1 Einführung

1.1 Setup[1]

Zu den kniffligeren Aufgaben bei jedem Softwareprojekt gehört es, alles zu installieren und zum Laufen zu bekommen. In diesem Kapitel erfahren Sie, welche Werkzeuge Sie benötigen, um die in diesem Buch beschriebenen Beispiele zu erstellen und auszuführen. Außerdem lernen Sie einige wichtige Tipps kennen, um nicht die gleichen Fehler zu machen wie ich.

In den ersten Abschnitten erhalten Sie Informationen zur Einrichtung unter Linux und Mac OS. Empfehlungen für Windows-C++-Entwickler folgen in Abschnitt 1.3.3.

1.2 Die Beispiele

Die Quelldateien für dieses Buch können Sie von *http://pragprog.com/titles/lotdd/source_code* herunterladen. Die Beispiele sind nach Kapiteln geordnet.

Bei vielem von dem, was Sie über TDD lernen werden, geht es darum, Code inkrementell weiterzuentwickeln. Deshalb sind auch die Beispiele in den Kapiteln jeweils inkrementell erweiterte Versionen des gleichen Codes. Die Versionsnummern entsprechen dabei den Nummern der Unterverzeichnisse innerhalb des Verzeichnisses für das Kapitel. So befindet sich beispielsweise das erste Codebeispiel von Kapitel 2 in `c2/1/SoundexTest.cpp`, die zweite Version dagegen in `c2/2/SoundexTest.cpp`.

Der Beispielcode steht auch auf GitHub zur Verfügung (*https://github.com/jlangr*). Dort finden Sie für jedes Kapitel, das Code enthält, ein eigenes Repository. Beispielsweise enthält das Repository `c2` das Soundex-Beispiel, das im zweiten Kapitel dieses Buches erstellt wird.

1. Anm. zur dt. Ausgabe: Setup ist in Unit Tests eine übliche Methode, in der Vorbereitungen für den Test getroffen werden. Hier können zum Beispiel die für jeden Test notwendigen Objekte erzeugt werden.

Die Versionsnummer für ein gegebenes Codebeispiel aus dem Buch entspricht einem Branch innerhalb eines GitHub-Repositorys. Beispielsweise finden Sie den Code für das Listing von `c5/4/PlaceDescriptionService.cpp` in der Datei `PlaceDescriptionService.cpp` innerhalb von Branch `4` des Repositorys `c5`.

Jedes Versionsverzeichnis enthält den notwendigen Quellcode einschließlich einer `main`-Funktion, um Tests auszuführen, und eines CMake-Build-Skripts. Um die Beispiele ausführen zu können, müssen Sie einige wenige Tools installieren und konfigurieren. Bei einigen Beispielen ist zusätzlich die Installation von Drittanbieter-Bibliotheken erforderlich.

Um die Beispiele zu erstellen, benötigen Sie einen C++11-konformen Compiler und ein Build-Tool. Bei den meisten ist Google Mock als Unit-Test-Werkzeug erforderlich. In drei Kapiteln wird für die Beispiele jedoch ein anderes Werkzeug für diesen Zweck verwendet, nämlich CppUTest.

Sie können die Quelldistribution ändern, um andere Compiler zu unterstützen (vor C++11) oder um ein anderes Build- oder ein anderes Unit-Test-Werkzeug zu verwenden. Zum Glück ist der Code der meisten Beispiele nicht umfangreich. Die einzige Ausnahme bildet der Bibliothekscode aus Kapitel 7.

In Tabelle 1-1 sind das Unterverzeichnis, das Unit-Test-Werkzeug und die zusätzlichen Drittanbieter-Bibliotheken angegeben, die für die Beispiele in den einzelnen Kapiteln erforderlich sind.

