Fachhochschule
Münster University of
Applied Sciences

Build-Management

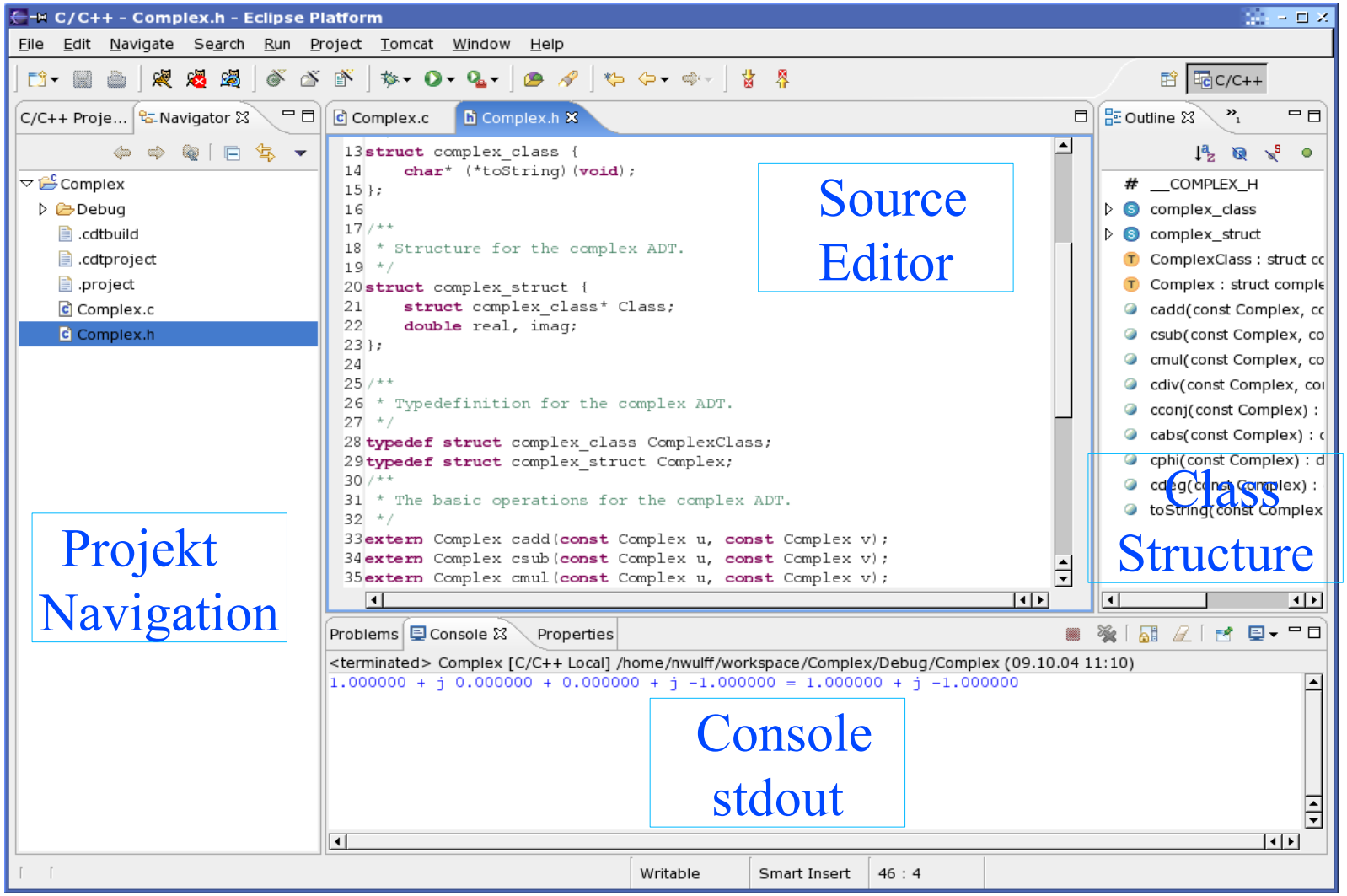
Der Einsatz von Make, Ant, Maven, Gradle und Co.

Prof. Dr. Nikolaus Wulff

Integrierter Arbeitsplatz

- Eine IDE wie Eclipse, JBuilder oder NetBeans unterstützt die alltägliche Arbeit.
- Sie bietet verschiedene Sichten auf das jeweilige Projekt:
  - Projektnavigation & Struktur
  - Quellcode Editor & Struktur
  - Commandozeile
  - Datenbankverbindungen
  - Debugger
  - Versionsverwaltung
  - etc ...

Eclipse IDE


C/C++ - Complex.h - Eclipse Platform
File Edit Navigate Search Run Project Tomcat Window Help
C/C++
C/C++ Proje... Navigator
Complex
Debug
.cdtbuild
.cdtproject
.project
Complex.c
Complex.h
Projekt Navigation
Complex.c Complex.h
13 struct complex_class {
14 char* (*toString)(void);
15 };
16
17 /**
18 * Structure for the complex ADT.
19 */
20 struct complex_struct {
21 struct complex_class* Class;
22 double real, imag;
23 };
24
25 /**
26 * Typedefinition for the complex ADT.
27 */
28 typedef struct complex_class ComplexClass;
29 typedef struct complex_struct Complex;
30 /**
31 * The basic operations for the complex ADT.
32 */
33 extern Complex cadd(const Complex u, const Complex v);
34 extern Complex csub(const Complex u, const Complex v);
35 extern Complex cmul(const Complex u, const Complex v);
Source Editor
Outline
__COMPLEX_H
complex_class
complex_struct
ComplexClass : struct cc
Complex : struct comple
cadd(const Complex, cc
csub(const Complex, co
cmul(const Complex, co
cdiv(const Complex, co
cconj(const Complex) :
cabs(const Complex) :
cphi(const Complex) :
toString(const Complex
Class Structure
Problems Console Properties
<terminated> Complex [C/C++ Local] /home/nwulff/workspace/Complex/Debug/Complex (09.10.04 11:10)
1.000000 + j 0.000000 + 0.000000 + j -1.000000 = 1.000000 + j -1.000000
Console stdout
Writable Smart Insert 46 : 4

