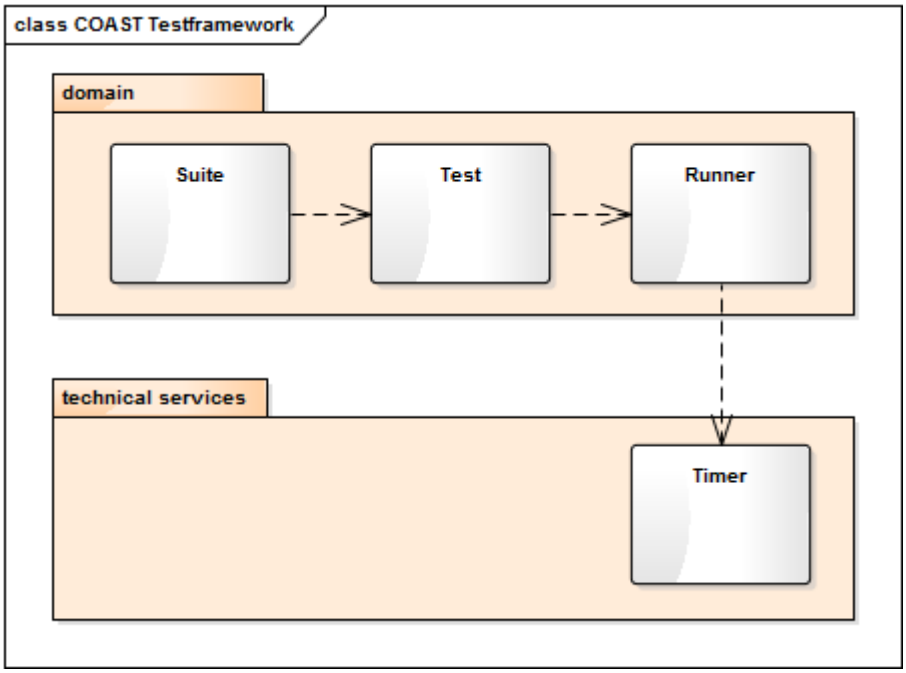


Abbildung 15

4.3.3.1 Domain/Suite

4.3.3.1.1 Klassenstruktur

Klasse	Beschreibung
TestSuite.cpp	Diese Klasse startet die Tests. Es erbt von der Klasse Test. Es hat eine Member Funktion, die alle Tests zählt.
TestSuite.h	
TestList.cpp	Die Testliste hat alle Tests. Es hat einen Iterator.
TestList.h	

4.3.3.1.2 Schnittstellen

- Domain/Test

4.3.3.2 Domain/Test

4.3.3.2.1 Klassenstruktur

Klasse	Beschreibung
Test.cpp	Die Tests haben Assertionen, ein Name, einen Zähler, welche die Anzahl Tests hat.
Test.h	

4.3.3.3 Domain/Runner

4.3.3.3.1 Klassenstruktur

Klasse	Beschreibung
TestRunner.cpp	Durchläuft die Testliste. Zählt die Fehler, Erfolge. Fügt Tests hinzu. Schreibt eine Logdatei.
TestRunner.h	

4.3.3.3.2 Schnittstellen

- Technical Services/Timer

4.3.3.4 Technical Services/Timer

4.3.3.4.1 Klassenstruktur

Klasse	Beschreibung
TestTimer.cpp	Es misst die Zeit zwischen den Tests in Sekunden. Die Betriebssysteme haben unterschiedliche Ticks.
TestTimer.h	

4.3.3.5 Wichtige Abläufe

Das COAST Testframework hat eine Klasse mit einer main Methode, die beginnt mit dem Runner und dem Setup. Die Klasse SetupRunner.h ist implementiert, dann sind alle Tests bekannt.

4.3.4 Deployment

Die COAST Quelltexte sind unverändert. Dazu kommen migrierte Tests/Testfälle. Im Ordner „cute“ sind die CUTE Header Dateien. In den Ordner „coast/foundation/base/cuteTest“, „coast/foundation/base/anythingoptional/cuteTest“ und „coast/foundation/base/io/cuteTest“ sind die migrierten CUTE Tests. Im Ordner „cuteExtensions“ sind die CUTE Erweiterungen.

4.4 Implementation

4.4.1 Die Konfigurationsdatei: CuteFoundationBaseTest.sconsider

```
import pkg_resources
pkg_resources.require(["SConsider"])
import os
import tempfile
import SConsider
import SCons
from stat import *

Import('*')

searchReplace = {
   'DIREXTENDBASEDIR': lambda env: os.path.join(
      env.getTargetBaseInstallDir().abspath,
      'makedirextendtest',
      'test'),
   'TMPDIR': lambda env: tempfile.gettempdir() if str(
      env['PLATFORM']) in [
      'cygwin',
      'win32'] else '/tmp',
   'ROOTFS': lambda env: 'c:/' if str(
      env['PLATFORM']) in [
      'cygwin',
      'win32'] else '/',
   'FILESIZE_len5.tst': SCons.Script.File('config/len5.tst').get_size(),
   'FILESIZE_AnythingTest.any': SCons.Script.File('config/AnythingTest.any').get_size(),
}


def SystemTestMakeDirectoryExtendCleanup(env):
   tdirabs = env.getTargetBaseInstallDir().abspath
   basepath = os.path.join(tdirabs, 'makedirextendtest')
   if os.path.isdir(basepath):
      shutil.rmtree(basepath, ignore_errors=True)


def setUp(target, source, env):
   env['ENV']['COAST_DOLOG'] = '4'
   env['ENV']['COAST_LOGONCERR'] = '4'
   env['ENV']['COAST_TRACE_STORAGE'] = '1'
   env['ENV']['COAST_TRACE_INITFINIS'] = '1'
   pass
```

Der Inhalt dieser Datei ist sehr ähnlich zu der Datei im COAST Testframework CoastFoundationBaseTest.sconsider.

```
def tearDown(target, source, env):
    # delete generated files
    SConsider.removeFiles(
        SConsider.findFiles(
            [env.getTargetBaseInstallDir()],
            extensions=['.res'],
            matchfiles=['include.any']))
    SystemTestMakeDirectoryExtendCleanup(env)

_sconsider_dist = pkg_resources.get_distribution("SConsider").parsed_version
if _sconsider_dist < pkg_resources.parse_version("0.5.dev"):
    buildSettings = {
        packagename: {
            'linkDependencies': [
                'CoastFoundation.Base',
                'CoastFoundation.Time',
                'CoastFoundation.Miscellaneous',
                'CuteExtensions',],
            'sourceFiles': SConsider.listFiles(
                ['*.cpp']),
            'targetType': 'ProgramTest',
            'copyFiles': [
                (SConsider.listFiles(
                    ['config/*.any']),
                 S_IRUSR | S_IRGRP | S_IROTH,
                 searchReplace),
                (SConsider.listFiles(
                    [
                        'config/*.tst',
                        'config/*.txt',
                        'tmp/*']),
                 S_IRUSR | S_IRGRP | S_IROTH),
            ],
            'runConfig': {
                'setUp': setUp,
                'tearDown': tearDown,
                'runParams':'',
            },
        },
    }

    SConsider.createTargets(packagename, buildSettings)
```

Da sind zwei Ausnahmen: Unter tearDown und extensions ist der Eintrag „.tst“ gelöscht. Die Konfigurationsdateien braucht nachher das Shell Skript. Beim Eintrag „‘runConfig‘:“ braucht es zusätzlich den Parameter „‘runParams‘:‘“. Das Argument ist provisorisch als SConsider Option.

