



Build-Management

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

Prof. Dr. Nikolaus Wulff



- 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 ...



The screenshot displays the Eclipse IDE interface for a C/C++ project named 'Complex'. The main window is titled 'C/C++ - Complex.h - Eclipse Platform'. The menu bar includes 'File', 'Edit', 'Navigate', 'Search', 'Run', 'Project', 'Tomcat', 'Window', and 'Help'. The toolbar contains various icons for file operations and development tools.

The left sidebar shows the 'Project Navigator' with a tree view of the project structure:

- Complex
 - Debug
 - .cdtbuild
 - .cdtproject
 - .project
 - Complex.c
 - Complex.h

The central 'Source Editor' displays the content of '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);

The right sidebar shows the 'Outline' view, which lists the classes and functions defined in the file:

- __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) : c
- cphi(const Complex) : d
- cddeg(const Complex) :
- toString(const Complex

The bottom panel shows the 'Console' view with the following output:<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

Three blue callout boxes highlight key features: 'Projekt Navigation' (Project Navigator), 'Source Editor', and 'Class Structure' (Outline view). The console output is also highlighted with a blue box labeled 'Console stdout'.



- 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



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:

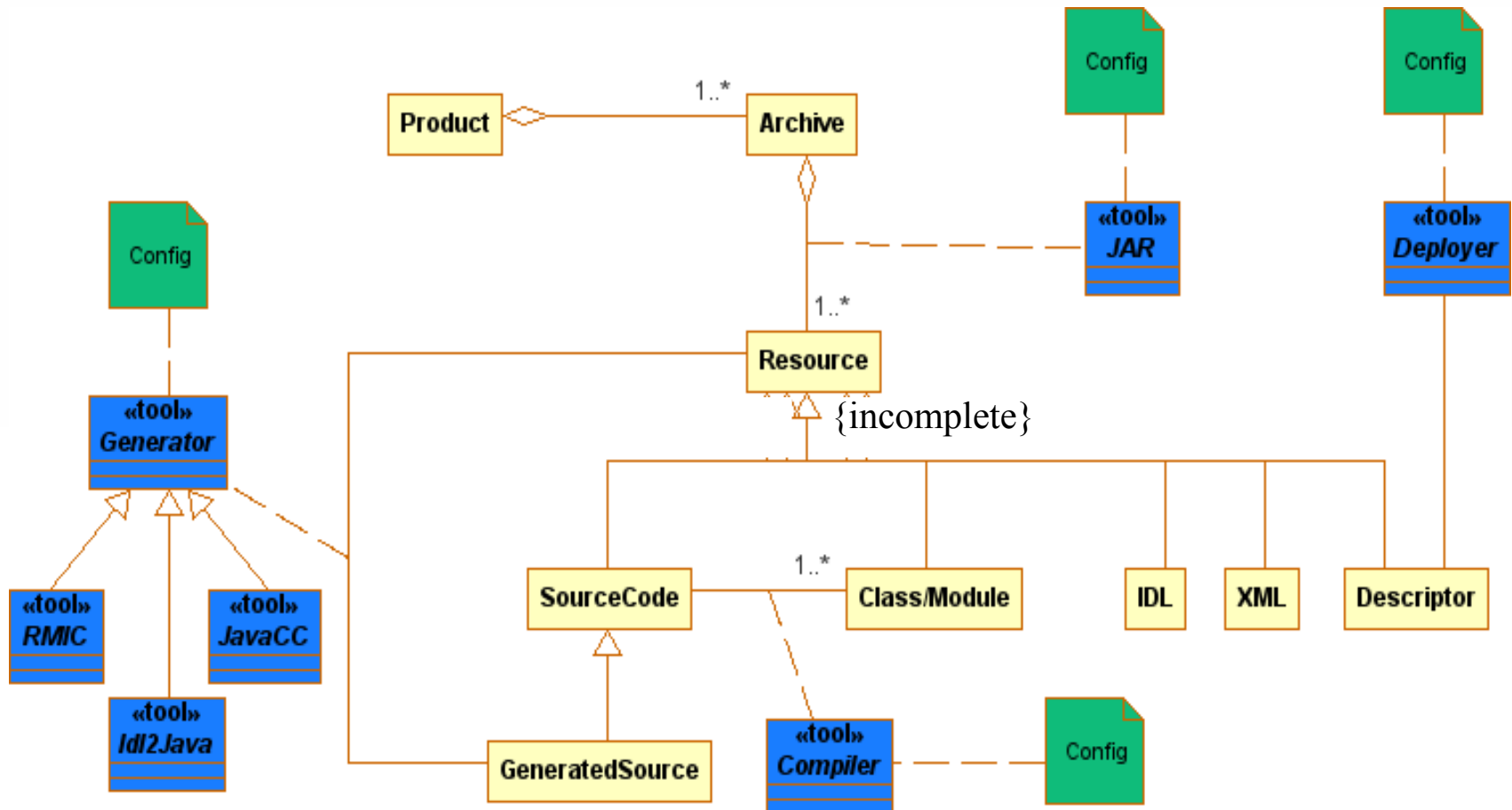
- Version Control
- Build Management
- Configuration Management
- Change Control
- ...



