AARHUS UNIVERSITET

Build Management

Clean environment that works...

Motivation

Sun provides Java SDK free of charge

- provides standard command line tools: javac, java, ...

These are sufficient only for *very* small systems

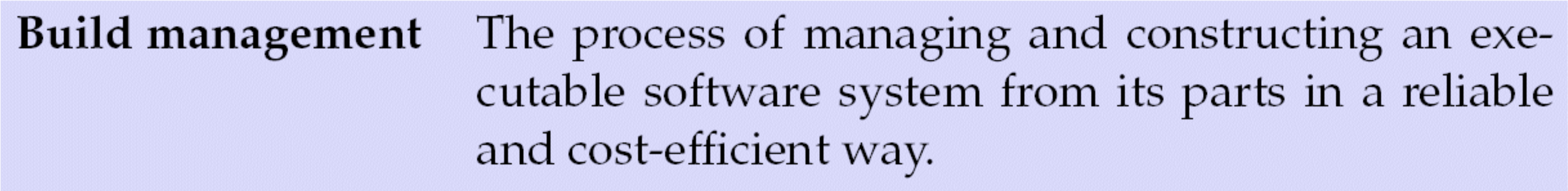
- javac only compile one directory at a time
- javac recompiles everything everytime

Large systems require many tasks

- manage resources (graphics, sound, config files)
- deployment (making jars, copying files)
- management (javadoc, coverage, version control)

Build-Management

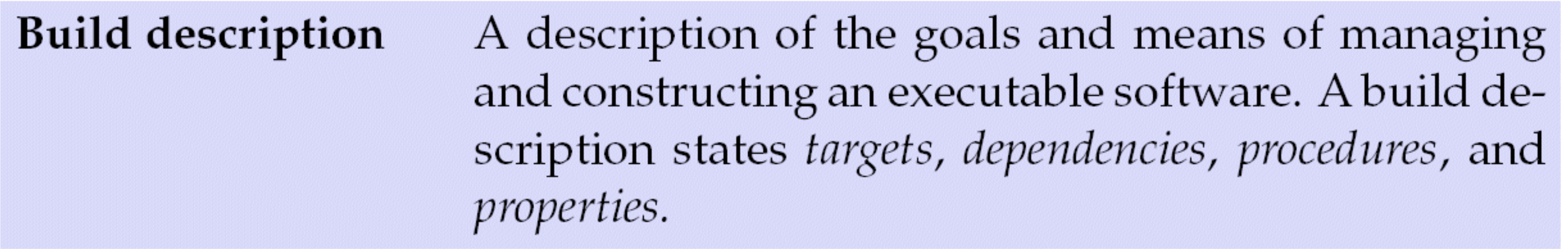
This problem is denoted:



Build management The process of managing and constructing an executable software system from its parts in a reliable and cost-efficient way.

Computer Scientists' standard solution: a tool...