Build- und Configurationmanagement

- Problem:
  - Ein Programm besteht aus vielen Quellen, die unter Umständen wechselseitig voneinander abhängig sind.
  - Mehrere Entwickler sind an der Erstellung beteiligt.
  - Unterschiedliche Sourcen und Konfigurationsdateien müssen verwaltet und synchronisiert werden.
  -
- Lösung:
  - Klares verbindliches Projektmanagement
  - Zentrale Versionsverwaltung
  - Automatisierte Build

Mehr als nur IDE ...

Eine integrierte Entwicklungsumgebung ist eine gute Investition, um ein komplexes Softwareprodukt zu erstellen. Nur dies alleine reicht nicht aus, um in größeren Teams kooperativ zu arbeiten.

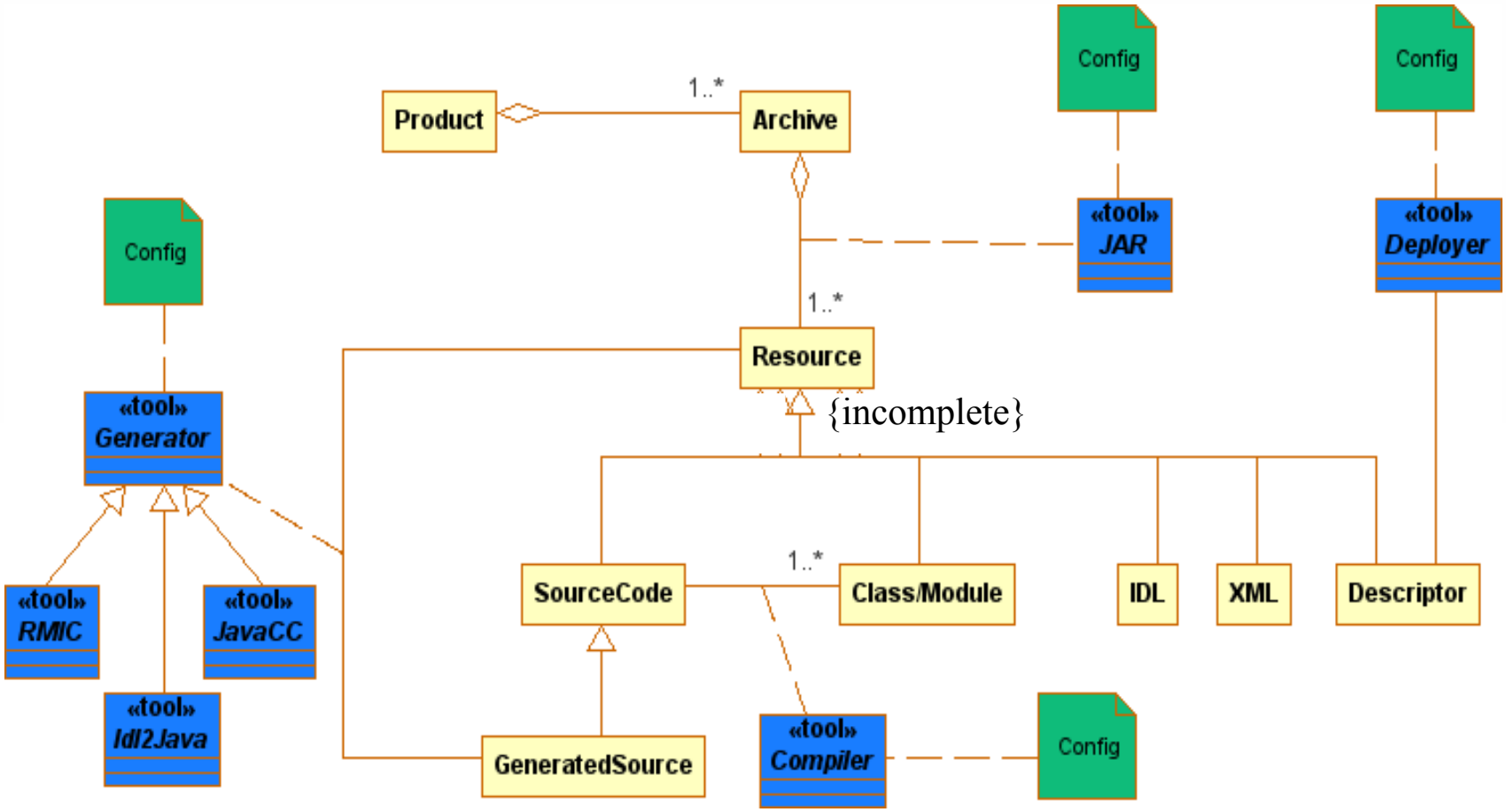
Benötigt werden ferner Tools für:

- Version Control
- Build Management
- Configuration Management
- Change Control
- ...