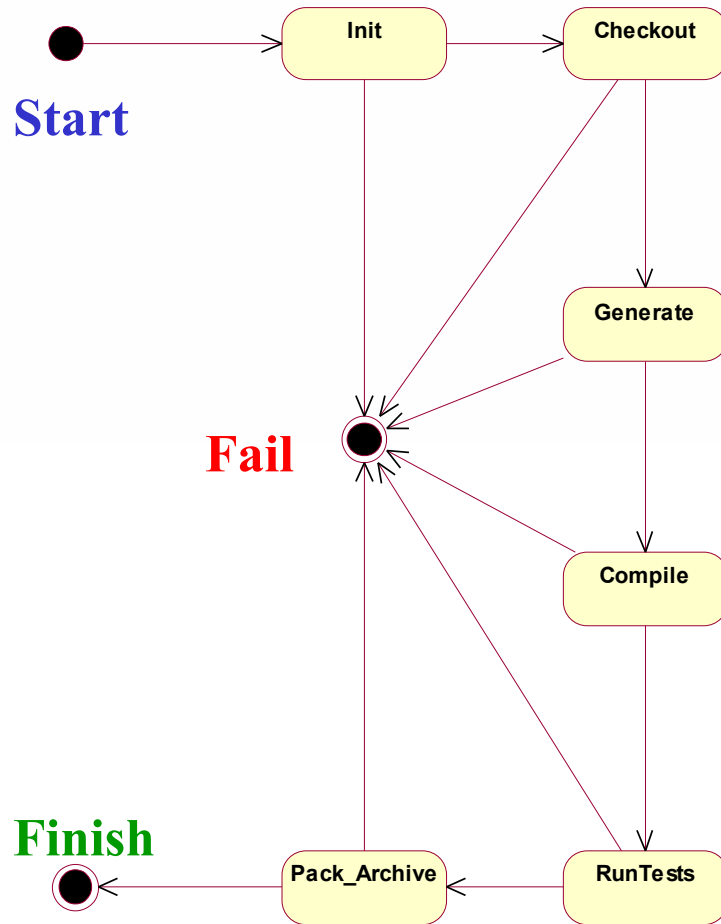
- Arbeitsplatzumgebung und IDE's
- Versionsverwaltung
- Automatische Build mit make oder Apache Ant
- Permanente Integration und Nightly Build
- Automatische Komponenten- und Regressionstests
- Performance- und Systemtest
- Softwarequalität und Metriken



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 Statusmaschine 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



```
<?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>

  <!-- ***** -->
  <!-- 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 -->
  <!-- ***** -->
  <target name="link" depends="compile">
    <exec executable="gcc" dir="bin">
      <arg line="Main.o" />
    </exec>
  </target>

  <!-- ***** -->
  <!-- Run -->
  <!-- ***** -->
  <target name="run" depends="link">
    <exec executable="./Main" />
  </target>
</project>
```

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 einzelnen 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>
```

Schritte werden als
Targets maskiert...