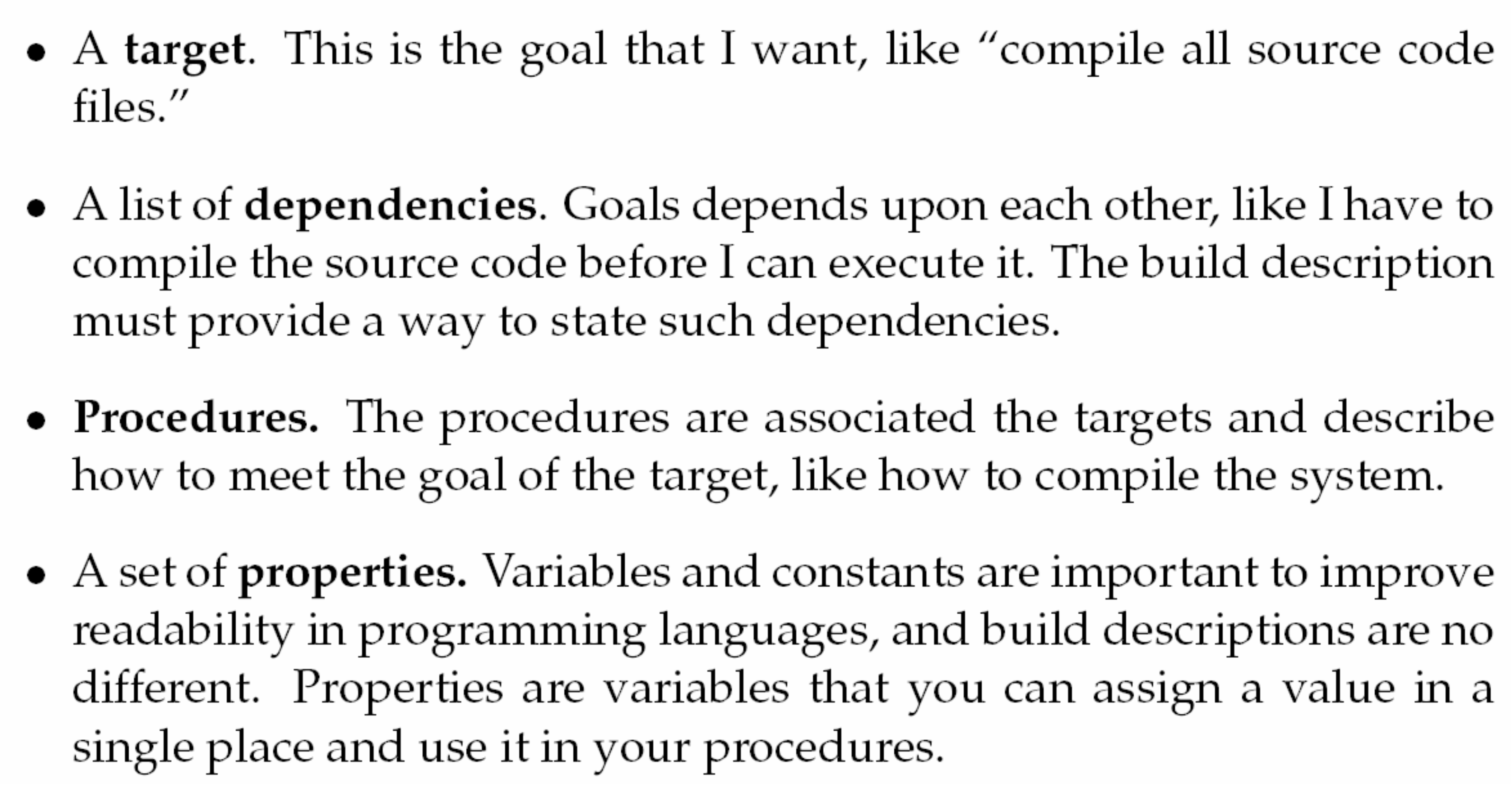
The tool read a *build-description*



Build description A description of the goals and means of managing and constructing an executable software. A build description states *targets*, *dependencies*, *procedures*, and *properties*.

Example: Make (Feldmann, 1979)

Script Parts



- A **target**. This is the goal that I want, like "compile all source code files."
- A list of **dependencies**. Goals depends upon each other, like I have to compile the source code before I can execute it. The build description must provide a way to state such dependencies.
- **Procedures.** The procedures are associated the targets and describe how to meet the goal of the target, like how to compile the system.
- A set of **properties.** Variables and constants are important to improve readability in programming languages, and build descriptions are no different. Properties are variables that you can assign a value in a single place and use it in your procedures.

Exercise

I want to run my Java system that is made of 500 sources files

- target?
- dependencies?
- procedure?
- properties?

Apache Ant

Ant is a young build-management tool geared towards Java

- ☺ has some strong build-in behaviour
  - javac on source *trees* and does smart recompile
- ☺ independent of large IDEs
  - easy for TA to unzip your submission and test it
- ☹ on the XML buzzword wave so it is verbose

Pay Station using Ant

The TDD again

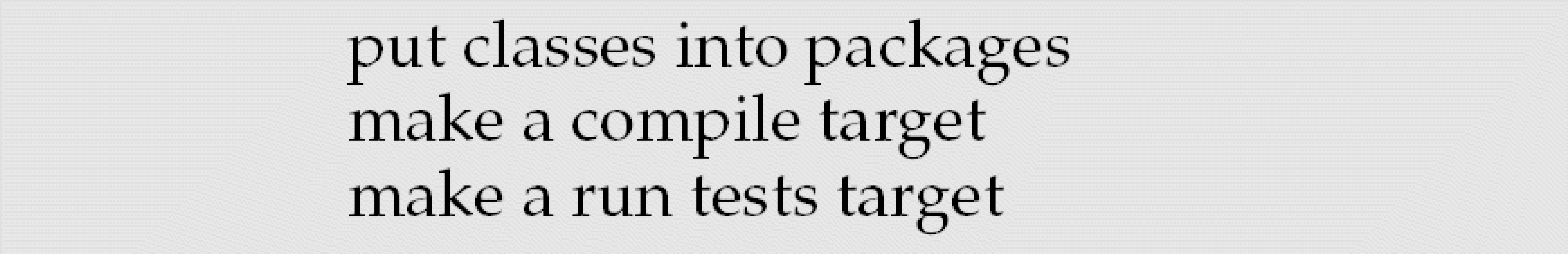
In my mind, TDD’s principles can be applied more widely than just developing code.

Basically I want a *refactoring* process

- from a windows .BAT development environment
- to an Ant based development environment
- ... but the external behaviour is the same:
    - compile it, test it

Test List

I start out (of course) with the test list



put classes into packages
make a compile target
make a run tests target

Java Packages

... in a minute or two...

Abstraction

Abstraction is the most important principle in computer science

- lower cognitive load on our poor mind by
- *hide large amounts of details behind meaningful named entities*

Examples:

- method
  - (name a large and complex set of statements)
- class
  - (name a large and complex set of methods)

Package

The Java *package* is the next level above classes.

Definition: A *package* is a collection of *related* classes and interfaces providing access protection and namespace management.