4.4.2 Die Konfigurationsdatei: CuteExtensions.sconsider

```
import pkg_resources
pkg_resources.require(["SConsider"])
import SConsider

Import('*')

_sconsider_dist = pkg_resources.get_distribution("SConsider").parsed_version
if _sconsider_dist < pkg_resources.parse_version("0.5.dev"):
    buildSettings = {
        packagename: {
            'linkDependencies': ['cute', 'CoastFoundation.AnythingOptional'],
            'sourceFiles': SConsider.listFiles(['*.cpp']),
            'targetType': 'LibraryShared',
            'public': {
                'includes': SConsider.listFiles(['*.h']),
            }
        }
    }

    SConsider.createTargets(packagename, buildSettings)
```

4.4.3 Die Konfigurationsdatei: CuteFoundationAnythingOptionalTest.sconsider

```
import pkg_resources
pkg_resources.require(["SConsider"])
import SConsider
from stat import *

Import('*')

_sconsider_dist = pkg_resources.get_distribution("SConsider").parsed_version
if _sconsider_dist < pkg_resources.parse_version("0.5.dev"):
    buildSettings = {
        packagename: {
            'linkDependencies': [
                'CuteExtensions'],
            'sourceFiles': SConsider.listFiles(
                ['*.cpp']),
            'targetType': 'ProgramTest',
            'copyFiles': [
                (SConsider.listFiles(
                    ['config/*.any']),
                 S_IRUSR | S_IRGRP | S_IROTH),
            ],
            'runConfig': {
                'runParams':'',
            },
        },
    }

    SConsider.createTargets(packagename, buildSettings)
```

Package definition[2]	Beschreibung
import pkg_resources import SConsider	Das import System lädt die Module pkg_resources und SConsider
Import('*')	Exportiert Variablen, die importiert sind
packagename	Die main Target
linkDependencies	Link zu den shared libraries
sourceFiles	Liste von Quelltexte
targetType	Eine shared library aufbauen
includes	Public Headers

4.4.4 Beschreibungen von Klassen und Member Funktionen

```
#include "cute_case.h"
#include "AnythingConstructorsTest.h"
#include "AnythingDeepCloneTest.h"
#include "AnythingImportExportTest.h"
#include "AnythingLookupTest.h"
#include "AnythingKeyIndexTest.h"
#include "AnythingTest.h"
#include "StringTest.h"
#include "StringTestExtreme.h"
#include "StringTokenizerTest.h"
#include "StringTokenizer2Test.h"
#include "AnythingParserSemanticTest.h"
#include "AnythingParserTest.h"
#include "StrSpecialTest.h"
#include "SysLogTest.h"
#include "TracerTest.h"
#include "SystemBaseTest.h"
#include "SystemFileTest.h"
#include "ROSimpleAnythingTest.h"
#include "AnyBuiltInSortTest.h"
#include "AnyImplsTest.h"
#include "TypeTraitsTest.h"
#include "AnythingIteratorTest.h"
#include "AnythingSTLTest.h"
#if !defined(WIN32)
#include "MmapTest.h"
#endif
#include "StringStreamTest.h"
#include "StringIteratorTest.h"
#include "StringReverseIteratorTest.h"
#include "StringSTLTest.h"
#include "AnythingImportExportTest.h"
#include "AnythingKeyIndexTest.h"
```

In dieser Implementation sind die Header Dateien aufgelistet.

[2] https://redmine.coast-project.org/projects/sconsider/wiki (27.5.15)

```
void setupSuite(cute::suite &s) {
	ROSimpleAnythingTest::runAllTests(s);
	StringIteratorTest::runAllTests(s);
	StringTestExtreme::runAllTests(s);
	StringTokenizerTest::runAllTests(s);
	StrSpecialTest::runAllTests(s);
	SysLogTest::runAllTests(s);
	TracerTest::runAllTests(s);
	TypeTraitsTest::runAllTests(s);
	AnythingIteratorTest::runAllTests(s);
	StringReverseIteratorTest::runAllTests(s);
	AnyImplsTest::runAllTests(s);
	AnythingLookupTest::runAllTests(s);
	StringSTLTest::runAllTests(s);
	StringTokenizer2Test::runAllTests(s);
	SystemBaseTest::runAllTests(s);
	AnythingDeepCloneTest::runAllTests(s);
	AnythingConstructorsTest::runAllTests(s);
	AnythingParserSemanticTest::runAllTests(s);
	StringTest::runAllTests(s);
#if !defined(WIN32)
	MmapTest::runAllTests(s);
#endif
	AnyBuiltInSortTest::runAllTests(s);
	AnythingTest::runAllTests(s);
	AnythingSTLTest::runAllTests(s);
	StringStreamTest::runAllTests(s);
	AnythingImportExportTest::runAllTests(s);
	AnythingKeyIndexTest::runAllTests(s);
	AnythingParserTest::runAllTests(s);
	SystemFileTest::runAllTests(s);
}
```

Die Testsuites sind in der Member Funktion. Die CUTE Suite übergibt es jedem Test als Referenz. Die Tests haben die statische Member Funktion runAllTests(cute::suite &s) implementiert.

```
#ifndef _AssertionAnything_H
#define _AssertionAnything_H

#include "cute.h"
#include "StringStream.h"
#include "AnyIterators.h"

struct Assertion {
	void ASSERT_ANY_EQUAL(const ROAnything &expected, const ROAnything
&actual);
	void ASSERT_ANY_EQUALM(std::string msg, const ROAnything &expected, const
ROAnything &actual);
	void ASSERT_ANY_COMPARE(const ROAnything &inputAny, const ROAnything
&masterAny, String pathSoFar, char delimSlot = '.', char delimIdx = ':');
private:
	String sexp, act;
};

#endif
```

Die Header Datei hat Deklarationen für Assertionen um die Klasse Anything.

```
#include "AssertionAnything.h"

void Assertion::ASSERT_ANY_EQUAL(const ROAnything &expected, const ROAnything
&actual) {
	OStringStream oexp(&sexp), oact(&act);
	expected.Export(oexp, false);
	actual.Export(oact, false);
	ASSERT_EQUAL(sexp, act);
}

void Assertion::ASSERT_ANY_EQUALM(std::string msg, const ROAnything &expected,
const ROAnything &actual) {
	OStringStream oexp(&sexp), oact(&act);
	expected.Export(oexp, false);
	actual.Export(oact, false);
	ASSERT_EQUALM(msg, sexp, act);
}

void Assertion::ASSERT_ANY_COMPARE(const ROAnything &master, const ROAnything
&actual, String location, char delimSlot, char idxdelim) {
	OStringStream s;
	String failingPath(location);
	if(!AnyUtils::AnyCompareEqual(actual, master, failingPath,&s, delimSlot, idxdelim)) {
		String strfail(failingPath);
		strfail << "\n" << s.str();
		ASSERTM((const char*)strfail, false);
	}
}
```