Kapitel		Verzeichnis	Unit-Test-Framework	Drittanbieter-Bibliotheken
2	Testgetriebene Entwicklung: Ein erstes Beispiel	`c2`	Google Mock	Keine
3	Testgetriebene Entwicklung: Grundlagen	`c3`	Google Mock	Keine
4	Tests konstruieren	`c3`	Google Mock	Keine
5	Testdoubles	`c5`	Google Mock	cURL, JsonCpp
6	Inkrementelles Design	`c6`	Google Mock	Boost (Gregorian)
7	Qualitativ hochwertige Tests	`c7`	Google Mock	Boost (Gregorian, Algorithm, Assign)
8	Herausforderungen durch Legacy-Code	`wav`	CppUTest	rlog, Boost (Filesystem)
9	TDD für Threads	`c9`	CppUTest	Keine
10	Weitere Aspekte von TDD	`tpp`	CppUTest	Keine
B	Code-Kata: Umrechner für römische Zahlen	`roman`	Google Mock	Keine

Tab. 1–1 *Verwendete Testframeworks und Drittanbieter-Bibliotheken*

1.3 C++-Compiler

1.3.1 Ubuntu

Ursprünglich habe ich die Beispiele in diesem Buch unter Ubuntu 12.10 mit g++ 4.7.2 erstellt.

Zur Installation von g++ verwenden Sie folgenden Befehl:

```
sudo apt-get install build-essential
```

1.3.2 OS X

Die Beispiele in diesem Buch habe ich auch erfolgreich unter Mac OS X 10.8.3 (Mountain Lion) mithilfe eines gcc-Ports erstellt. Die Version 4.2 von gcc, die in der Zeit, als ich dieses Buch schrieb, mit Xcode ausgeliefert wurde, reicht zur erfolgreichen Kompilierung der C++-Beispiele nicht aus.

Um den gcc-Port zu installieren, benötigen Sie MacPorts, eine Infrastruktur, die die Installation von freier Software auf Ihrem Mac ermöglicht. Weitere Informationen darüber erhalten Sie auf *http://www.macports.org/install.php*.

Als Erstes sollten Sie MacPorts aktualisieren:

```
sudo port selfupdate
```

Installieren Sie anschließend den gcc-Port mit dem folgenden Befehl:

```
sudo port install gcc47
```

Die Ausführung dieses Befehls kann beträchtliche Zeit in Anspruch nehmen. (Sie können am Ende des `port`-Befehls auch die Variante `+universal` angeben. Dadurch wird die Kompilierung von Binärdateien sowohl für PowerPC- als auch für Intel-Architekturen ermöglicht.)

Nachdem Sie den gcc-Port erfolgreich installiert haben, müssen Sie ihn als Standard benennen:

```
sudo port select gcc mp-gcc47
```

Es ist sinnvoll, den Befehl zur Pfadnamenliste hinzuzufügen:

```
hash gcc
```

1.3.3 Windows

Um den Code auf Windows so zum Laufen zu bringen, wie er in diesem Buch (und damit auch in der Quelldistribution) erscheint, ist es am besten, einen MinGW- oder Cygwin-Port von g++ zu verwenden. Weitere Möglichkeiten sind unter anderem die CTP-Version des Microsoft Visual C++-Compilers von November 2012

sowie Clang. Aber zurzeit bieten diese keine ausreichende Unterstützung für den Standard C++11. In diesem Abschnitt gebe ich Ihnen einen kurzen Überblick über die Schwierigkeiten, die Beispiele unter Windows zum Laufen zu bekommen, sowie einige Lösungsvorschläge.

Visual-C++-Compiler, CTP-Version November 2012

Ein CTP-Release (Community Technology Preview) des Visual C++11-Compilers steht zum Download zur Verfügung.[2] Beschrieben wird es in einem Blogeintrag[3] des Visual C++-Teams.

