TU-Ilmenau

Fakultät für Informatik und Automatisierung

Praktische Informatik / Telematik

Hauptseminar

Ant - das Java Build-Tool

Funktionalität, Mächtigkeit und Praxiserfahrungen

Christoph Lühr, Matr.Nr.: 24714

Betreuer: Dipl. Inf. Thorsten Strufe

Inhaltsverzeichnis

Build-Tools 3

1.1 Definition 3

1.2 Herausforderungen für Java Build-Tools 4

2 Ant 5

2.1 Übersicht 5

2.2 Arbeitsweise 5

2.3 Ausgewählte Ant Tasks 7

2.3.1 Dateisystemoperationen 7

2.3.2 Interoperabilität 7

2.3.3 Archivfunktionen 7

2.3.4 Parallele Build Vorgänge 8

2.3.5 Netzwerk-Funktionalitäten 8

2.3.6 Software Configuration Management 8

2.3.7 XML / Dokumentation 9

2.3.8 Automatisierte Tests 9

2.4 Erweiterung von Ant 10

2.4.1 Entwicklung eigener Tasks 10

2.4.2 Anbindung an integrierte Entwicklungsumgebungen 10

3 Java Build Alternativen 12

3.1 make – Der Standart unter den Build-Tools 12

3.2 jmk – Java make 13

4 Fazit: make, jmk & Ant im Vergleich 15

5 Anhang 17

5.1 Sourcecode / Beispiele 17

5.1.1 Beispiel build.xml 17

5.2 Quellenverzeichnis 18

Build-Tools

1.1 Definition

Build-Tools unterstützen und begleiten den Prozeß der Softwareentwicklung.
Projekte in der Softwareentwicklung, sofern es sich nicht um trivialste Systeme handelt, durchlaufen im Entwicklungszyklus eine Reihe von unterschiedlichen Phasen.
Es existieren in der Theorie unterschiedlichen Definitionen und Modelle vom Ablauf dieser Phasen, wobei sich jedoch bei allen Modellen folgende Kernbereiche bzw. -aufgaben identifizieren lassen:

1. Spezifikation
2. Implementation
3. Integration
4. Test
5. Systembetrieb (Deployment)

In der Spezifikationsphase wird in der Regel der Funktionsumfang und die Anforderungen an ein System festgelegt und beschrieben.
Die Implementation umfaßt das eigentliche Erstellen des Quelltexte und Module durch die Entwickler.
Diese einzelnen Komponenten werden in der Integrationsphase zu dem Gesamtsystem verbunden und in der Testphase auf die Einhaltung der Spezifikation überprüft.
Typischerweise werden diese Schritte bis zur entgültigen Version eines Systems mehrfach durchlaufen.
Unter dem Deployment versteht man die Überführung eines neuen System in den Produktionsbetrieb. Dies ist eine Aufgabe, die insbesondere im Bereich der Web-Applikationen häufig durchgeführt werden muss.

Wenn möglichst viele dieser Phasen vollständig durch ein Build-Tool unterstützt werden, ergibt sich eine Reihe von Vorteilen.
Die wichtigste Aufgabe ist die Kontrolle und die Durchführung der Integrationsphase. Wünschenswert ist meist auch eine plattformübergreifende Einsetzbarkeit, um unterschiedliche Umgebungen unterstützen zu können. Weiterhin erfolgt neben der Arbeitserleichterung durch die Automatisierung der Prozesse eine Standardisierung des

Softwareentwicklungs- und Build-Vorgangs, was zu einer größeren Transparenz des kompletten Ablaufs führt. Der Entwicklungsprozeß wird dokumentiert und ist jederzeit für alle Projektbeteiligten nachvollziehbar und konsistent.

.

1.2 Herausforderungen für Java Build-Tools

Einer der wichtigsten Eigenschaften von Java als Implementationsprache und System ist eine plattformübergreifende Einsatzmöglichkeit. Hieraus ergeben sich Anforderungen an das Build-Tool, welches ebenfalls auf den benötigten Betriebssystemumgebungen verfügbar sein muss. Unterschiedliche Umgebungen haben möglicherweise auch eine unterschiedliche Syntax für die Dateisystemauszeichnung oder andere Differenzen.

Weitere Probleme können sich ergeben, wenn das Build-Tool auf externe Systeme oder Programme angewiesen ist, da hier ebenfalls Inkonsistenzen zwischen verschiedenen Implementationen auftreten können.