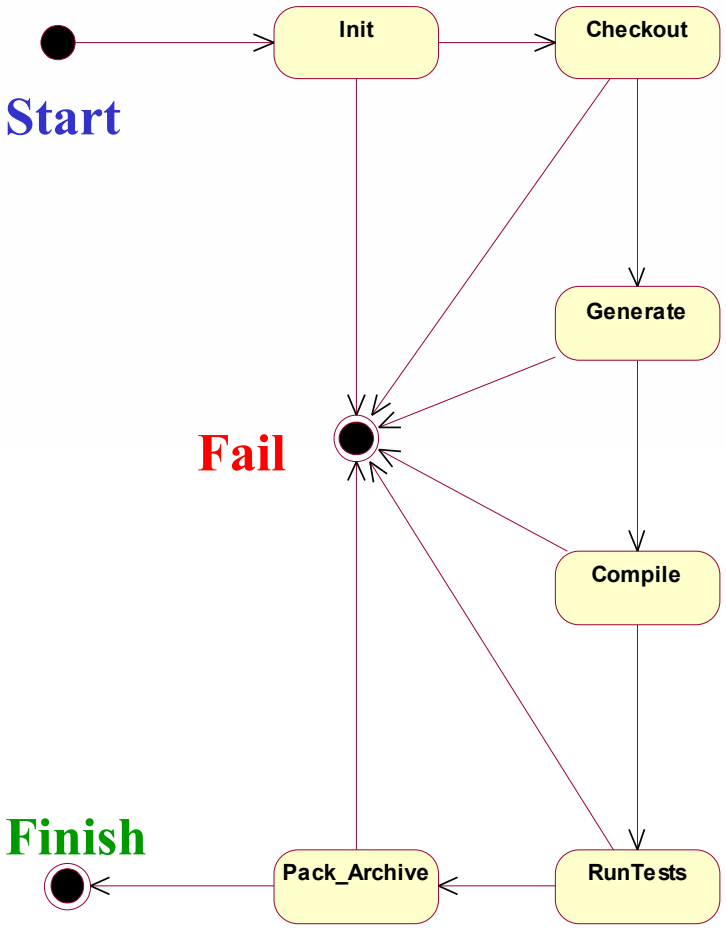
Projektinfrastruktur

- Arbeitsplatzumgebung und IDE‘s
- Versionsverwaltung
- Automatische Build mit make oder Apache Ant
- Permanente Integration und Nightly Build
- Automatische Komponenten- und Regressiontests
- Performance- und Systemtest
- Softwarequalität und Metriken