Ein erster Versuch, die CTP-Version für die Beispiele in diesem Buch zu verwenden, ließ schnell die folgenden Probleme erkennen:

- Die Memberinitialisierung innerhalb der Klasse scheint noch nicht vollständig unterstützt zu sein.
- In der Bibliothek `std` ist die Unterstützung für C++11 besonders mangelhaft. Beispielsweise unterstützen die Collection-Klassen noch keine einheitlichen Initialisiererlisten. Auch gibt es noch keine Implementierung für `std::unordered_map`.
- Die von Google Mock und Google Test verwendeten variadischen Templates werden ebenfalls noch nicht vollständig unterstützt. Wenn Sie versuchen, Google Mock zu erstellen, wird ein Kompilierungsfehler ausgegeben. In einem solchen Fall müssen Sie den Präprozessordefinitionen einen Eintrag hinzufügen, der `_VARIADIC_MAX` für alle betroffenen Projekte auf 10 setzt. Weitere Informationen über die Behebung dieses Problems erhalten Sie auf *http://stackoverflow.com/questions/12558327/google-test-in-visual-studio-2012*.

Windows-Beispielcode

Kurz vor der Veröffentlichung dieses Buches habe ich mich bemüht, die Windows-Codebeispiele funktionsfähig zu machen (indem ich nicht unterstützte C++11-Elemente entfernt habe). Die umgearbeiteten Beispiele finden Sie in einem eigenen Satz von Repositorys (je eines pro Kapitel) auf meiner GitHub-Seite (*https://github.com/jlangr*). Weitere Informationen über die Windows-Beispiele veröffentliche ich nach und nach im Google-Groups-Forum unter *https://groups.google.com/forum/?fromgroups#!forum/modern-cpp-with-tdd*.

Die Windows-Repositorys auf GitHub enthalten Lösungsdateien (`.sln`) und Projektdateien (`.vcxproj`). Damit können Sie den Beispielcode in Visual Studio Express 2012 für Windows Desktop laden und mit MSBuild Tests für diese Beispiele erstellen und an der Befehlszeile ausführen.

2. *http://www.microsoft.com/en-us/download/details.aspx?id=35515*
3. *http://blogs.msdn.com/b/vcblog/archive/2012/11/02/visual-c-c-11-and-the-future-of-c.aspx*

Es sollte auch nicht allzu dramatisch sein, die Codebeispiele selbst umzuarbeiten. Eine Änderung der Initialisierung außerhalb der Klasse ist nicht schwer, und `std::unordered_map` können Sie einfach durch `std::map` ersetzen. Da viele der neuen Ergänzungen zu C++11 aus der Bibliothek `boost::tr1` stammen, sollte es auch möglich sein, die Boost-Implementierungen direkt zu ersetzen.

Tipps zu Windows

Ich habe mich im Internet über eine Reihe von Hindernissen wie Kompilierungswarnungen und -fehler sowie andere Build-Probleme schlau gemacht. Dabei habe ich die in Tabelle 1-2 angegebenen Erkenntnisse gewonnen:

Fehler/Problem	Lösung
C297:`'std:tuple'`: zu viele Template-Argumente	Fügen Sie die Präprozessordefinition `_VARIADIC_MAX=10` hinzu (siehe *http://stackover-flow.com/questions/8274588/c2977-stdtuple-toomany-template-arguments-msvc11*).
Das angegebene Plattform-Toolset (v110) ist nicht installiert oder ungültig.	Setzen Sie VisualStudioVersion auf 11.0.
Wo ist `msbuild.exe`?	Bei mir befindet sich diese Datei unter `C:\Windows\Microsoft.NET\Framework\v4.0.30319`.
Warnung C4996: `'std::_Copy_impl'`: Funktionsaufruf mit Parametern, die möglicherweise unsicher sind.	`-D_SCL_SECURE_NO_WARNINGS`
Das Konsolenfenster wird geschlossen, wenn Sie die Ausführung eines Tests mit Strg + F5 abschließen.	Setzen Sie Konfigurationseigenschaften → Linker → System → Teilsystem auf Konsole (`/SUBSYSTEM:CONSOLE`).
Visual Studio versucht für Boost-Merkmale, die nur Headerdateien benötigen, automatisch eine Verknüpfung zu einer Bibliothek herzustellen.	Fügen Sie die Präprozessordirektive `BOOST_ALL_NO_LIB` hinzu.

Tab. 1–2 *Fehler in Visual Studio und mögliche Lösungen*

Viele der Lösungen für diese Probleme sind bereits in die Projektdateien eingearbeitet.