Die Klasse Assertion hat verschiedene Member Funktionen. Diese testen die Member Funktionen in der Klasse Anything. Die Member Funktion ist derselben COAST Testframework Member Funktion ähnlich.

```
void setQuery(String name = "");
```
Die Deklaration hat einen String, der leer bleibt, wenn es ohne Argument ist.

```
void AnythingTest::setQuery(String name) {
	if (!system::LoadConfigFile(fConfig, "AnythingTest", "any")) {
		ASSERT_EQUAL( "'read AnythingTest.any'", "'could not read
AnythingTest.any'" );
	}
	fQuery = fConfig["Queries"][name];
}
```
Die Implementation lädt eine Query aus der Konfigurationsdatei AnythingTest.any. Wenn die Konfiguration nicht lesbar ist, dann ist die Assertion falsch.

```
ROAnything GetTestCaseConfig(String strClassName = "", String strTestName = "");
```
Die Deklaration haben Strings, die leer bleiben, wenn es ohne Argument ist.

```
ROAnything SystemFileTest::GetTestCaseConfig(String strClassName, String strTestName)
{
	Anything fConfig;
	if(!system::LoadConfigFile(fConfig, "SystemFileTest", "any")) {
		ASSERT_EQUAL("'read SystemFileTest.any'", "'could not read
SystemFileTest.any'");
	}
	return fConfig[strClassName][strTestName];
}
```
Die Implementation lädt die Konfigurationsdatei SystemFileTest.any und dessen Inhalte. Wenn die Konfiguration nicht lesbar ist, dann ist die Assertion falsch.

4.5 Test

In Eclipse den Ordner CoastFoundation importieren. Diese SCons Target erstellen:

- CuteFoundationBaseTest
- CuteFoundationAnythingOptionalTest
- CuteFoundationIOTest

Die SCons Option „--run-force" hinzufügen. Die Tests starten mit „Build target". Die Auswertung ist in der Eclipse Konsole. Je nach Konfiguration kann der Test Navigator das Ergebnis nochmals darstellen.

4.6 Schlussbericht

4.6.1 Zielerreichung

Das COAST Testframework hat eine Testsuite und dazugehörige Tests. Die Testsuite ist eine Ablage für die Tests. Die Tests und ihre Funktionen benutzen eine Vererbung TestCase. Das SCons Target startet den Testablauf. Das Resultat ist nur in der Konsole, dafür misst es die Zeit und stellt es für jeden Test und jede Testsuite dar. Der Vorteil ist, dass die JUnit kompatible XML Datei mit Jenkins vollständig auswerten lässt. Damit es diese Datei erstellt, gibt es die SConsider Option „--usetool=TestfwTransformer" und die XML Datei befindet sich im Unterverzeichnis tests.

Ein Test ist nicht migriert: SS1Test. Dieser testet ein veraltetes Verhalten und ist nicht mehr aktuell. Die Member Funktionen ASSERT_ANY_EQUAL, ASSERT_ANY_EQUALM und

ASSERT_ANY_COMPARE sind dazugekommen. Die Assertionen testen die Member Funktionen der Klasse Anything. Die Klasse TestTimer, welche die Zeit für jede Assertion misst, ist weggelassen. Die Quelltexte sind refaktorisiert, so dass eine Member Funktion die Konfigurationsdatei parst (und nicht mehr ein Makro).

4.7 Ausblick

Die Eclipse View „Test Results“ stellt das letzte Ergebnis dar, aber erst nach dem SConsider gestartet wurde. Das funktioniert in einem Schritt. Es ist umständlich, dass zuerst SCons das Testsystem erstellen muss, damit der Test Navigator funktioniert, darum die Idee: Ein Shell Skript startet den gesamten Prozess.

In der Migrationsanleitung ist die std::not_equal_to zusammenzufassen. Die Member Funktion braucht es nicht, wenn daraus lange RegEx entstehen. Das Resultat muss gleich bleiben. Es gibt die Klasse TString, die zu std::string geändert werden soll. Zu diesen Member Funktionen sind RegEx denkbar.

Die neuen Assertionen in CUTE Extensions könnten detailliertere Fehlermeldungen ausgeben. Bislang steht nur die Zeile, die den Fehler hat und die ist in CUTE Extensions. Die Fehlermeldungen der CUTE Assertion sind gleichgeblieben. Der Test Navigator vergleicht die Resultate via Dialogfenster.

5 Installationsanleitung

5.1 Vorbereitung

- Ein Linux Betriebssystem installieren, zum Beispiel Ubuntu 14.10.
- Im Home Verzeichnis ist das Zertifikat: coast-project.pem
- Cevelop 1.1.1

5.2 Installation

1. Terminal starten und installieren:

```
sudo apt-get install build-essential gcc-4.9-multilib g++-4.9-multilib git libboost-all-dev
```

2. PIP vorbereiten:

```
export PIP_CERT=$HOME/coast-project.pem
export PIP_INDEX_URL=https://devpi.coast-project.org/coast/CoastSconsider/+simple/
```

3. Virtualenv herunterladen, installieren und aktivieren:

```
git clone https://github.com/pypa/virtualenv
python virtualenv/virtualenv.py --python=python2.7 .venv27scons
. .venv27scons/bin/activate
```

5.3 Setup COAST Quelltexte

1. Quelltexte vom Git Server herunterladen:

```
git clone https://git.hsr.ch/git/cute_for_coast
export GIT_SSL_NO_VERIFY=true
git clone https://gerrit.coast-project.org/r/p/boost.git 3rdparty/boost
git clone https://gerrit.coast-project.org/r/p/zlib.git 3rdparty/zlib
```

2. COAST Abhängigkeiten installieren:

```
pip install --use-wheel --requirement *require*.txt
```

5.4 SCons unter Cevelop konfigurieren

1. COAST Quelltexte importieren mit „Import…“:

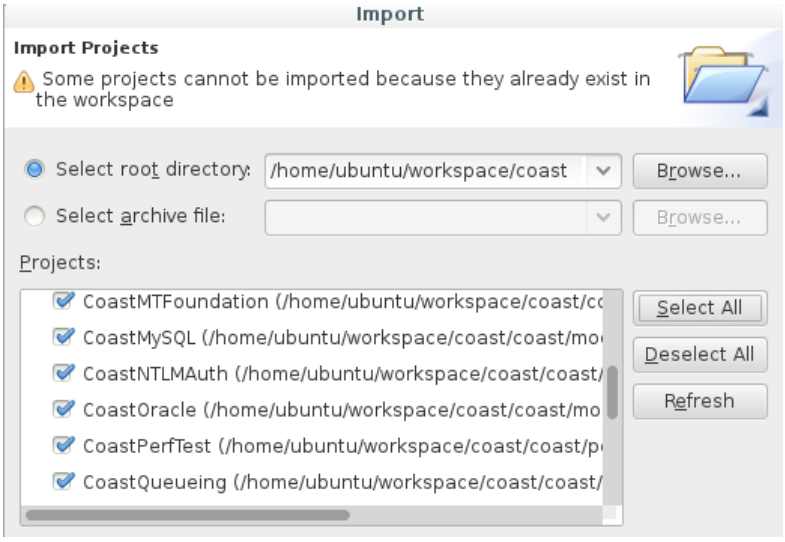


Abbildung 16 Eclipse Dialog: Import

2. „SCons target“ erstellen:

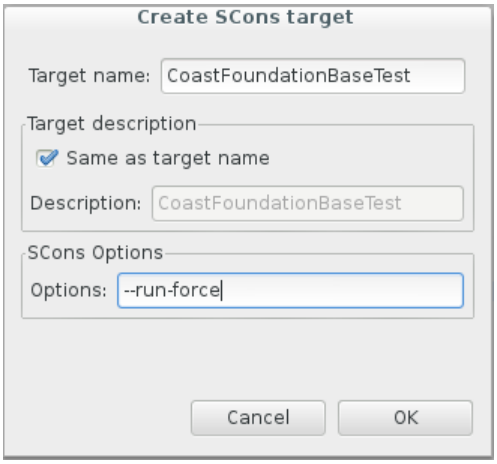


Abbildung 17 Eclipse Dialog: Create SCons target

3. Permanente SCons Optionen sind in der Cevelop Einstellung unter „SCons->Build Settings“ einzugeben:

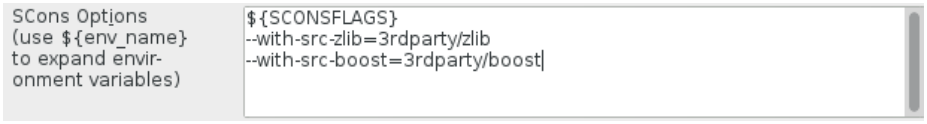


Abbildung 18 Eclipse Einstellung: SCons Options

4. Virtualenv SCons Pfad Einstellung ist unter „SCons“:

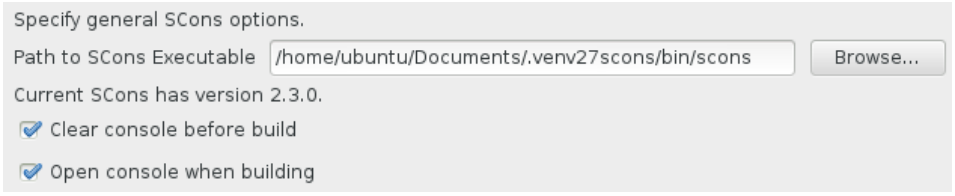


Abbildung 19 Eclipse Einstellung: SCons

5.5 CUTE Test Navigator unter Cevelop konfigurieren

1. SConsider Target einmal kompilieren mit „Build target“:

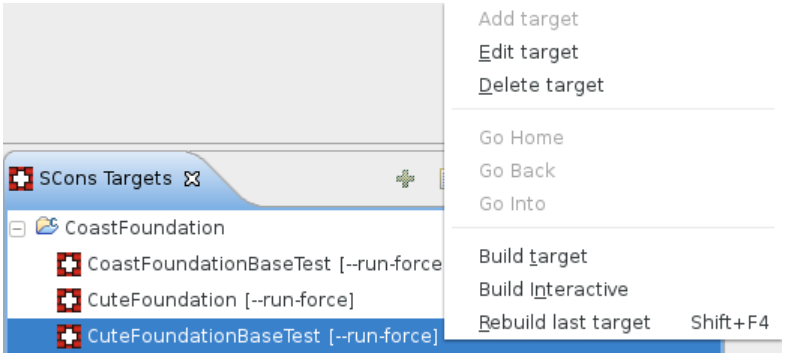


Abbildung 20 Eclipse Perspektive: SCons Targets

2. Run Configurations… einstellen. Das Shell Skript ist im Ordner „scripts“:

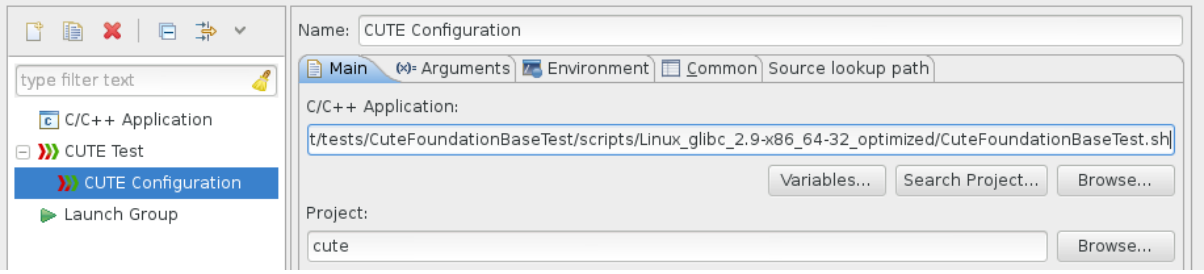


Abbildung 21 Eclipse Einstellung: Run Configurations

5.6 Jenkins Build Shell Skript

```
mkdir -p $WORKSPACE/3rdparty
rm -rf $WORKSPACE/3rdparty
export PIP_CERT=$HOME/coast-project.pem
export PIP_INDEX_URL=https://devpi.coast-project.org/coast/CoastSconsider/+simple/
export GIT_SSL_NO_VERIFY=true
git clone https://gerrit.coast-project.org/r/p/boost.git 3rdparty/boost
git clone https://gerrit.coast-project.org/r/p/zlib.git 3rdparty/zlib
cd $HOME
. .venv27scons/bin/activate
cd $WORKSPACE
pip install --use-wheel --requirement *require*.txt --upgrade
scons -u --jobs=4 --with-src-zlib=3rdparty/zlib --with-src-boost=3rdparty/boost --run-force
CuteFoundationBaseTest cutetest CoastFoundationBaseTest --usetool=TestfwTransformer
```

6 Migrationsanleitung

6.1 COAST Testframework Migration

6.1.1 Vorbereitung

Die Tests und dazugehörende Dateien, Ordner in ein separates Verzeichnis kopieren. Anschliessend in das Verzeichnis die Datei cute_case.cpp erstellen. Der Inhalt dazu:

```
#include "cute_case.h"

const char * setupSuite(cute::suite &s) {
        return "CuteTest";
}
```

Die Bezeichnung "CuteTest" kann angepasst werden. Die SConsider Datei umbenennen. Anstatt dass es mit „Coast“ beginnt, heisst es „Cute“. Der Inhalt um folgendes ergänzt (fett) oder wenn der Eintrag fehlt, dann den gesamten Eintrag hinzufügen:

```
'runConfig': {
   'runParams':'',
},
```

COAST Testframework Inhalte löschen:

Die Header entfernen.

```
#include "FoundationTestTypes.h"
```

6.1.2 Anwendung

Besonderheiten, die nicht oder nur teilweise mit RegEx gelöst sind:

Wenn nach einer if-Anweisung eine Assertion folgt…

```
if(t_assertm(iSSRet >= 0, "expected select to timeout or succeed")) { … }
```

…dann könnte es in CUTE so aussehen:

```
if(ASSERTM("expected select to timeout or succeed", iSSRet >= 0)) { … }
```

CUTE Assertionen haben keinen Rückgabewert und hält den Test immer an, wenn die Assertion falsch ist. Die Assertion ist ohne die if-Anweisung:

```
ASSERTM("expected select to timeout or succeed", iSSRet >= 0);
```

Wenn die Klasse „TString“ des COAST Testframeworks mit der Klasse std::string ersetzt wird,…

```
ASSERT_EQUAL(TString(name()) << "." << fQuery.SlotName(l), expectedStore, result);
```

…ändern die Member Funktionen:

```
std::string m(".");
m += fQuery.SlotName(l);
ASSERT_EQUAL(m, expectedStore, result);
```

Wenn Headers fehlen, dann sind es folgende:

```
#include "Tracer.h"
#include "StringStream.h"
#include "SystemFile.h"
#include "SystemBase.h"
```

Neu hinzugekommen sind die Member Funktionen ASSERT_ANY_EQUAL, ASSERT_ANY_EQUALM und ASSERT_ANY_COMPARE. Die Header ist

```
#include "AssertionAnything.h"
```

und die Vererbung ist

```
: public Assertion
```

Die Assertionen sind für die Member Funktionen in der Klasse Anything.