Die Leistung und Geschwindigkeit von Build-Tools stellt auch einen nicht zu vernachlässigenden Faktor dar, wenn umfangreiche Projekte einen kurzen Integrationszyklus haben und oft einen kompletten Build, z.B. zu Testzwecken, benötigen.

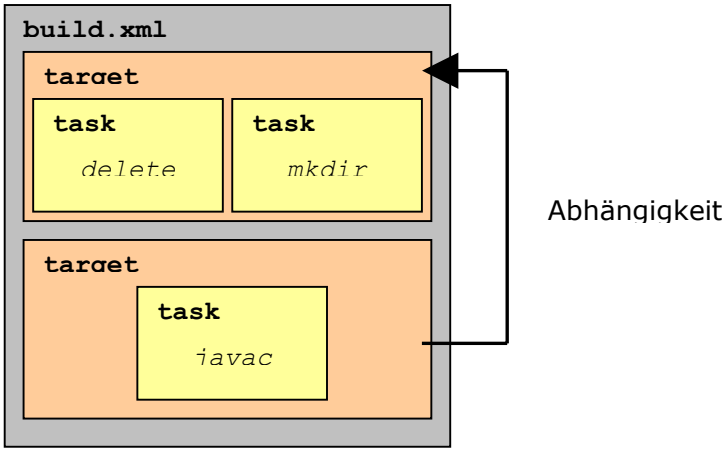
2 Ant

2.1 Übersicht

Ant ist ein Projekt der Apache Software Foundation[1], das im Rahmen des Jakarta Projektes gestartet wurde. Jakarta umfaßt eine Reihe von Softwaresystemen die einen starken Bezug zu Java und dem WWW haben. Ant entstand aus dem Bedürfnis nach einem plattformübergreifenden Build-Tool, welches die Probleme der bisherigen Lösungen durch eine native Implementation in Java umgeht. Weiterhin ist bei Ant die Erstellung der Build-Anweisungsskripte aufgrund der konsequenten Verwendung von XML als Beschreibungssprache schnell erlernbar und einfach einzusetzen. Dieser Umstand vereinfacht auch die Anbindung an integrierte Entwicklungsumgebungen (IDE). Die Liste der Ant unterstützenden IDEs umfaßt neben jBuilder, NetBeans und VisualAage for Java auch eine große Anzahl von populären Editoren.
Die Software ist frei verfügbar und steht unter einer Open Source Lizenz.

2.2 Arbeitsweise

Die Konfiguration eines Ant Build-Prozesses erfolgt durch eine zentrale XML Datei.



Verschiedene Prozesse und Aktionen werden über Targets (Ziele) innerhalb der build.xml Datei spezifiziert. Für jedes Target kann eine Sequenz von Tasks (Aufgaben) festgelegt werden, nach deren vollständigen Abarbeitung das entsprechende Ziel erreicht ist. Weiterhin lassen sich Abhängigkeiten zwischen verschiedenen Targets festlegen, um

[1] Die Apache Software Foundation ist ein non-profit Unternehmen, welches 1995 zum Zweck der Weiterentwicklung des Open Source Web-Servers Apache in den USA gegründet wurde und heute eine große Zahl von Open Source Projekten unterstützt

bestimmte Abfolgen im Build-Prozess zu koordinieren- hierbei stellt Ant auch sicher, daß ein Target pro Durchlauf nie mehrfach ausgeführt werden kann.

```
<project name="DemoProjekt" default="deploy" basedir=".">
  <target name="deploy" depends="compile,init">
    <copy file="${jsp}" todir="${deployDir}"/>
  </target>
</project>
```

Bei dem Start des Build-Vorgangs über den Aufruf von Ant in der Kommandozeile, lassen sich einzelne Ziele spezifieren, deren Ablauf gewünscht wird. So könnte z.B. die Eingabe von

```
> ant compile
```

den Kompilierungsvorgang einer bestimmten Software starten, während

```
> ant deploy
```

den aktuellen Build in das Produktionssystem übernimmt (Deployment).
Jeder Task ist als eigenständige Java Klasse implementiert und besitzt eine eigene „Intelligenz“, welche sicherstellt, dass Aktionen auch nur bei Bedarf ausgeführt werden. Beispielweise erkennt der Java-Compiler-Task `javac`, ob bereits eine *.class Datei vorliegt, oder ob aufgrund von Änderungen an dem entsprechenden Ursprungs- *.java Quelltext ein erneuter Kompilierungslauf notwendig ist.
Ant enthält eine große Anzahl von Tasks / Klassen, die bereits ein breites Spektrum der bei Build-Prozessen notwendigen Aktionen abdecken. Werden zusätzliche Operationen benötigt, so lassen sich einfach neue Tasks definieren, zu Ant hinzufügen und einsetzen.