Vorschau auf Visual Studio 2013

Kurz vor Ablauf meiner Abgabefrist für allerletzte Änderungen an diesem Buch hat Microsoft erste Downloads für Visual Studio 2013 veröffentlicht, die eine erweiterte Konformität mit C++11 sowie die Unterstützung für einige vorgeschlagene Funktionen von C++14 zu bieten scheinen. Der Windows-Code auf GitHub ist zurzeit für die CTP-Version von November 2012 geeignet, aber es wird in Kürze

neue Versionen geben, die C++11 noch besser nutzen, wenn wir (einige großartige Helfer und ich) in Visual Studio 2013 damit arbeiten. Ich hoffe, dass eine Windows-spezifische Version irgendwann gar nicht mehr notwendig sein wird. Freuen wir uns auf einen vollständig C++11-konformen Windows-Compiler!

1.4 CMake

Um plattformübergreifende Builds zu unterstützen, habe ich mich für CMake entschieden.

Die Version zum Erstellen der Beispiele für Ubuntu ist CMake 2.8.9. Zur Installation von CMake verwenden Sie folgenden Befehl:

```
sudo apt-get install cmake
```

Benutzer von OS X benötigen CMake 2.8.10.2. Die Installation können Sie mithilfe der Downloads auf *http://www.cmake.org/cmake/resources/software.html* durchführen.

Wenn Sie CMake für die bereitgestellten Build-Skripts ausführen, wird möglicherweise die folgende Fehlermeldung angezeigt:

```
Make Error: your CXX compiler: "CMAKE_CXX_COMPILER-NOTFOUND" was not found.
Please set CMAKE_CXX_COMPILER to a valid compiler path or name.
```

Das bedeutet, dass kein geeigneter Compiler gefunden wurde. Das kann der Fall sein, wenn Sie gcc statt g++ installiert haben. Unter Ubuntu können Sie dieses Problem lösen, indem Sie `build-essential` installieren. Unter OS X definieren Sie CXX oder ändern die Definition dafür:

```
export CC=/opt/local/bin/x86_64-apple-darwin12-gcc-4.7.2
export CXX=/opt/local/bin/x86_64-apple-darwin12-g++-mp-4.7
```

1.5 Google Mock

Das in vielen Beispielen dieses Buches verwendete Google Mock ist ein Framework zum Erstellen von Mocks (Attrappen) und zur Beschreibung von Annahmen. Es enthält das Unit-Test-Framework Google Test, wobei ich beide Begriffe in diesem Buch austauschbar verwende, der Einfachheit halber aber meistens nur von Google Mock schreibe. Für einige der Features, die ich als Bestandteile von Google Mock bezeichne, müssen Sie daher möglicherweise die Dokumentation von Google Test zurate ziehen.

Da Sie Google Mock mit den Beispielen verlinken, müssen Sie zunächst die Google-Mock-Bibliothek erstellen. Die folgenden Anleitungen helfen Ihnen dabei. Sie können sich auch die mit Google Mock mitgelieferte Datei `README.txt` ansehen, um ausführlichere Installationsanweisungen zu erhalten (siehe *https://code.google.com/p/googlemock/source/browse/trunk/README*).

1.5.1 Google Mock installieren

Die offizielle Google-Mock-Website ist *https://code.google.com/p/googlemock/*. Die Downloads finden Sie auf *https://code.google.com/p/googlemock/downloads/list*. Die Beispiele in diesem Buch wurden mit Google Mock 1.6.0 erstellt.

Entpacken Sie die heruntergeladene ZIP-Datei (z.B. `gmock-1.6.0.zip`) zum Beispiel in Ihrem Benutzerordner.