Wenn eine Member Funktion GetTestCaseConfig fehlt, dann folgende Header

```
ROAnything GetTestCaseConfig(String strClassName, String strTestName);
```

und Implementation hinzufügen.

```
ROAnything Klassename::GetTestCaseConfig(String strClassName, String strTestName) {
        Anything fConfig;
        if(!coast::system::LoadConfigFile(fConfig, "Klassename", "any")) {
                ASSERT_EQUAL("'read Klassename.any'", "'could not read
Klassename.any'");
        }
        return fConfig[strClassName][strTestName];
}
```

Die Bezeichnung Klassename anpassen.

6.2 CUTE Test

6.2.1 Vorbereitung

Es gibt Argumente in Assertionen, die können weder mit Eclipse Find/Replace noch mit Reguläre Ausdrücke (RegEx) ersetzen. Folgende Member Funktionen ersetzt Eclipse Find/Replace ohne Reguläre Ausdrücke:

COAST Header	CUTE Header
TestSuite.h	cute.h
static Test *suite()	static void runAllTests(cute::suite &s)

Die Headers umbenennen.

COAST Implementationen	CUTE Implementationen
setUp	Initialisierungsliste/Konstruktor
tearDown	Destruktor

Die Member Funktionen setUp/tearDown umbenennen. Wenn die Member Funktion leer ist, dann die Member Funktionen löschen. Wenn es den Konstruktor/Destruktor bereits gibt, dann den setUp/tearDown verschieben.

COAST Assertionen	CUTE Assertionen
t_assert(cond)	ASSERT(cond)
t_assertm(cond, msg)	ASSERTM(msg, cond)
assertEqual(exp, act)	ASSERT_EQUAL(exp, act)

assertEqualm(exp, actual, msg)	ASSERT_EQUALM(msg, exp, actual)
assertEqualRaw	-
assertEqualRawm	-
assertCompare(exp, equal_to, act)	ASSERT_EQUAL(exp, act)
assertComparem(exp, equal_to, act, msg)	ASSERT_EQUALM(msg, exp, act)
assertCompare(exp, not_equal_to, act)	ASSERT_NOT_EQUAL_TO(exp, act)
assertComparem(exp, not_equal_to, act, msg)	ASSERT_NOT_EQUAL_TOM(msg, exp, act)
assertAnyEqual(exp, act)	ASSERT_EQUAL(exp, act)/ ASSERT_ANY_EQUAL(exp, act)
assertAnyEqualm(exp, actual, msg)	ASSERT_EQUALM(msg, exp, actual)/ ASSERT_ANY_EQUALM(msg, exp, actual)
assertCharPtrEqual(exp, act)	ASSERT_EQUAL(exp, act)
assertCharPtrEqualm(exp, actual, msg)	ASSERT_EQUALM(msg, exp, actual)
assertDoublesEqual(exp, act)	ASSERT_EQUAL_DELTA(exp, act)
assertDoublesEqualm(exp, actual, delta, msg)	ASSERT_EQUAL_DELTAM(msg, exp, actual, delta)
assertLongsEqual(exp, act)	ASSERT_EQUAL(exp, act)
assertAnyCompareEqual	ASSERT_ANY_COMPARE
assertCompare(exp, greater, act)	ASSERT_GREATER(exp, act)
assertComparem(exp, greater, act, msg)	ASSERT_GREATERM(msg, exp, act)
assertCompare(exp, greater_equal, act)	ASSERT_GREATER_EQUAL(exp, act)
assertComparem(exp, greater_equal, act, msg)	ASSERT_GREATER_EQUALM(msg, exp, act)
assertCompare(exp, less, act)	ASSERT_LESS(exp, act)
assertComparem(exp, less, act, msg)	ASSERT_LESSM(msg, exp, act)
assertCompare(exp, less_equal, act)	ASSERT_LESS_EQUAL(exp, act)
assertComparem(exp, less_equal, act, msg)	ASSERT_LESS_EQUALM(msg, exp, act)
-	ASSERT_THROW
-	ASSERT_THROWM
-	ASSERT_THROWS
-	ASSERT_THROWSM
-	ASSERT*_DDT
-	ASSERT*_DDTM
-	FAIL
-	FAILM

Die Assertionen umbenennen.

COAST Member Funktionen	**CUTE Member Funktionen**
TString	std::string
name()	Je nach dem löschen oder das Makro benutzen: __FUNCTION__

Die Klassen umbenennen und die Member Funktionen ändern.

6.2.2 Anwendung

Eine Anleitung für eine Migration. Es können Reguläre Ausdrücke ausgelassen werden:

1. COAST Testsuite ersetzen: Finde

   ```
   Test \*(.*)::suite\s*\(\)\s*\{.*\s*.*((\s*.*\s*){1,}?)return testSuite;\s*\}
   ```

   und ersetze mit

   ```
   void \1::runAllTests(cute::suite &s) {\2\R}
   ```

Die Member Funktion vor der Testsuite Instanziierung nach der Testsuite Instanziierung verschieben.

2. COAST Testsuite Member Funktion ersetzen: Finde

```
ADD_CASE\s*\(\s*(.*\s*,)\s*(.*,.*)\)
```

und ersetze mit

```
s.push_back(CUTE_SMEMFUN(\2))
```

3. Die Konstruktor Initialisierung in der Implementation löschen. Finde:

```
(#include\s*"TestSuite.h")|(TString tname)|(\s*:\s*TestCaseType\(tname\))|,\s*equal_to
```

und ersetze mit

```

```

4. Die Vererbung und das Konstruktor Argument im Header löschen. Finde:

```
( :.*)|(TString tstrName)
```

und ersetze mit

```

```

5. If-Anweisung löschen: Finde

```
if\s*\(\s*!{0,1}?(?i)assert(.*)\s*\)\s*\{((\s*.*\s*){1,}?)\}
```

und ersetze mit

```
ASSERT\1;\2
```

6. Assertionen auf einen Zeile setzen (ohne „“): Finde

```
„,\R”
```

und ersetze mit

```
„, ”
```

7. Assertion ASSERT_EQUALM. Wähle aus (entweder a oder b);
   a. Finde

```
ASSERT_EQUALM\s*\(\s*([^,]*)\s*,\s*([^,]*)\s*,\s*(.*)\s*\)
```

und ersetze mit

```
ASSERT_EQUALM(\3, \1, \2)
```

   b. Finde

```
ASSERT_EQUALM\s*\(\s*(.*)\s*,\s*(.*)\s*,\s*(.*)\s*\)
```

und ersetze mit

```
ASSERT_EQUALM(\3, \1, \2)
```

8. Assertion ASSERTM. Wähle aus (entweder a oder b):
   a. Finde

```
ASSERTM\s*\(\s*([^,]*)\s*,\s*(.*)\s*\)
```

und ersetze mit

```
ASSERTM(\2, \1)
```

   b. Finde

```
ASSERTM\s*\(\s*(.*)\s*,\s*(.*)\s*\)
```

und ersetze mit

```
ASSERTM(\2, \1)
```

9. Assertionen mit less, less_equal, greater, greater_equal: Finde

```
assertCompare\((.*),\s*(less){0,1}?(less_equal){0,1}?(greater){0,1}?(greater_equal){0
,1}?\s*,(.*)\)
```

und ersetze mit

ASSERT_X(\1,\2)