2.3 Ausgewählte Ant Tasks

2.3.1 Dateisystemoperationen

Ant unterstützt nahezu alle wichtigen Dateioperationen durch integrierte Tasks, hierdurch werden betriebssystemspezifische Eigenheiten gekapselt.

- `Chmod`
- `Copy`
- `Delete`
- `MkDir`
- `Move`

2.3.2 Interoperabilität

Die Erweiterbarkeit von Ant ist nicht nur auf die Implementation von Tasks als neue Klassen beschränkt. Weiterhin ist die Integration von externen, möglicherweise plattformspezifischen Werkzeugen über den `Exec` Task möglich. Da bei der Verwendung von betriebsystemspezifischen Tools immer lokale Beschränkungen z.B. des Dateisystems existieren, stellt Ant auch Tasks zur Konvertierung von Pfad- und Dateiangaben zur Verfügung (`Pathconvert`).

2.3.3 Archivfunktionen

Neben der eigentlichen Integrationsphase ist es auch im Bereich des Systembetriebs und Deployments häufig notwendig, vor kritischen Aktionen Sicherheitskopien anzulegen oder gegebenfalls alte Archive erneut einzuspielen. Hierzu stellt Ant auch eine Reihe von Tasks für unterschiedliche Paketformate zur Verfügung:

- `Unzip`
- `Gzip`
- `Jar`
- `UnJar`
- `SignJar`
- `Tar`
- `Untar`
- `Zip`
- `Unzip`

2.3.4 Parallele Build Vorgänge

Ant erlaubt die Schachtelung von Tasks mittels `parallel` und `sequential` Bereiche. Hierdurch lässt sich unkompliziert eine explizite Steuerung von Aktionsgruppen bezüglich des jeweils benötigten konkurrenten Ablaufsverhaltens erreichen. Die Abarbeitung von Tasks, die direkt innerhalb von `parallel` Blöcken aufgeführt werden, erfolgt „gleichzeitig“ (i.d.R. Task-Switching durch den jeweiligen Betriebssystemkern). Wenn innerhalb der `parallel` Strukturen hingegen jedoch auch Gruppen von Tasks wieder eine definierte Reihenfolge der Bearbeitung erfordern, so lässt sich dies durch `sequential`-Blöcke erreichen.

```
<parallel>
      <startserver ...>
      <sequential>
            <testserver ...>
            <stopserver/>
      </sequential>
</parallel>
```

Über diesen Mechanismus lassen sich z.B. einfach Geschwindigkeitsgewinne bei Build-Vorgängen auf Multiprozessormaschinen realisieren.

2.3.5 Netzwerk-Funktionalitäten

Insbesondere für den Deployment-Bereich finden sich im Repertoire der bereits in Ant enthaltenen Tasks eine Reihe von interessanten Funktionen. So ist es beispielsweise eine automatische Verteilung von Klassen auf multiple Server über die Verwendung von FTP und HTTP Tasks unkompliziert und plattformübergreifend zu konfigurieren. Auch die Benachrichtigung über regelmäßige, nicht direkt überwachte Build-Vorgänge oder andere Aufgaben mit ähnlichen Anforderungen ist z.B. mittels eines eMail-Tasks möglich.

2.3.6 Software Configuration Management

Kein umfangreiches Softwareprojekt, an dem mehrere Entwickler, möglichweise sogar an unterschiedlichen Standorten zusammen entwickeln, kommt ohne ein umfassendes Software Configuration Management (SCM) aus. Zielsetzung ist die Nachvollziehbarkeit des Entwicklungsprozesses durch umfangreiche Versionskontrolle aller Module. Ant unterstützt neben dem PVCS System eines der meistverbreiteten SMC Systeme, das Open Source Concurrent Versioning System. Über die vordefinierten `CVS` und `CVSPass` Tasks

können so durch Ant z.B. automatisiert Updates und kontrolliertes Deployment direkt aus CVS-Repositories erfolgen.
Einfachere Aufgaben lassen sich auch mittels des `Patch` Tasks lösen, über den Differenzdateien eingespielt und Änderungen an Dateien durchgeführt werden können.