Erstellen Sie dann wie im folgenden Beispiel die Umgebungsvariable `GMOCK_HOME`, die auf dieses Verzeichnis zeigt:

```
export GMOCK_HOME=/home/jeff/gmock-1.6.0
```

Unter Windows geht das wie folgt:

```
setx GMOCK_HOME c:\Users\jlangr\gmock-1.6.0
```

Unix

Wenn Sie unter Unix die Build-Anweisungen in der `README`-Datei überspringen wollen, können Sie auch, so wie ich es getan habe, mit den folgenden Schritten ans Ziel gelangen. Ich habe Google Mock mithilfe von CMake erstellt. Gehen Sie im Wurzelverzeichnis Ihrer Google-Mock-Installation (im Folgenden `$GMOCK_HOME` genannt) wie folgt vor:

```
mkdir mybuild
cd mybuild
cmake ..
make
```

Das Build-Verzeichnis kann auch einen anderen Namen tragen, allerdings erwarten die Beispiele in diesem Buch `mybuild`. Wenn Sie diesen Namen ändern, müssen Sie auch alle `CMakeLists.txt`-Dateien anpassen.

Außerdem müssen Sie Google Test erstellen, das in Google Mock verschachtelt ist:

```
cd $GMOCK_HOME/gtest
mkdir mybuild
cd mybuild
cmake ..
make
```

Windows

In der Google-Mock-Distribution finden Sie die Datei `.\msvc\2010\gmock.sln`, die in Visual Studio 2010 und neueren Versionen funktionieren sollte. (Außerdem gibt es die Datei `.\msvc\2005.gmock.sln`, die für Visual Studio 2005 und 2008 vorgesehen ist.)

Um Google Mock in Visual Studio 2010 und 2012 zu kompilieren, müssen Sie die Projekte so einrichten, dass Sie die CTP von November 2012 nutzen. Öffnen Sie in den Projekteigenschaften Konfigurationseigenschaften → Allgemein → Plattformtoolsets und wählen Sie die CTP aus.

Die CTP bietet keine Unterstützung für variadische Templates. (In Visual Studio 2013 wird eine solche Unterstützung *möglicherweise* vorhanden sein.) Stattdessen werden solche Templates künstlich simuliert.[4] Dazu müssen Sie mit einer Präprozessordefinition den Wert von `_VARIADIC_MAX` über die Standardeinstellung 5 hinaus anheben. Ein Wert von 10 ist gut geeignet.

Wenn Sie Projekte erstellen, die Google Mock nutzen, müssen Sie darin auf den richtigen Speicherort der Include- und Bibliotheksdateien verweisen. Öffnen Sie Konfigurationseigenschaften → Visual C++-Verzeichnisse und gehen Sie wie folgt vor:

- Fügen Sie `$(GMOCK_HOME)\msvc\2010\Debug` zu den Bibliotheksverzeichnissen hinzu.
- Fügen Sie `$(GMOCK_HOME)\include` zu den Include-Verzeichnissen hinzu.
- Fügen Sie `$(GMOCK_HOME)\gtest\include` zu den Include-Verzeichnissen hinzu.

Fügen Sie `gmock.lib` unter Linker → Eingabe zu den zusätzlichen Abhängigkeiten hinzu.

Außerdem müssen Sie sicherstellen, dass Google Mock und Ihr Projekt mit demselben Speichermodell erstellt werden. Standardmäßig verwendet Google Mock `/MTd`.

1.5.2 Ein Main-Programm zum Ausführen von Google-Mock-Tests erstellen

Der Code für Beispiele in diesem Buch enthält jeweils eine `main.cpp`-Datei zur Verwendung mit Google Mock.

c2/1/main.cpp

```
#include "gmock/gmock.h"

int main(int argc, char** argv) {
   testing::InitGoogleMock(&argc, argv);
   return RUN_ALL_TESTS();
}
```

Die hier gezeigte `main()`-Funktion initialisiert als Erstes Google Mock und übergibt dabei alle eventuell bereitgestellten Befehlszeilenparameter. Weitere Informationen erhalten Sie unter *http://code.google.com/p/googletest/wiki/Primer#Writing_the_main()_Function*.