Ein typisches Projekt

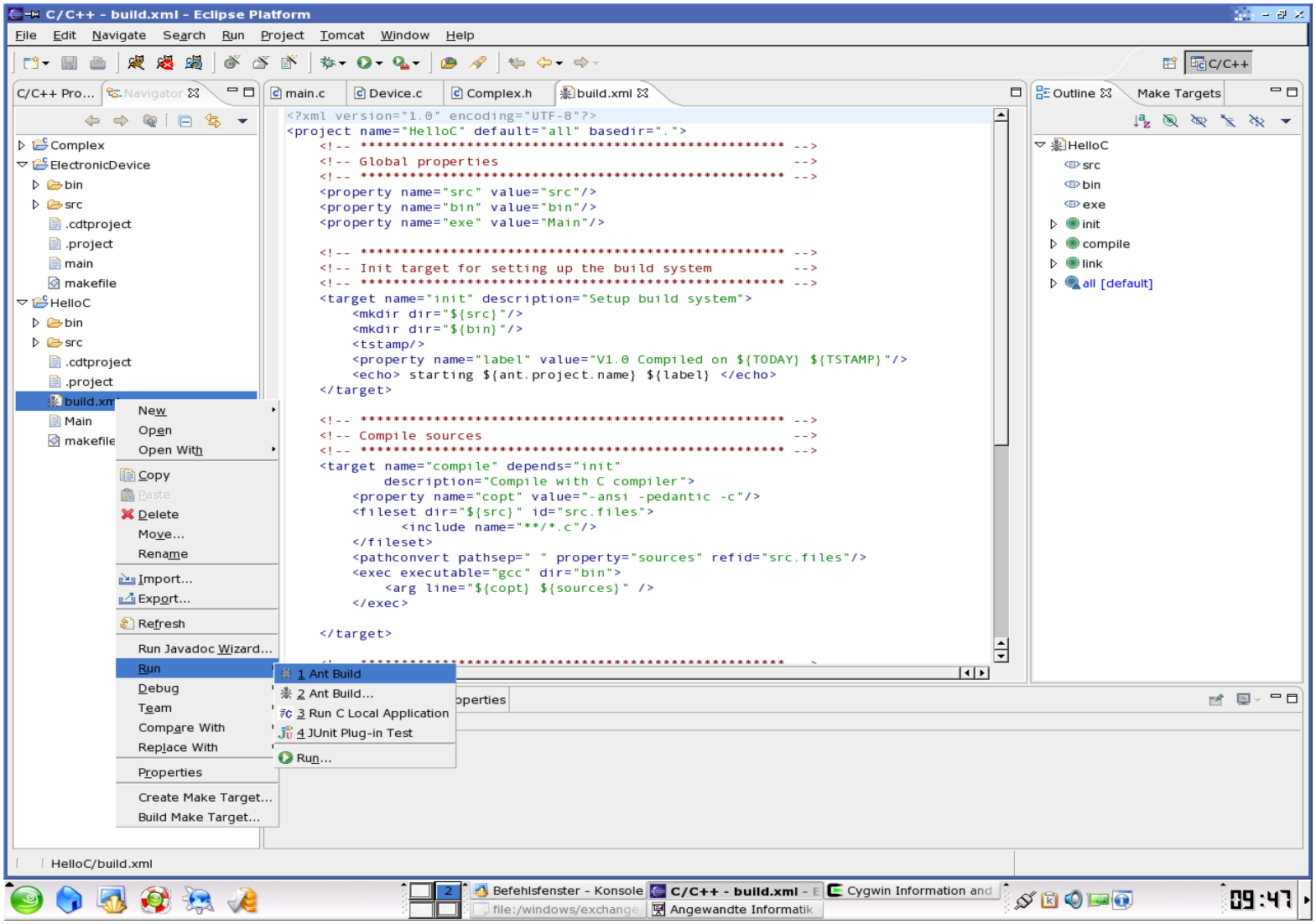


Eine typische Build



- Die einzelnen Schritte einer Build folgen einer Abhängigkeitskette.
- Jeder Schritt erfordert den erfolgreichen Abschluss des Vorhergehenden.
- Diese Abhängigkeitsregeln lassen sich als Graph analog einer Status-maschine darstellen.
- Gängigen Werkzeuge, um solche eine Build zu automatisieren sind in der Java Welt *Ant* oder *Maven* und für C/C++ *Make*.
- *Gradle* basiert auf Groovy und wird in AndroidStudio verwendet.
- Die meisten IDEs können mit diesen Werkzeugen umgehen.

Ant & Build.xml in Eclipse



Ant shell script

```
#!/bin/sh
#
# shell script to invoke ant
# author Nikolaus Wulff
#
MAIN=org.apache.tools.ant.Main
CLASSPATH=
ANT_HOME=/opt/eclipse/plugins/org.apache.ant_1.6.1

for f in $ANT_HOME/lib/*.jar
do
  CLASSPATH=$CLASSPATH:$f
done

java -cp $CLASSPATH $MAIN $*
```

build.xml (1)

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="HelloC" default="all" basedir=".">
    <!-- ********************************************** -->
    <!-- Global properties                              -->
    <!-- ********************************************** -->
    <property name="src" value="src"/>
    <property name="bin" value="bin"/>
    <property name="exe" value="Main"/>

    <!-- ********************************************** -->
    <!-- Init target for setting up the build system    -->
    <!-- ********************************************** -->
    <target name="init" description="Setup build system">
        <mkdir dir="${src}"/>
        <mkdir dir="${bin}"/>
        <tstamp/>
        <property name="label" value="V1.0 Compiled on ${TODAY} ${TSTAMP}"/>
        <echo> starting ${ant.project.name} ${label} </echo>
    </target>
```

- Die Build Schritte werden als XML Dokumente abgespeichert.