2.3.7 XML / Dokumentation

XML entwickelt sich nicht nur im Web-Bereich zu einem universellen Datenaustausch und Beschreibungsformat, sondern wird auch in vielen anderen Einsatzgebieten immer häufiger verwendet. Ant unterstützt den Entwicklungsprozeß mit XML durch einen XML Validator-Task, der die Überprüfung von XML Daten ermöglicht.

2.3.8 Automatisierte Tests

Eine Tätigkeit die in der Softwareentwicklung oft vernachlässigt wird, ist die Durchführung von Funktionsüberprüfungen. Viele dieser Tests lassen sich automatisieren und fördern so eine höhere Stabilität der Applikationsysteme. JUnit ist ein umfangreiches Framework zur Erstellung und Evaluation von Test-Suites, welches direkt von Ant über die Junit Tasks eingebunden werden kann.

2.4 Erweiterung von Ant

2.4.1 Entwicklung eigener Tasks

Aufgaben, die sich noch nicht über die integrierten Tasks von Ant lösen lassen, können über eigene Tasks bearbeitet werden. Hier beispielhaft ein Aufruf eines neu erstellten Tasks:

```
<coffee extra="sugar"/>
```

Jeder Task hat einen eindeutigen Bezeichner und eine bliebige Menge von Parametern. Um einen neuen Task zu erstellen, ist es zunächst erforderlich, eine Klasse zu implementieren, die das Interface `org.apache.tools.ant.task` implementiert. In dieser Klasse muss dann für jeden möglichen Parameter eine `setXXX()` Methode definiert werden. Zur Auslösung der eigentlichen Aktion muss weiterhin genau eine `execute()` Methode erstellt werden.

```
makeCoffee extends org.apache.tools.ant.Task ...

setExtra()...
execute() ..
```

Die Integration in ein `build.xml` für dieses Beispiel könnte dann folgendermaßen erreicht werden:

```
<target name="init">
  <taskdef name="coffee" classname="local.makeCoffee"/>
  <coffee extra="sugar, cream"/>
</target>
```

2.4.2 Anbindung an integrierte Entwicklungsumgebungen

Eine weitere Eigenschaft von Ant ist die Möglichkeit, während des Build-Vorgangs Informationen an andere externe Prozesse zu senden. Hierzu existiert das `BuildListener` Interface, welches durch eine entsprechende Klasse implementiert werden muss. Sofern diese Klasse eine oder mehrere der folgenden Methoden definiert, werden diese zu den entsprechenden Ereignissen mit einem `BuildEvent` Objekt als

Parameter aufgerufen. Über das `BuildEvent` Objekt lassen sich dann genauere Informationen über den aktuellen Status des Build-Prozesses ermitteln.

`BuildListener` Methoden:

- `buildStarted(BuildEvent)`
- `buildFinished(BuildEvent)`
- `targetdStarted(BuildEvent)`
- `targetFinished(BuildEvent)`
- `taskStarted(BuildEvent)`
- `taskFinished(BuildEvent)`

Um einen deratigen Listener an Ant anzubinden, existiert eine Kommandozeilenoption, die als Parameter die entsprechende listener-Klasse erwartet.

```
> ant -listener org.apache.tools.ant.XmlLogger
```

3 Java Build Alternativen

Es existieren eine Reihe von verschiedenen Werkzeugen zur Unterstützung von Build-und Entwicklungsprozessen.

3.1 make – Der Standart unter den Build-Tools

Seit den ersten Unix Implementationen hat sich make als Standart für C und C++ basierte Projekte bewährt und wird auch heute noch für viele Projekte eingesetzt.
Make ist in vielen Portierungen für fast alle gängigen Betriebsysteme verfügbar.
Ziel von make ist es, die Übersetzung des Quellcodes zu steuern und unnötige Kompilationen zu vermeiden, um so einen möglichst schnellen Übersetzungslauf zu gewährleisten. Die Steuerung des make Build-Prozesses erfolg über zentrale Makefiles, die bestimmte Ziele, Abhängigkeiten und Arbeitsanweisungen zum Build enthalten. Diese Makefiles haben eine mächtige, allerdings auch sehr kryptische Syntax, die insbesondere auf der Signifikanz von nicht-druckbaren Zeichen (Tabs) und kurzen Sonderzeichensequenzen ($*, $^, ..) beruht.

```
srcdir=.

JFLAGS=-O
JAVA=part1.java part2.java part3.java

all build: testprog

testprog: $(JAVA)
 <TAB>  javac $(JFLAGS) $@

clean:
 <TAB>  rm -f $(JAVA)
 <TAB>  mkdir $(srcdir)/classes
```