4. *http://stackoverflow.com/questions/12558327/google-test-in-visual-studio-2012*

1.6 CppUTest

Bei CppUTest handelt es sich um ein weiteres Unit-Test-Framework für C++. Möglicherweise bevorzugen Sie es gegenüber Google Test/Google Mock, da es viele vergleichbare Features aufweist und überdies einen eingebauten Speicherleckdetektor bietet. Weitere Beispiele zur Verwendung von CppUTest finden Sie im Buch »Test Driven Development for Embedded C« von James Grenning [Gre 10].

1.6.1 CppUTest installieren

(Hinweis: Diese Anleitung gilt für CppUTest 3.3. Version 3.4 umfasst eine Reihe von Änderungen, wurde aber knapp vor meinem Abgabetermin veröffentlicht, sodass ich sie in diesem Buch nicht mehr berücksichtigen konnte.)

Die Website des Projekts CppUTest lautet *http://www.cpputest.org/*. Die Downloads finden Sie auf *http://cpputest.github.io/cpputest/*. Laden Sie die passende Datei herunter und entpacken Sie sie am besten in ein neues Verzeichnis namens `cpputest` innerhalb Ihres Benutzerordners.

Erzeugen Sie wie im folgenden Beispiel die Umgebungsvariable `CPPUTEST_HOME`:

```
export CPPUTEST_HOME=/home/jeff/cpputest
```

CppUTest können Sie mithilfe von `make` erstellen. Außerdem müssen Sie `CppUTestExt` erstellen, das Unterstützung für Mocks bietet:

```
cd $CPPUTEST_HOME
./configure
make
make -f Makefile_CppUTestExt
```

Installieren Sie CppUTest mit dem Befehl `make install` in `/usr/local/lib`.

CppUTest können Sie auch mit CMake erstellen, wenn Ihnen das lieber ist. Für die Verwendung unter Windows werden Batchdateien für Visual Studio 2008 und 2010 bereitgestellt. Diese Dateien nutzen `MSBuild`.

1.6.2 Ein Main-Programm zum Ausführen von CppUTest-Tests erstellen

Der Code für das WAV-Reader-Beispiel in diesem Buch enthält die Datei `testmain.cpp`, die zur Verwendung mit CppUTest gedacht ist.

wav/1/testmain.cpp

```
#include "CppUTest/CommandLineTestRunner.h"

int main(int argc, char** argv) {
   return CommandLineTestRunner::RunAllTests(argc, argv);
}
```

1.7 libcurl

libcurl bietet eine clientseitige Bibliothek zur URL-Übertragung, die HTTP und viele andere Protokolle unterstützt. Außerdem unterstützt sie das Tool cURL zur Übertragung von Befehlszeilen, weshalb ich die Bibliothek in diesem Buch als cURL bezeichne.

Die Website des Projekts cURL lautet *http://curl.haxx.se/*. Die Downloads finden Sie auf *http://curl.haxx.se/download.html*. Laden Sie die passende Datei herunter und entpacken Sie sie beispielsweise in Ihren Benutzerordner. Erzeugen Sie dann wie im folgenden Beispiel die Umgebungsvariable `CURL_HOME`:

```
export CURL_HOME=/home/jeff/curl-7.29.0
```

Um die Bibliothek zu erstellen, können Sie CMake verwenden:

```
cd $CURL_HOME
mkdir build
cd build
cmake ..
make
```

1.8 JsonCpp

JsconCpp bietet Unterstützung für das Datenaustauschformat JSON (JavaScript Object Notation).

Die Website des Projekts JsonCpp lautet *http://jsoncpp.sourceforge.net/*. Die Downloads finden Sie auf *http://sourceforge.net/projects/jsoncpp/files/*. Laden Sie die passende Datei herunter und entpacken Sie sie zum Beispiel in Ihren Benutzerordner. Erzeugen Sie dann wie im folgenden Beispiel die Umgebungsvariable `JSONCPP_HOME`:

```
export JSONCPP_HOME=/home/jeff/jsoncpp-src-0.5.0
```

Für JsconCpp ist das Python-Build-System Scons erforderlich. Unter Ubuntu installieren Sie Scons wie folgt:

```
sudo apt-get install scons
```