Es gibt ein X={ LESS, LESS_EQUAL, GREATER, GREATER_EQUAL }

10. Assertion ASSERT_EQUAL_DELTAM: Finde

ASSERT_EQUAL_DELTAM\((.*),(.*),(.*),(.*)\)

und ersetze mit

ASSERT_EQUAL_DELTAM(\4, \1, \2, \3)

11. std::string und <<operator mit append: Finde

(.*) <<\s*(.*),\s*(.*)

und ersetze mit

\1.append(\2), \3

Legende:

Reguläre Ausdrücke[3]	**Bedeutung**
\R	Neue Zeile
\t	Tab
.*	Zeichenkette beliebiger Länge und Inhalt
\n	Wert 1, 2, 3, 4 usw.
\s*	Leerschlag beliebiger Länge
(?i)	Gross-/Kleinschreibung nicht berücksichtigt
{n,m}?	Mindestens n, höchstens m Vorkommen
{n,}?	Mindestens n Vorkommen
^,	Nicht Komma

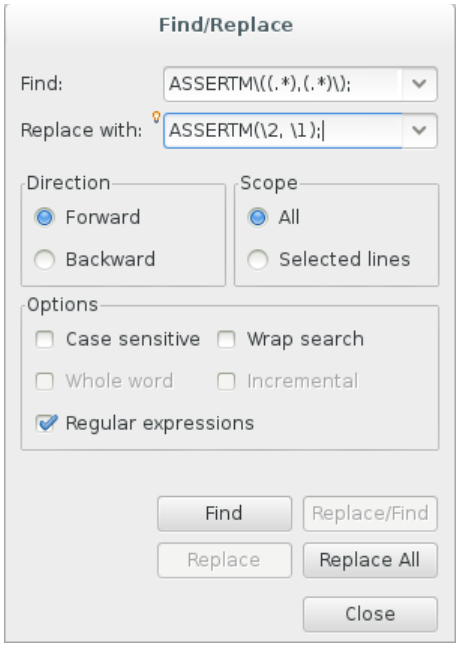


Abbildung 22 Eclipse Dialog: Find/Replace

[3] http://www.eclipse.org/tptp/home/downloads/installguide/gla_42/ref/rregexp.html (27.5.15)

7 Projektplan

7.1 Projekt Übersicht

Das COAST (C++ Open Application Server Toolkit) ist ein Framework, um Webapplikationen in C++ zu entwickeln. Das Testframework ist mit CUTE (http://cute-test.com) zu ersetzen.

7.1.1 Zweck und Ziel

- Eine Migrationsanleitung erstellen.
- Möglichst viele Tests migrieren.
- Untersuchung einer Automatisierung repetitiver Aufgaben, z.B. durch den Einsatz regulärer Ausdrücke.
- Auf https://git.hsr.ch/git/cute_for_coast sind die Quelltexte abgelegt.

7.1.2 Lieferumfang

- Quelltexte

7.1.3 Annahmen und Einschränkungen

- COAST ist mit C++ programmiert.
- COAST Testframework läuft unter Eclipse/Cevelop.
- Nur das Testframework mit CUTE ersetzen.
- Möglichst viele Testfälle migrieren.

7.2 Projektorganisation

Die Aufgabe ist das CUTE als neues Testframework für COAST.

7.2.1 Organisationsstruktur

Projektverantwortlicher: David Tran

7.2.2 Externe Schnittstellen

Betreuer: Prof. Hans Rudin, Mirko Stocker

Gegenleser: Prof. Dr. Eduard Glatz

Experte: Daniel Hiltebrand, Crealogix

7.3 Management Abläufe

7.3.1 Kostenvoranschlag

Insgesamt sind 16 Wochen geplant und mit mindestens 376 Stunden Arbeitsaufwand (pro Woche sind es mindestens 23.5 Stunden).

7.3.2 Zeitliche Planung

Geplant sind Phasen mit Iterationen beginnend ab dem 16. Februar 2015:

- Planung: 0.5 Woche (16.2.15 – 22.2.15)
- COAST/CUTE Framework einarbeiten (23.2.2015 – 12.4.15)
- CUTE Migration (13.4.15 – 7.6.15)
- Demo und Abgabe: 1 Woche (8.6.2015 – 12.6.2015)

Abgabetermin ist spätestens am 12. Juni 2015 um 12:00.

7.3.2.1 Phasen/Iterationen

Planung (16.2.2015 – 22.2.2015)

W	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
		- Plan erstellen															
		- Zeiterfassung planen															
		- IT-Infrastruktur einrichten															

COAST/CUTE Framework einarbeiten (23.2.2015 – 12.4.15)

W	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
								- Cevelop mit COAST einrichten									
								- COAST/CUTE Quelltexte									
								- COAST Testframework									

CUTE Migration (13.4.15 – 7.6.15)

W	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
- CUTE Tests								Os									
- COAST/CUTE Tests vergleichen								te									
- COAST mit CUTE vergleichen								rn									

Demo und Abgabe (8.6.2015 – 12.6.2015)

W	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
- Bericht fertigstellen und abgeben																	
- Demo COAST mit CUTE und Testfälle																	
- Abgabe ist bis am 12.6.15 um 12:00 Uhr																	

7.3.2.2 Meilensteine (MS)

MS1 (16.2.15 – 22.2.15)

- Plan erstellen
- Git Repository erstellen
- VMware Player mit Ubuntu 14.10 installieren
- Cevelop mit COAST Quelltexte einrichten
- Entwicklungsumgebung kennenlernen
- IT Infrastruktur konfigurieren

MS2 (23.2.15 – 8.3.15)

- Entwicklungsumgebung kennenlernen
- COAST mit Cevelop und Virtualenv aufstellen
- COAST/CUTE Framework einsetzen
- COAST Test migrieren (Versuch 1)
- COAST/CUTE Testframework (Testfall/-fälle) vergleichen

MS3 (9.3.15 – 22.3.15)

- CUTE Tests erstellen (Versuch 2)
- COAST/CUTE Testfunktionen abgleichen
- CUTE Tests automatisieren (mindestens zwei Klassen)

MS4 (23.3.15 – 12.4.15)

- Erstellte COAST/CUTE Tests vergleichen
- COAST Tests mit CUTE Tests migrieren
- Automatisierung repetitive Aufgaben untersuchen

MS5 (13.4.15 – 26.4.15)

- Tests migrieren (CoastFoundation: Base Tests)
- Reguläre Ausdrücke suchen für komplexe Anweisungen
- Reguläre Ausdrücke erweitern für komplexe Assertionen

MS6 (27.4.15 – 17.5.15)

- Tests migrieren (mit CUTE Extensions)
- Fehler beseitigen (Redmine Tickets, Dokumentation)
- Tests untersuchen (CoastFoundation: Anything Optional Test)
- Tests refaktorisieren

MS7 (18.5.15 – 7.6.15)

- Modul- und Systemtest
- Migrationsanleitung erstellen
- Tests migrieren
- Reguläre Ausdrücke zusammenfassen

MS8 (8.6.15 – 12.6.15)

- Demonstration COAST mit CUTE

7.3.3 Besprechungen

Vereinbart wurde mit den Betreuern, dass wir uns wöchentlich nach Vereinbarung treffen. Die Sitzungen sind protokolliert.