- XML Dokumente bilden eine hierarchische Baumstruktur.
- Die einzelenen Markups sind ähnlich ausgezeichnet wie HTML

build.xml (2)

```
<!-- ********************************************** -->
<!-- Compile sources                                -->
<!-- ********************************************** -->
<target name="compile" depends="init" description="Compile with C compiler">
    <property name="copt" value="-ansi -pedantic -c"/>
    <fileset dir="${src}" id="src.files">
         <include name="**/*.c"/>
    </fileset>
    <pathconvert pathsep=" " property="sources" refid="src.files"/>
    <exec executable="gcc" dir="bin">
     <arg line="${copt} ${sources}" />
    </exec>
</target>

<!-- ********************************************** -->
<!-- Link binaries                                  -->
<!-- ********************************************** -->
<target name="link"  depends="compile" description="Link the binaries">
    <fileset dir="${bin}" id="obj.files">
         <include name="**/*.o"/>
    </fileset>
    <pathconvert pathsep=" " property="objs" refid="obj.files"/>
    <exec executable="gcc " dir="bin">
      <arg line="-lm -o ${exe} ${objs}" />
    </exec>
</target>
```

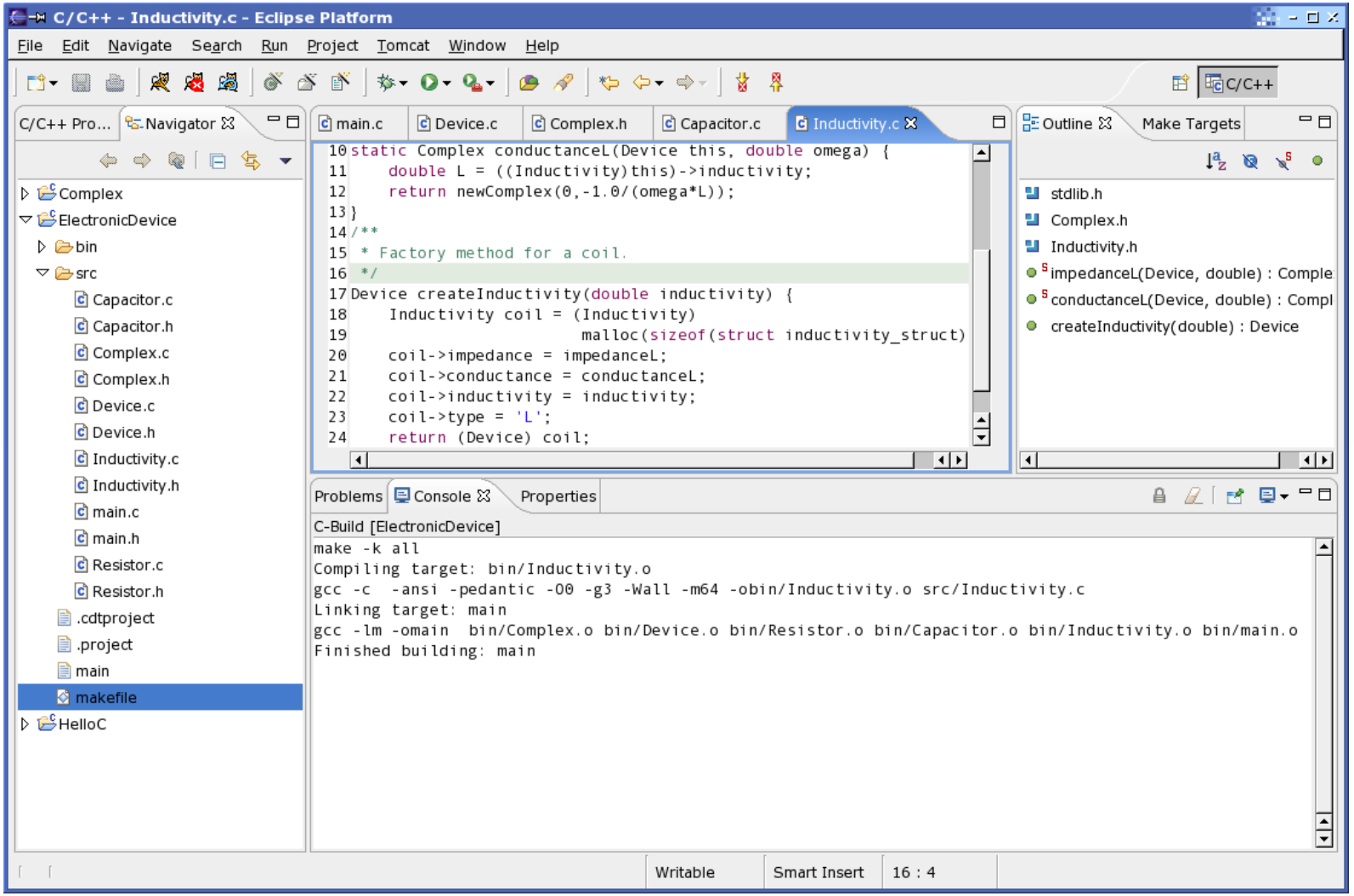
Schritte werden als Targets maskiert...

…und per *depends* verkettet

Make

- Während in Java Projekten Ant das Buildwerkzeug ist wird in C/C++ Projekten meistens das klassische Make-Werkzeug verwendet.
- Make bietet ähnliche Funktionalität wie Ant. Make ist ein Programm, das eine textbasierte make-Datei einliest, interpretiert und die dortigen Anweisungen ausführt.
- Regeln legen fest, wie aus einer C-Quelle die Plattform entsprechenden Binaries erzeugt werden.
- Eclipse führt bei jede C/C++ Sourceänderung im Hintergrund ein make aus.

Makefile in der Eclipse CDT



Ein generisches Makefile