```
<!-- ***** -->
<!-- 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>
```

...und per *depends* verkettet



- 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



The screenshot shows the Eclipse IDE interface for a C/C++ project named 'Inductivity.c'. The main editor displays the source code for the 'Inductivity.c' file, which includes a static function 'conductanceL' and a factory method 'createInductivity'. The 'Outline' view on the right lists the project's structure, including headers and source files. The 'Console' view at the bottom shows the output of a 'make -k all' command, indicating successful compilation and linking of the project.

```
10 static Complex conductanceL(Device this, double omega) {
11     double L = ((Inductivity)this)->inductivity;
12     return newComplex(0, -1.0/(omega*L));
13 }
14 /**
15  * Factory method for a coil.
16  */
17 Device createInductivity(double inductivity) {
18     Inductivity coil = (Inductivity)
19         malloc(sizeof(struct inductivity_struct)
20     coil->impedance = impedancel;
21     coil->conductance = conductancel;
22     coil->inductivity = inductivity;
23     coil->type = 'L';
24     return (Device) coil;

```

Outline:

- stdlib.h
- Complex.h
- Inductivity.h
- impedanceL(Device, double) : Comple
- conductanceL(Device, double) : Compl
- createInductivity(double) : Device

Console:

```
C-Build [ElectronicDevice]
make -k all
Compiling target: bin/Inductivity.o
gcc -c -ansi -pedantic -O0 -g3 -Wall -m64 -obin/Inductivity.o src/Inductivity.c
Linking target: main
gcc -lm -omain bin/Complex.o bin/Device.o bin/Resistor.o bin/Capacitor.o bin/Inductivity.o bin/main.o
Finished building: main

```

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  
## libraries 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.



- 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.



- 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 Website 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.



- 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“.



- 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.



- 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.



- 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-rep
  </repository>
```