7.4 Risikomanagement

7.4.1 Risiken

Die Auswertung ist wöchentlich erneuert.

ID	Risiko	Auswirkung	Massnahme	Wahrscheinlichkeit [%]
~~Rx~~	~~Nie mit SCons gearbeitet~~	~~Mehr Zeit planen~~	~~Tutorials anschauen~~	~~5~~
~~Rx~~	~~Zertifikat einfügen klappt nicht~~	~~Verzögerung~~	~~In einem Linux Handbuch lesen~~	~~1~~
~~Rx~~	~~COAST Testframework kompiliert nicht~~	~~Setup ist falsch~~	~~In Cevelop/Eclipse versuchen~~	~~3~~

~~Rx~~	~~Aktuelles CUTE ist inkompatibel zur vorherigen Version~~	~~Tests laufen nie ab~~	~~CUTE Tests erstellen~~	~~1~~
~~Rx~~	~~Ohne Virtualenv gearbeitet~~	~~COAST Test Fehlermeldungen ignoriert~~	~~Dokumentation überprüfen~~	~~1~~
~~Rx~~	~~COAST Tests benutzen unbekanntes C++~~	~~Falsche CUTE Tests~~	~~In einem C++ Buch nachschlagen~~	~~2~~
~~Rx~~	~~CUTE ist falsch ins COAST implementiert~~	~~Keine CUTE Tests~~	~~Separates Projekt aufsetzen~~	~~1~~
~~Rx~~	~~CUTE Test kompiliert nicht~~	~~Falsche Funktionen benutzt~~	~~CUTE Funktionen nachschauen~~	~~1~~
~~Rx~~	~~CUTE Test laufen nicht ab~~	~~CUTE ist falsch konfiguriert~~	~~Testkonfiguration verbessern~~	~~1~~
~~Rx~~	~~Sconsider Datei ist unvollständig~~	~~Testresultat ist zu kurz~~	~~Scons Handbuch nachschauen~~	~~1~~
~~Rx~~	~~CUTE Tests laufen nur einzeln ab~~	~~Tests automatisieren ist schwierig~~	~~Per CUTE TestCase Klasse konfigurieren können~~	~~1~~
~~Rx~~	~~CUTE Test Navigator geht nicht~~	~~In Eclipse kein ‚Green/Redbar'~~	~~Test Navigator von Run Configuration… starten~~	~~1~~
~~Rx~~	~~COAST Testfälle stimmen mit CUTE nicht überein~~	~~Weniger Tests/Testfälle~~	~~Testfall erweitern~~	~~5~~
~~Rx~~	~~Keine Automatisierung repetitiver Aufgaben~~	~~Nur ein Test pro Ablauf~~	~~Manueller Test~~	~~5~~
~~Rx~~	~~Reguläre Ausdrücke mit Eclipse können nicht ersetzen~~	~~Mehr Zeit planen~~	~~Externer RegEx Tester benutzen~~	~~5~~
~~Rx~~	~~COAST Testframework Member Funktion portieren~~	~~Falsches Testresultat~~	~~COAST Testframework Member Funktion weglassen~~	~~1~~
~~Rx~~	~~Build-System kompiliert nicht mit Virtualenv Scons~~	~~Keine automatisierte Tests~~	~~Build-System ist mit Scons ohne Virtualenv~~	~~4~~
~~Rx~~	~~Schwieriger C++ Quelltext, z.B. Headers in Ordner~~	~~Ohne die Headers kompiliert nichts~~	~~Header Dateien zu den Tests/CUTE verschieben~~	~~1~~
~~Rx~~	~~CUTE hat keine gleichwertige Funktion~~	~~Nur Tests migriert wie CUTE Funktionen hat~~	~~Gleichwertige COAST Member Funktionen portieren~~	~~1~~
R1	Tests refaktorisieren ist schwierig	Tests haben Fehler	Dead end erstellen, dann refaktorisieren	2

R2	Neue Testfälle portieren	CUTE ausbauen	Assertionen implementieren	2

7.5 Arbeitspakete

Die Arbeitspakete sind im separaten Dokument *Projektplan.xlsx*. Die wöchentliche Zeiterfassung ist im Kapitel *Zeiterfassung* genauer beschrieben.

7.6 Qualitätsmassnahmen

7.6.1 Dokumentation

Die Migrationsanleitung und Dokumentation sind auf Microsoft OneDrive und owncloud.hsr.ch. Die Betreuer haben auf owncloud.hsr.ch Leserechte.

7.6.2 Projektmanagement

Arbeitspakete sind weiter oben aufgelistet.

7.6.3 Entwicklung

Die Quelltexte sind auf dem internen Git-Server (https://git.hsr.ch). Die Betreuer haben Leserechte.

7.6.3.1 Vorgehen

- Setup von COAST in Eclipse unter Linux
- Analyse des alten COAST Testframeworks und Vergleich mit CUTE
- Migrationsanleitung, wie die Testfälle auf CUTE portiert werden können und welche Besonderheiten beachtet werden müssen
- Migration von Tests und Dokumentation

7.6.3.2 Unit Testing

Die Tests sind im COAST.

7.6.3.3 Code Reviews

Die Quelltext Reviews sind mit dem Betreuer.

7.7 Testen

7.7.1 Unit Test

COAST Testframework hat über 10000 Tests. Möglichst viele davon sind ins CUTE zu migrieren.