```
## ------------------------------------------------------------
##
##  A generic makefile
##  author Nikolaus Wulff
##
## ------------------------------------------------------------

## ------------------------------------------------------------
## some basic settings for the project
## ------------------------------------------------------------
EXE    :=main
SRCDIR :=src
BINDIR :=bin
## ------------------------------------------------------------
## the basic settings for the plattform specific used tools
## ------------------------------------------------------------

## the compiler to use gcc with compile only flag
CC   :=gcc -c
## compiler flags to use: pedantic ANSI C
CFLAGS := -ansi -pedantic -O0 -g3 -Wall
## the linker to use simply let gcc decide
LINK :=gcc -lm
## libaries to include: so far none
LIBS :=
## removal of files
RM   :=rm
```

Ein generisches Makefile (2)

```
## ------------------------------------------------
## build rule for the executable
## pseudo target all
## ------------------------------------------------
all: $(EXE)

## ------------------------------------------------
## build rule for object binaries
## compile C sources into the bin directory
## ------------------------------------------------
$(BINDIR)/%.o : $(SRCDIR)/%.c $(SRCDIR)/%.h
    @echo 'Compiling target: $@'
    $(CC) $(CFLAGS) -o$@ $<

## ------------------------------------------------
## build rule for the object binaries
## link the binaries to an executable
## ------------------------------------------------
$(EXE): $(OBJ)
    @echo 'Linking target: $@'
    $(LINK) -o$@ $(OBJ) $(LIBS)
    @echo 'Finished building: $@'

## ------------------------------------------------
## build rule for a fresh build
## ------------------------------------------------
clean:
   $(RM) -f $(EXEDIR)/$(EXE) $(BINDIR)/*.*
```

Durch eine Target:Source Beziehung lassen sich Abhängigkeiten abbilden.

Die make Schritte werden durch Regeln beschrieben.

Abhängigkeiten

- Eine Herausforderung in größeren Projekten ist die Abhängigkeit von externen Bibliotheken in unterschiedlichen Versionsständen.
- Zu jeder Build müssen die verwendeten Ressourcen genau protokolliert und versioniert werden.
- Das „Zusammensuchen“ der zu verwendenden Bibliotheken kann sehr fehleranfällig werden.
- Abhilfe schafft Maven, das die transitive Hülle aller Abhängigkeiten auflöst und diese automatisch aus dem Web in der jeweiligen Version lädt.

Maven

- Das Ant Tool wird zunehmend durch das Apache Maven2 Projekt ersetzt.
- Maven bietet die Möglichkeit Abhängigkeiten zu externen Bibliotheken automatisch aufzulösen.
- Ebenso wie bei Ant lassen sich weitere Tools als Plugins in den Build Prozess einbinden.
- Maven generiert auf Wunsch eine komplette Web-Site für das Projekt und erleichtert das Verteilen der Artefakte als Bibliotheken in einem Repository.
- Das Projekt kann dann als „Dependency“ für andere Projekte aufgelöst werden.

POM

- Analog zu Ant verwendet Maven ein XML basiertes Project Object Model (POM).
- Alle Bibliotheken und Tools werden hierdrin in ihrer jeweiligen Version spezifiziert.
- Maven interpretiert die pom.xml Datei und führt die notwendigen Schritte zum Bauen des Projektes aus.
- Projekte können in einer Eltern-Kind-Beziehung voneinander abhängig gemacht werden.
- Maven verwendet zum Bauen Ziele, wie z.B. „clean“, „test“ und „install“ oder „site“.

Versionsmanagement

- Das Fragment zeigt die Plugin-Definition zur Spezifizierung einer Build für das JDK 1.6 mit Optimizer und Debug Einstellung:

```
<plugin>
  <artifactId>maven-compiler-plugin</artifactId>
  <version>2.3.2</version>
  <configuration>
    <debug>false</debug>
    <optimize>true</optimize>
    <source>1.6</source>
    <target>1.6</target>
  </configuration>
</plugin>
```

- Wird die pom.xml Datei versioniert lassen sich alle Bibliotheken und Tools jederzeit restaurieren.

Abhängigkeiten auflösen

- Das Fragment zeigt wie die zwei Tools JUnit und AntLR in die Abhängigkeitskette eingefügt werden.

```
<dependencies>
   <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.2</version>
      <scope>test</scope>
   </dependency>
   <dependency>
     <groupId>org.antlr</groupId>
      <artifactId>antlr-complete</artifactId>
      <version>3.5.1</version>
   </dependency>
```

- Die Toolversionen werden explizit aufgeführt.

Repositories einbinden

- Externe Repositories können explizit in den Build Prozess integriert werden.

```
<repositories>
  <repository>
    <snapshots>
      <enabled>true</enabled>
    </snapshots>
    <id>lab4inf-repository</id>
    <name>Lab4Inf Repository</name>
    <url>http://www.lab4inf.fh-muenster.de/lab4inf/maven-re
  </repository>
```

- Das Maven-Plugin erleichtert die POM Erstellung, so dass die XML Datei nicht per Hand editiert werden muss.

Gradle

- Groovy verwendet seit 2009 Gradle zum Bauen von Projekten. Gradle basiert auf Groovy und verwendet Groovy Skript zum definieren des Build Plans.
- So lassen sich Build Skripte ohne kompliziertes XML direkt in einer Programmiersprache mit einer eigenen DSL erstellen.
- Insbesondere Android Studio setzt auf Gradle und ist seit 2015 die offiziell von Google unterstütze IDE, d.h. das Android Eclipse Plugin wird nicht mehr von Google weiter entwickelt :-(

Ein Gradle Build Skript

```
apply plugin: 'com.android.application'

repositories {
    jcenter()
    flatDir {
        dirs 'prebuilt-libs'
    }
}

android {
    compileSdkVersion 19
    BuildToolsVersion "24.0.3"

    defaultConfig {
        applicationId "com.example.nwulff.myapplication"
        minSdkVersion 19
        targetSdkVersion 24
        versionCode 1
        versionName "1.0"
    }
```

Groovy ist eine Java ähnliche Programmiersprache und läuft auf der JVM. Groovy braucht kein Semikolon um eine Anweisung zu beenden und hat viele nette Eigenschaften die Java (noch) nicht kennt … :-)

Gradle Build Skript cont...

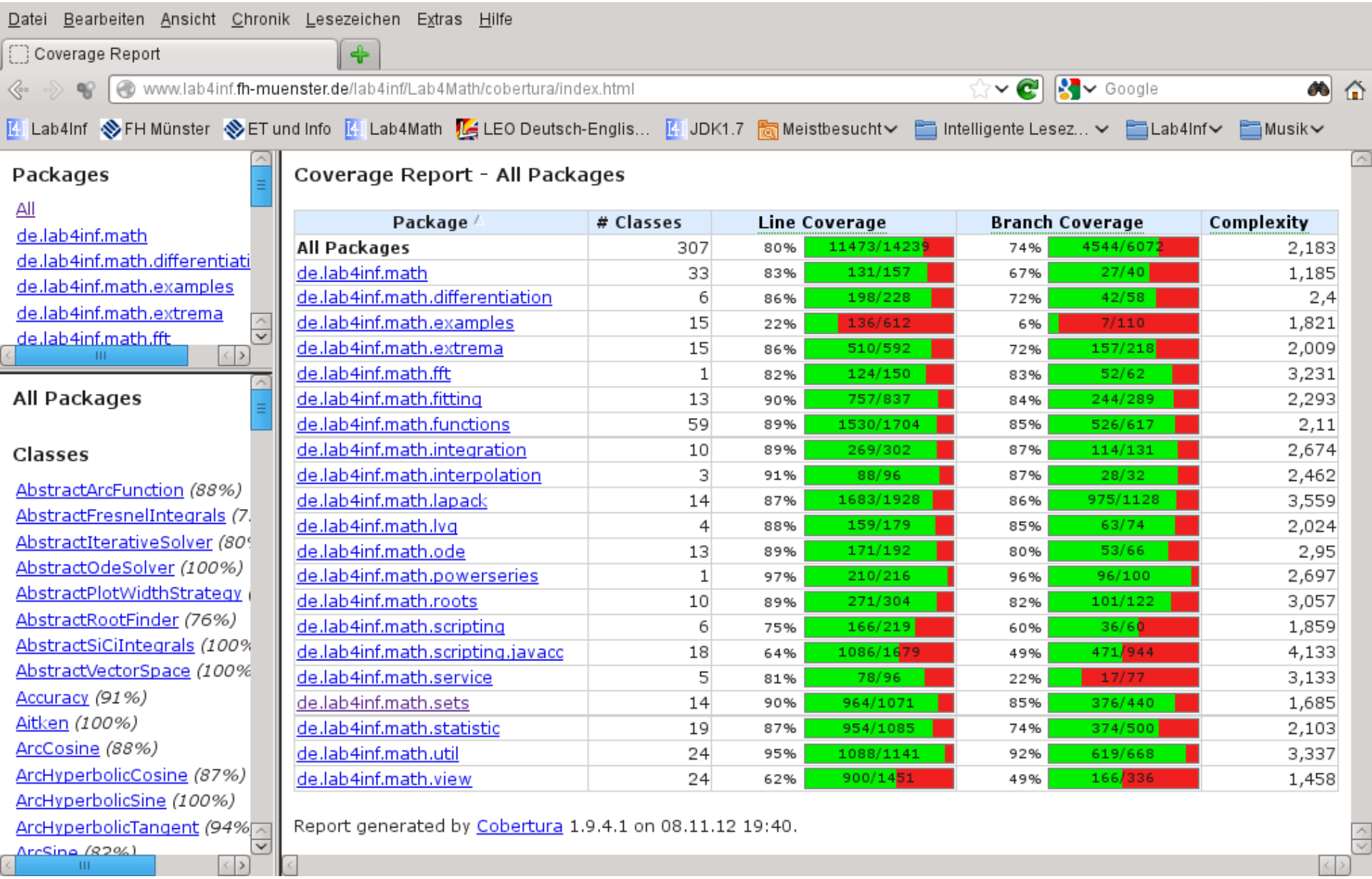
```
android {
    // ….
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_7
        targetCompatibility JavaVersion.VERSION_1_7
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'),
                                                  'proguard-rules.pro'
        }
    }
}

dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
}
```

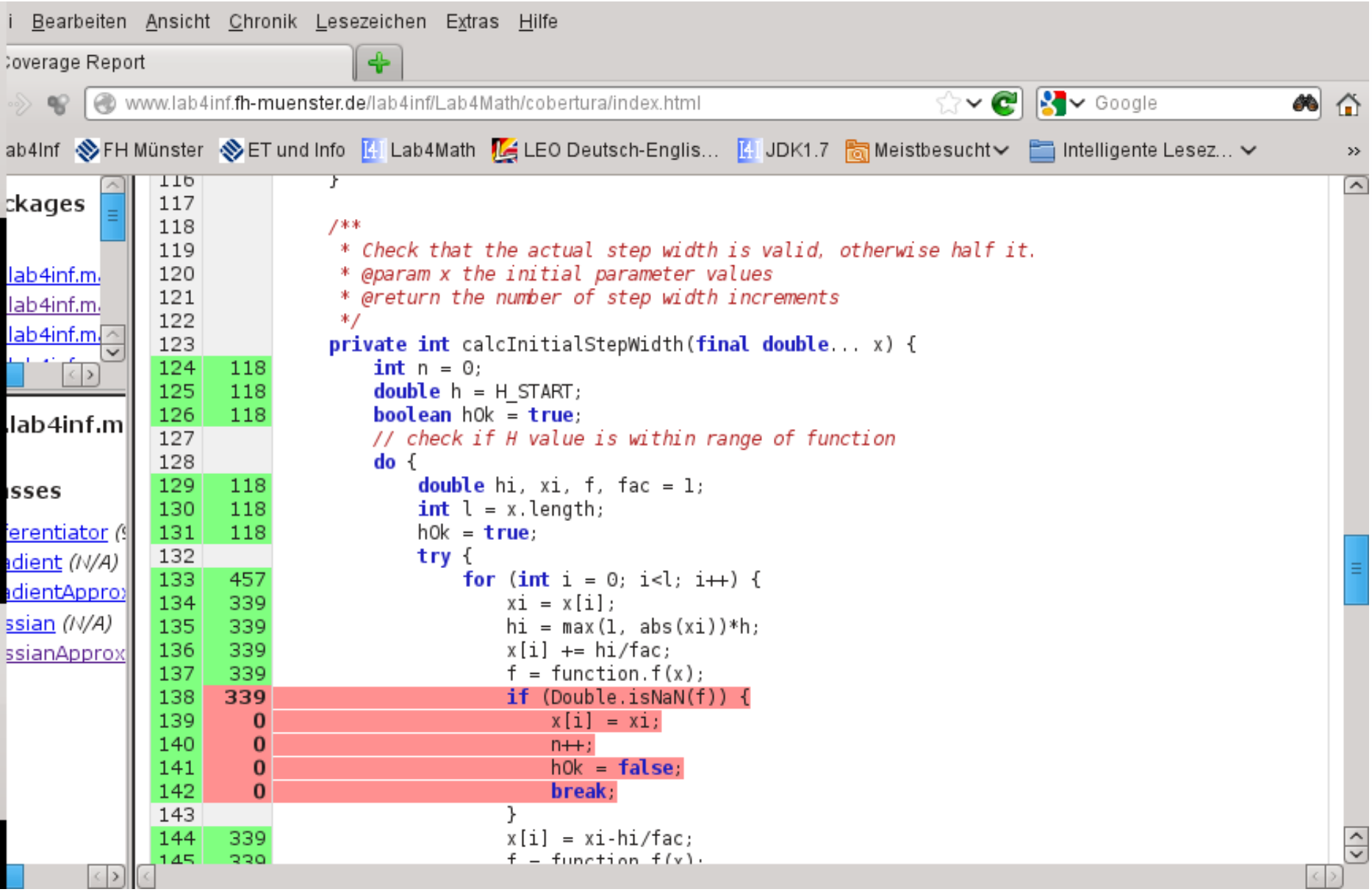
Automatisiertes Testen

- Wichtig ist eine ausreichende automatisierte Test-Suite, die in den Build-Prozess mit integriert ist.
- Meist werden regelmäßig die Testüberdeckung und einige Software-Metriken mit protokolliert.
- Nur so kann sichergestellt werden, dass alle Bibliotheken und Programme in einer gewünschten Qualität erstellt werden.
- Diese Form der Automatisierung erleichtert und beschleunigt – nach einer anfänglichen Lernkurve – das Projektgeschäft und entlastet teilweise die Überwachungsaufgaben der Projektleitung.

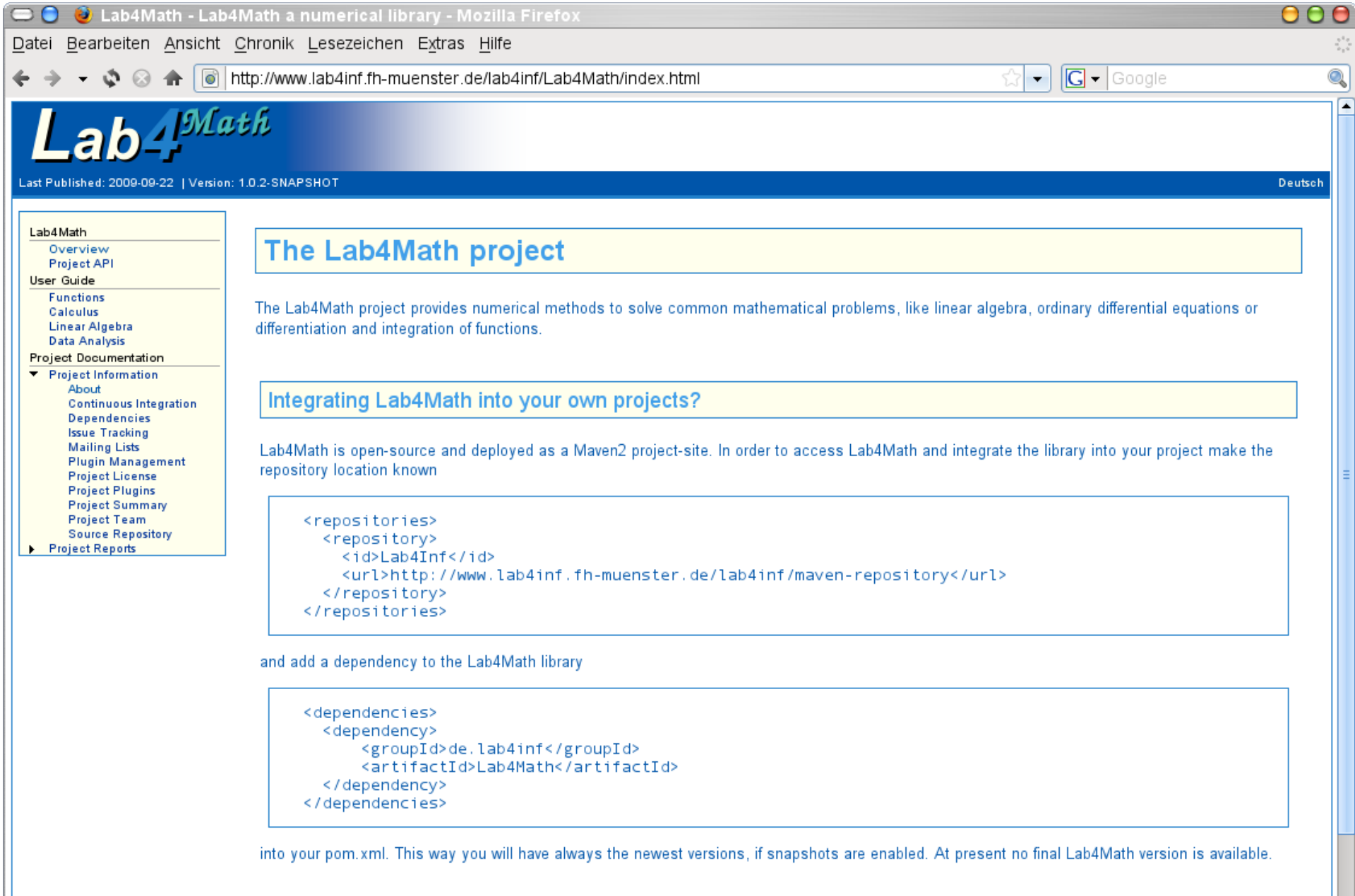
Generierter Coverage Report



Anzeige der Überdeckung



Maven generierte Web-Site



CI/CD

- Werkzeuge wie Ant, Maven & Co werden vom Entwickler innerhalb seiner Entwicklungsumgebung (Eclipse, Netbeans...) während der Entwicklung ausgeführt.
- Nach dem Einchecken in das Versionierungssystem (CVS, SVN, Git...) wird für Continious Integration eine automatische CI/CD Pipline angestoßen, die das komplette Projekt auschecked, neu baut, alle Tests und Berichte erstellt und die fertigen Bibliotheken und Artefakte Skript gesteuert in einem Repository zur Verfügung stellt.

Zusammenfassung

- Ant, Maven und Gradle bieten sehr komfortabel das automatisierte Bauen eines Projekts an.
- Alle Projekt Resourcen gehören unter Versions-verwaltung.
- Für eine „clean Build“ wird das Projekt komplett gelöscht und neu aufgebaut und getestet.
- Die neuen Bibliotheken werden anschließend an einer zentralen Stelle zur Verfügung gestellt.
- Dieses Vorgehen eignet sich insbesondere für das Arbeiten in verteilten Teams.