Weiteres Schlüsselement des Designs ist, daß nur die grundlegensten Funktionen, wie z.B. Variablendefinitionen direkt in make integriert sind. Funktionalitäten, die über diese einfache Fähigkeiten hinausgehen, werden in der Regel über andere externe Programme oder Shell-Skripte realisiert. In homogenen Umgebungen ist dies meist kein Problem, jedoch treten bei der plattformübergreifenden Verwendung von make schnell Probleme,

insbesondere im Bereich der Dateipfadtransformation auf. Daneben ist auch die Verfügbarkeit und Kompatibilität von externen Werkzeugen auf unterschiedlichen Plattformen oft ein Hindernis bei dem Einsatz von make.
Aufgrund der Tatsache, daß alle Aktionen über externe Prozesse durchgeführt werden müssen, ist make insbesondere für Java nur bedingt geeignet, da hier für jeden Kompilationsvorgang eine neue Virtuelle Machine (VM) gestartet werden muss, was durch den hierbei enstehenden Overhead Nachteile gegenüber Java integrierenden Lösungen beinhaltet.

3.2 jmk – Java make

Java make (jmk) entstand ähnlich wie Ant aus dem Bedürfnis heraus, die Unzulänglichkeiten des traditionellen make-Tools zu beseitigen.
Jmk ist komplett in Java implementiert und benötigt zur Ausführung lediglich mindestens eine Java VM 1.2 kompatible Laufzeitumgebung. Da jmk die wichtigsten Dateioperationen integriert hat, werden für gwöhnliche Build-Aufgaben keine externen Werkzeuge benötigt, was die Portabilität von jmk wesentlich erhöht.
Die Build-Beschreibungsprache ist stark an die des standart-make angelehnt, so daß nur ein geringer Lernaufwand und eine einfache Bedienbarkeit gewährleistet wird.

```
javaflags = "-O";

srcdir = ".";
srcs = (glob (join srcdir, "/*.java"));

testprog:    srcs;
{
  exec "javac" javaflags ?;
}

"clean":;
{
  delete (join srcdir, "/*.class");
}
```

Neben den Filesystemoperationen besitzt jmk weiterhin einfache Konstrukte zur Ablaufsteuerung, wie z.B. Schleifen und einfache konditionale Ausdrücke. Jmk ist über externe Klassen erweiterbar und kann so, ähnlich wie Ant, einfach plattformübergreifend an verschiedenste Anforderungen angepasst werden. Darüberhinaus lassen sich auch einfache Funktionen direkt in der jmk Syntax definieren, was die Handhabung von jmk nicht unerheblich verbessert. Größtes Manko dieses Build-Tools ist, daß der Aufrufe des javac compilers nicht innerhalb der selben Virtual Machine stattfinden – daher müssen hier Performancenachteile aufgrund der wiederholten Initialisierung der Java Systeme in Kauf genommen werden.

4 Fazit: make, jmk & Ant im Vergleich

Bei der Auswahl eines Werkzeugs zur Unterstützung des Softwarentwicklungsprozesses eines Projekts muß zunächst eine genaue Untersuchung des Einsatzbereiches stattfinden. Je nach Anforderungsprofil können sich folgende Bereiche verschieden stark auf die Bewertung auswirken:

- plattformübergreifende Verfügbarkeit
- einfache Bedienung / Syntax der Build-Beschreibung
- Umfang der Integrierten Funktionen
- Erweiterbarkeit
- Leistungsfähigkeit / Geschwindigkeit