Wechseln Sie in das Verzeichnis `$JSONCPP_HOME` und erstellen Sie die Bibliothek mithilfe von Scons:

```
scons platform=linux-gcc
```

Unter OS X geben Sie als Plattform `linux-gcc` an. Zumindest hat das bei meiner Installation funktioniert. Der Build-Vorgang für JsonCpp hat bei mir dazu geführt, dass die Datei `$JSONCPP_HOME/libs/linux-gcc-4.7/libjson_linux-gcc-4.7_libmt.a` angelegt wurde. Erstellen Sie wie folgt einen symbolischen Link dorthin:

```
cd $JSONCPP_HOME/libs/linux-gcc-4.7
ln -s libjson_linux-gcc-4.7_libmt.a libjson_linux-gcc-4.7.a
```

1.9 rlog

rlog bietet eine Möglichkeit zur Protokollierung von Nachrichten für C++.

Die Website des Projekts rlog lautet *https://code.google.com/p/rlog/*. Laden Sie die passende Datei herunter und entpacken Sie sie zum Beispiel in Ihren Benutzerordner. Erzeugen Sie dann wie im folgenden Beispiel die Umgebungsvariable `RLOG_HOME`:

```
export RLOG_HOME=/home/jeff/rlog-1.4
```

Unter Ubuntu erstellen Sie rlog mit den folgenden Befehlen:

```
cd $RLOG_HOME
./configure
make
```

Unter OS X konnte rlog nur nach der Anwendung eines Patches kompilieren. Weitere Informationen über dieses Problem sowie den Code des Patches finden Sie unter *https://code.google.com/p/rlog/issues/detail?id=7*. Ich habe den Code in dem dritten Kommentar verwendet (»This smaller diff ...«). Den Patchcode erhalten Sie auch in der Quelldistribution als `code/wav/1/rlog.diff`.

Um den Patch anzuwenden und rlog zu erstellen, gehen Sie folgendermaßen vor:

```
cd $RLOG_HOME
patch -p1 [path to file]/rlog.diff
autoreconf
./configure
cp /opt/local/bin/glibtool libtool
make
sudo make install
```

Der Befehl `configure` kopiert die Binärdatei `libtool` in das Verzeichnis `rlog`, aber dies ist *nicht* die von rlog erwartete Binärdatei! Der Befehl, der `glibtool` über `libtool` kopiert, korrigiert diesen Fehler.

Wenn der Patch bei Ihnen nicht funktioniert, können Sie versuchen, manuelle Änderungen vorzunehmen. In der Datei `$RLOG_HOME/rlog/common.h.in` finden Sie folgende Zeile:

```
# define RLOG_SECTION __attribute__ (( section("RLOG_DATA") ))
```

Ersetzen Sie sie durch Folgendes:

```
#ifdef _APPLE_
# define RLOG_SECTION __attribute__ (( section("__DATA, RLOG_DATA") ))
#else
# define RLOG_SECTION __attribute__ (( section("RLOG_DATA") ))
#endif
```

Sollten Sie dann *immer noch* Probleme damit haben, rlog zu erstellen (das ist sowohl unter Mac OS als auch unter Windows eine ziemliche Herausforderung!), verzweifeln Sie nicht! In dem Beispiel aus Abschnitt 8.9, in dem es um die Arbeit mit Legacy-Code geht, erfahren Sie, wie Sie komplett auf rlog verzichten können.

1.10 Boost

Boost bietet eine große Menge an grundlegenden C++-Bibliotheken.

Die Website des Projekts Boost lautet *http://www.boost.org*. Die Downloads finden Sie auf *http://sourceforge.net/projects/boost/files/boost*. Es werden regelmäßig aktualisierte Versionen bereitgestellt. Laden Sie die passende Datei herunter und entpacken Sie sie zum Beispiel in Ihren Benutzerordner. Erzeugen Sie dann wie im folgenden Beispiel Umgebungsvariablen sowohl für `BOOST_ROOT` als auch für die von Ihnen installierte Boost-Version:

```
export BOOST_ROOT=/home/jeff/boost_1_53_0
export BOOST_VERSION=1.53.0
```

Viele Boost-Bibliotheken erfordern lediglich Headerdateien. Wenn Sie der vorstehenden Anweisung gefolgt sind, können Sie alle Beispiele, die Boost verwenden, erstellen. Die einzige Ausnahme bildet der Code in Kapitel 8.