Projektplan.xlsx

Arbeitspakete:

Arbeitspakete	Zeit [h]	SOLL	IST	Meilensteine	Phase/Iteration	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Meetings	18,5	18,5	18,5	*	*	1	1	1	1	1	1,5	1,5	1,5	1	1	1	1	1	1	1	1	1
Plan	10	10	10	MS1	Planung	10																
Git	0,1	0,1	0,1	MS1	Planung	0,1																
Dokumentationsvorlagen	2	2	2	MS1	Planung	2																
VMware Player mit Ubuntu und Entwicklerwerkzeug	2	2	2	MS1	Planung	1,5	0,5															
IT Infrastruktur (Ticket, Wiki)	0,5	0,5	0,5	MS1	Planung	0,5																
Risikomanagement	2	2	2	MS1	Planung	1	1															
Cevelop kennenlernen	0,5	0,5	0,5	MS1	Planung	0,5																
CUTE Test schreiben	0,5	0,5	0,5	MS1	Planung	0,5																
SSL Zertifikat	0,1	0,1	0,1	MS1	Planung	0,1																
SCons	6	6	6	MS1	Planung	5,5	0,5															
Zwischentotal	*23,7*	*23,7*	*23,7*																			
Setup COAST unter Cevelop	5	5	5	MS2	COAST/CUTE Framework einarbeiten		5															
PDF: "COAST: C++ Open Application Server Toolkit"	1	1	1,5	MS2	COAST/CUTE Framework einarbeiten		1,5															
Entwicklungsumgebung	1	1	1	MS2	COAST/CUTE Framework einarbeiten		1															
COAST/CUTE Framework Quelltexte	15	15	15,5	MS2	COAST/CUTE Framework einarbeiten		10,5	5														
Hello World' SCons Projekt mit SConsider	2	2	2	MS2	COAST/CUTE Framework einarbeiten			2														
CUTE SConsider Datei	5	5	5	MS2	COAST/CUTE Framework einarbeiten			5														
COAST/CUTE Testframework vergleichen	8	8	8	MS2	COAST/CUTE Framework einarbeiten			7	1													
COAST/CUTE Tests erstellen	8	8	8	MS3-MS4	COAST/CUTE Framework einarbeiten				6				2									
CUTE Tests automatisieren	10	10	10	MS3	COAST/CUTE Framework einarbeiten				9	1												
COAST Testframework Funktionen analysieren	9	9	9	MS3	COAST/CUTE Framework einarbeiten					9												
COAST/CUTE Tests migrieren	20	20	20	MS4	COAST/CUTE Framework einarbeiten							11	9									
COAST Tests mit CUTE Tests ersatzen	7,5	7,5	7,5	MS4	COAST/CUTE Framework einarbeiten						3,5		4									
CUTE Test Navigator	4	4	4	MS4	COAST/CUTE Framework einarbeiten						4											
Automatisierung repetitiver Aufgaben (z.B. reguläre Ausdrücke)	5	5	4	*	COAST/CUTE Framework einarbeiten				0,5			1	2	0,5								
Powerpoint Präsentation vorbereiten	8	8	8	*	COAST/CUTE Framework einarbeiten							5	1,5		1,5							
Dokumentation Bericht allgemein (tech. Bericht/SW-Doku)	25	25	25	*	COAST/CUTE Framework einarbeiten		2	3	5,5	3	3,5	0,5		0,5				2		5		
Dokumentation Anforderungen	5	5	5	*	COAST/CUTE Framework einarbeiten			2	0,5			2,5										
Zwischentotal	*138,5*	*138,5*	*138,5*																			
Neue Reguläre Ausdrücke suchen	10	10	10	MS5	CUTE Migration									10								
Reguläre Ausdrücke erweitern	5	5	5	MS5	CUTE Migration									5								
CUTE Testfall migrieren (Spezialfall)	8	8	8	MS5	CUTE Migration									8								
Tests migrieren	5	5	5	MS5	CUTE Migration										5							
Build-System einarbeiten	5	5	5	MS5	CUTE Migration										5							
Build-System konfigurieren	10	10	10	MS6	CUTE Migration											10						
Tests portieren	10	10	10	MS6	CUTE Migration											10						
CUTE Testfall Fehler beseitigen (Spezialfall)	7	7	7	MS6	CUTE Migration												7					
Fehler beseitigen (Tickets, Dokumentation)	20	20	20	MS5-MS6	CUTE Migration									7	4,5	8,5						
Reguläre Ausdrücke zusammenfassen	5	5	5	MS7	CUTE Migration													0,5	4,5			
Modultests	10	10	10	MS7	CUTE Migration														7	1	2	
Systemtests	5	5	5	MS7	CUTE Migration														2	1	2	
Quelltext Review	5	5	5	MS5-MS7	CUTE Migration										1		1	1	1	1		
Quelltext Refactoring	10	10	10	MS6-MS7	CUTE Migration												5	1	3	1		
Dokumentation Analyse	15	15	15	*	CUTE Migration					9,5	3,5		2									
Dokumentation Ergebnisse	10	10	10	*	CUTE Migration							7	2		1							
Dokumentation Design	15	15	15	*	CUTE Migration						7,5				7,5							
Dokumentation Implementation	10	10	10	*	CUTE Migration												10					
Dokumentation Test	3	3	3	*	CUTE Migration												1		2			
Dokumentation Schlussbericht	10	10	10	*	CUTE Migration													9,5	0,5			
Dokumentation Abstract/Management Summary	25	25	25	*	CUTE Migration													14	1	2	8	
Dokumentation Ausblick	1	1	1	*	CUTE Migration															1		
Poster erstellen	5	5	5	*	CUTE Migration															5		
Zwischentotal	*209*	*209*	*209*																			
Plan nachführen	5	5	5	*	Demo		0,5	0,2	0,3	0,1	0,1	0,2	1	0,2	0,3	0,2	0,4	0,4	0,2	0,5	0,4	
Implementation testen	3	3	3	MS7-MS8	Demo																3	
Demo COAST	0,5	0,5	0,5	MS8	Demo																	0,5
Quelltexte für die Abgabe vorbereiten	1	1	1	MS8	Demo																	1
Dokumentation drucken und binden	4	4	4	MS8	Demo																	4
Dokumentation Abstract/Poster korrigieren	5	5	5	MS7-MS8	Demo																5	
Schlusspräsentation	4	4	4	MS8	Demo																	4
Zwischentotal	*22,5*	*22,5*	*22,5*			*22,7*	*23,5*	*25,2*	*23,8*	*23,6*	*23,6*	*28,7*	*25*	*32,2*	*26,8*	*29,7*	*25,4*	*29,4*	*22,2*	*18,5*	*21,4*	*10,5*
	412,2	**412,2**	**412,2**		**SubtotalKW**	**412,2**																
Insgesamt																						

Kapitel III Anhang

8 Infrastruktur

8.1 Git

Git ist ein Configuration Mangement für Quelltexte.

8.2 owncloud.hsr.ch

Es ist eine persönliche Dateiablage, die man mit anderen HSR Angehörige teilen kann. Der Client synchronisiert regelmässig mit dem Server. Alle Dateien sind auf dem Client und auf dem Server gespeichert.

8.3 Redmine

Eine Projektmanagementsoftware, die Funktionen für die Projektverwaltung, Wikis, Ticketverwaltung hat.

8.4 VMware Player/vSphere (ESX)

Die Software erstellt und verwaltet virtuelle Maschinen.

8.5 Jenkins

Das Build System kompiliert und testet die Quelltexte mit Unit Tests. Ein Git Server stellt die Quelltexte bereit.

9 Glossar

CDT C/C++ Development Tooling

COAST C++ Open Application Server Toolkit

CUTE C++ Unit Testing Easier

RegEx Regular Expression

RUP Rational Unified Process

SCons Software Construction Tool

WUI Web User Interfaces

10 Literaturverzeichnis

[1] https://redmine.coast-project.org/projects/sconsider/wiki (27.5.15)

[2] http://www.cute-test.com/projects/cute/wiki (27.5.15)

[3] http://www.eclipse.org/tptp/home/downloads/installguide/gla_42/ref/rregexp.html (27.5.15)