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



- 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 unterstützte 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'])  
}
```



- 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



Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

Coverage Report



www.lab4inf.fh-muenster.de/lab4inf/Lab4Math/cobertura/index.html



Google

Lab4Inf FH Münster ET und Info Lab4Math LEO Deutsch-Englis... JDK1.7 Meistbesucht Intelligente Lesez... Lab4Inf Musik

Packages

All
[de.lab4inf.math](#)
[de.lab4inf.math.differentiati](#)
[de.lab4inf.math.examples](#)
[de.lab4inf.math.extrema](#)
[de.lab4inf.math.fft](#)

All Packages

Classes

[AbstractArcFunction](#) (88%)
[AbstractFresnelIntegrals](#) (7...
[AbstractIterativeSolver](#) (80...
[AbstractOdeSolver](#) (100%)
[AbstractPlotWidthStrategy](#) (...
[AbstractRootFinder](#) (76%)
[AbstractSiCiIntegrals](#) (100%)
[AbstractVectorSpace](#) (100%)
[Accuracy](#) (91%)
[Aitken](#) (100%)
[ArcCosine](#) (88%)
[ArcHyperbolicCosine](#) (87%)
[ArcHyperbolicSine](#) (100%)
[ArcHyperbolicTangent](#) (94%)
[ArcSine](#) (82%)

Coverage Report - All Packages

Package ^	# Classes	Line Coverage	Branch Coverage	Complexity
All Packages	307	80%	74%	2,183
de.lab4inf.math	33	83%	67%	1,185
de.lab4inf.math.differentiation	6	86%	72%	2,4
de.lab4inf.math.examples	15	22%	6%	1,821
de.lab4inf.math.extrema	15	86%	72%	2,009
de.lab4inf.math.fft	1	82%	83%	3,231
de.lab4inf.math.fitting	13	90%	84%	2,293
de.lab4inf.math.functions	59	89%	85%	2,11
de.lab4inf.math.integration	10	89%	87%	2,674
de.lab4inf.math.interpolation	3	91%	87%	2,462
de.lab4inf.math.lapack	14	87%	86%	3,559
de.lab4inf.math.lvg	4	88%	85%	2,024
de.lab4inf.math.ode	13	89%	80%	2,95
de.lab4inf.math.powerseries	1	97%	96%	2,697
de.lab4inf.math.roots	10	89%	82%	3,057
de.lab4inf.math.scripting	6	75%	60%	1,859
de.lab4inf.math.scripting.javacc	18	64%	49%	4,133
de.lab4inf.math.service	5	81%	22%	3,133
de.lab4inf.math.sets	14	90%	85%	1,685
de.lab4inf.math.statistic	19	87%	74%	2,103
de.lab4inf.math.util	24	95%	92%	3,337
de.lab4inf.math.view	24	62%	49%	1,458

Report generated by [Cobertura](#) 1.9.4.1 on 08.11.12 19:40.

Anzeige der Überdeckung



Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

Coverage Report

www.lab4inf.fh-muenster.de/lab4inf/Lab4Math/cobertura/index.html

lab4Inf FH Münster ET und Info Lab4Math LEO Deutsch-Englis... JDK1.7 Meistbesucht Intelligente Lesez...

Package	Line	Count	Code
	116		}
	117		
	118		/**
	119		* Check that the actual step width is valid, otherwise half it.
	120		* @param x the initial parameter values
	121		* @return the number of step width increments
	122		*/
	123		private int calcInitialStepWidth(final double... x) {
	124	118	int n = 0;
	125	118	double h = H_START;
	126	118	boolean hOk = true;
	127		// check if H value is within range of function
	128		do {
	129	118	double hi, xi, f, fac = 1;
	130	118	int l = x.length;
	131	118	hOk = true;
	132		try {
	133	457	for (int i = 0; i < l; i++) {
	134	339	xi = x[i];
	135	339	hi = max(1, abs(xi))*h;
	136	339	x[i] += hi/fac;
	137	339	f = function.f(x);
	138	339	if (Double.isNaN(f)) {
	139	0	x[i] = xi;
	140	0	n++;
	141	0	hOk = false;
	142	0	break;
	143		}
	144	339	x[i] = xi-hi/fac;
	145	339	f = function.f(x);

Maven generierte Web-Site



Lab4Math - Lab4Math a numerical library - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

http://www.lab4inf.fh-muenster.de/lab4inf/Lab4Math/index.html

Google

Lab4Math

Last Published: 2009-09-22 | Version: 1.0.2-SNAPSHOT Deutsch

- Lab4Math
 - Overview
 - Project API
- User Guide
 - Functions
 - Calculus
 - Linear Algebra
 - Data Analysis
- Project Documentation
 - Project Information
 - About
 - Continuous Integration
 - Dependencies
 - Issue Tracking
 - Mailing Lists
 - Plugin Management
 - Project License
 - Project Plugins
 - Project Summary
 - Project Team
 - Source Repository
 - Project Reports

The Lab4Math project

The Lab4Math project provides numerical methods to solve common mathematical problems, like linear algebra, ordinary differential equations or differentiation and integration of functions.

Integrating Lab4Math into your own projects?

Lab4Math is open-source and deployed as a Maven2 project-site. In order to access Lab4Math and integrate the library into your project make the repository location known

```
<repositories>
  <repository>
    <id>Lab4Inf</id>
    <url>http://www.lab4inf.fh-muenster.de/lab4inf/maven-repository</url>
  </repository>
</repositories>
```

and add a dependency to the Lab4Math library

```
<dependencies>
  <dependency>
    <groupId>de.lab4inf</groupId>
    <artifactId>Lab4Math</artifactId>
  </dependency>
</dependencies>
```

into your pom.xml. This way you will have always the newest versions, if snapshots are enabled. At present no final Lab4Math version is available.



- 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 Continuous Integration eine automatische CI/CD Pipeline 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.



- Ant, Maven und Gradle bieten sehr komfortabel das automatisierte Bauen eines Projekts an.
- Alle Projekt Ressourcen gehören unter Versionsverwaltung.
- 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.