Um ihn zu erzeugen, müssen Sie Bibliotheken erstellen und von Boost aus verlinken. Zum Erstellen habe ich folgende Befehle verwendet:

```
cd $BOOST_ROOT
./bootstrap.sh --with-libraries=filesystem,system
./b2
```

Diese Befehle sollten auch bei Ihnen funktionieren. Wenn nicht, lesen Sie die Anleitung unter *http://ubuntuforums.org/showthread.php?t=1180792*. (Beachten Sie jedoch, dass das Argument von `bootstrap.sh` nicht `--with-library`, sondern `--with-libraries` lauten muss.)

1.11 Beispiele erstellen und Tests ausführen

Nachdem Sie die passende Software installiert haben, können Sie alle Versionen der Beispiele erzeugen und anschließend die Tests ausführen. Im Verzeichnis für eine Version eines Beispiels erstellen Sie als Erstes mithilfe von CMake ein Makefile:

```
mkdir build
cd build
cmake ..
```

Der Legacy-Code (siehe Kap. 8) verwendet *Bibliotheken* von Boost, nicht nur die Header. `CMakeLists.txt` nutzt die Umgebungsvariable `BOOST_ROOT`, die Sie zweimal definiert haben: erstens ausdrücklich durch `include_directories`, um anzugeben, wo die Boost-Header zu finden sind, und zweitens implizit, wenn CMake `find_package` ausführt, um die Boost-Bibliotheken zu finden.

Wenn Sie einen Build des Legacy-Codes versuchen, erhalten Sie möglicherweise die Fehlermeldung, dass Boost nicht zu finden ist. In diesem Fall können Sie den Speicherort ändern, indem Sie bei der Ausführung von CMake einen Wert für `BOOST_ROOT` übergeben:

```
cmake -DBOOST_ROOT=/home/jeff/boost_1_53_0 ..
```

Anderenfalls müssen Sie dafür sorgen, dass Sie die Boost-Bibliotheken korrekt erstellt haben.

Nachdem Sie mit CMake ein Makefile angelegt haben, können Sie die Beispiele erstellen, indem Sie in deren Build-Verzeichnis wechseln und dort Folgendes ausführen:

```
make
```

Um Tests ablaufen zu lassen, führen Sie den folgenden Befehl ebenfalls im Build-Verzeichnis des Beispiels aus:

```
./test
```

Die ausführbare Datei für den Test des Bibliotheksbeispiels in Kapitel 7 finden Sie in `build/libraryTests`.

1.12 Teardown[5]

In diesem Kapitel haben Sie erfahren, was Sie benötigen, um die Beispiele in diesem Buch zu erstellen und auszuführen. Denken Sie immer daran, dass man am besten lernt, wenn man sich selbst die Finger schmutzig macht. Führen Sie die Beispiele also während der Lektüre aus.

Wenn Sie Probleme haben sollten, etwas einzurichten, wenden Sie sich zunächst an einen Vertrauten, der Ihnen eventuell weiterhelfen kann. Ein zweites Paar Augen findet oft schnell die Ursache für ein Problem, mit dem Sie lange gekämpft haben. Sie können auch die Webseite zu diesem Buch unter *http://pragprog.com/titles/lotdd* aufsuchen, wo Sie hilfreiche Tipps und ein Diskussionsforum finden. Wenn Sie und Ihr Helfer beide nicht mehr weiterkommen, senden Sie mir bitte eine E-Mail (in englischer Sprache).

5. Anm. zur dt. Ausgabe: Teardown wird in Unit Tests benutzt, um nach jedem Test wieder aufzuräumen. So kann z.B. durch Objekte allokierter Speicher wieder freigegeben werden.