In allen diesen Bereichen kann Ant viele positive Eigenschaften aufweisen. Durch den durchgehenden Einsatz von Java hat auch Ant den Vorteil, auf jedem System lauffähig zu sein, für welches es eine geeignete Implementation der Virtuellen Maschine gibt. Auch jmk ist in Java entwickelt und teilt diese Eigenschaft. make ist zwar in vielen Portierungen für die unterschiedlichsten Systeme verfügbar, trotzdem gibt es jedoch oft Kompatibilitätsprobleme zwischen den verschiedenen Varianten.
Die Syntax von make und jmk ist in großen Teilen ähnlich und ist bei vielen Entwicklern bekannt, oft gibt es allerdings Probleme dadurch, daß die Syntax unintuitiv und kryptisch ist. Ant setzt im Bereich der Buildbeschreibung auf ein XML Format, welches leicht zu Erlernen und neben Entwicklern auch durch Programmsysteme leicht zu verarbeiten ist.
Das breiteste Spektrum der integrierten Funktionalitäten wird von Ant zur Verfügung gestellt. Make und jmk haben keine, bzw. nur grundlegende Dateioperationen enthalten.
Eine sehr große Flexibilität ist bei make im Bereich der Erweiterungsmöglichkeiten gegeben, da dieses Werkzeug komplett auf externe Programme angewiesen ist. Auch Ant und jmk erlauben diese Form der Erweiterung, drüberhinaus können auch neue Java Klassen mit Funktionalitäten erstellt werden. Jmk stellt auch bereits in der Beschreibungssprache die Möglichkeit zur Definition von Funktionen und Operatoren bereit- dies ist meiner Meinung nach jedoch nicht nur als Vorteil zu werten, da hierdurch die Build-Dateien schwerer les- und wartbar werden.

Insbesondere für den Einsatz bei Java Projekten ist Ant prädesteniert, da dieses Build-Werkzeug die Fähigkeit besitzt, alle Übersetzungsvorgänge in einer einzigen Virtuellen Maschine durchzuführen. Da so der Aufwand für die jeweilige Neu-Initialisierung der VM's entfällt, weist Ant eine hohe Leistungsfähigkeit bei der Integrationsdurchführung auf.

Meines Erachtens ist Ant für neue Projekte zu empfehlen, die auf eine gute, plattformübergreifende Ünterstützung angewiesen sind. Komplexe Build-Vorgänge lassen sich leicht über die vielen Integrierten Funktionen lösen und selbst exotische Anforderungen sind einfach über die bereitgestellte Schnittstelle nachträglich zu Integrieren. Das Build-Format ist leicht erlernbar, verständlich und vor allem maschinenlesbar, so daß eine weitere Integration des gesamten Softwareentwicklungs-zyklusses gut zu implementieren ist.

5 Anhang

5.1 Sourcecode / Beispiele

5.1.1 Beispiel build.xml

```
<project name="DemoProjekt" default="deploy" basedir=".">
   <target name="init">
      <property name="sourceDir" value="src"/ >
      <property name="outputDir" value="classes" />
      <property name="deployDir" value="/web/deploy/" />
   </target>
   <target name="clean" depends="init">
      <delete dir="${outputDir}" />
      <mkdir dir="${outputDir}" />
   </target>
   <target name="compile" depends="clean">
     <javac srcdir="${sourceDir}" destdir="${outputDir}" />
   </target>
   <target name="deploy" depends="compile,init">
     <copy file="${jsp}" todir="${deployDir}"/>
  </target>
</project>
```

5.2 Quellenverzeichnis

Homepage des Apache Foundation Jakarta Projekts (Stand: 02.11. 2001)
http://jakarta.apache.org/ant/index.html

Cymerman, Michael. Automate your build process using Java and Ant (Stand: 02.11.2001)
http://www.javaworld.com/javaworld/jw-10-2000/jw-1020-ant.html

Wang, Laura Geele. Ant: A Build Tool from the Jakarta Project (Stand: 30. 05. 2001)
http://dcb.sun.com/practices/profiles/ant.jsp

Loughran, Steve. Ant Task Guidelines (Stand: 04.11.2001)
http://jakarta.apache.org/ant/ant_task_guidelines.html

Chanezon, Patrick. Ant Build Tool (Stand: 04.11. 2001)
http://people.netscape.com/chanezon/tech/ant/ant_preso.ppt

Free Software Foundation. GNU make (Stand: 06.09.2000)
http://www.gnu.org/software/make/make.html

Ramsdell, John D. Make in Java (Stand: 03.11.2001)
http://jmk.sourceforge.net/edu/neu/ccs/jmk/

Hompage des Junit-Framworks (Stand 02.11.2001)
http://www.junit.org/index.htm