Declaration

A class declares that it belongs to a package by a statement:

```
package myPackage;
public class mySuperDuperClass {
…
}
```

Pretty weird! Compare C#

- namespace myPackage { class ... }

Access

To use a class in a package you must either

- qualify its name completely (package names are part of the class name)
  - java.util.List l = new java.util.ArrayList();
- or once and for all import the class
  - import java.util.List; import java.util.ArrayList;
- or get all the classes
  - import java.util.*;

Physical Structure

Java is peculiar in that it insists on a one-to-one match between package structure and physical storage structure of the binary .class files:

- java.util.List
- **must** be stored in a directory structure
- (something)/java/util/List.class
- (something)/java/util/List.java

I like the correspondence as it helps me locate the source files!

Classpath

You must tell where the compiler and JVM must start searching for the files:

- (something)/java/util/List.class

The CLASSPATH tells where ‘something’ is.

- javac –classpath src;lib\myutil.jar myclass.java
- means: search in folder ‘src’ and in the named jar file.

jar files are simply zip archives that obey the folder hierarchical structure.

Iteration 1

Packages

Pay Station

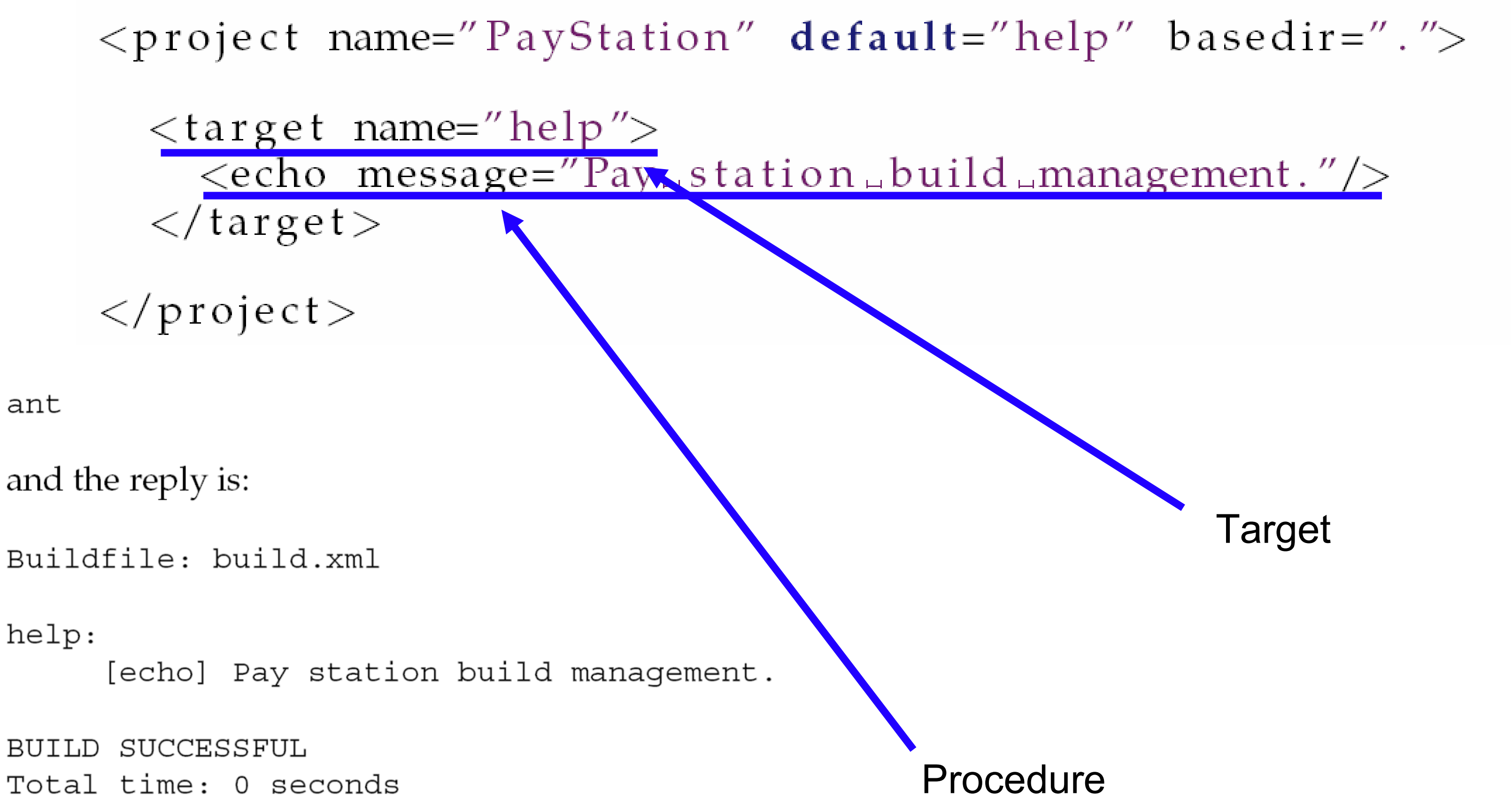
[iteration-1 in chapter/build-management in the FSE source code zip]

Iteration 2

Make a compile target

Hitting Ant

Take small steps!



```
<project name="PayStation" default="help" basedir=".">

  <target name="help">
    <echo message="Pay␣station␣build␣management."/>
  </target>

</project>
```

```
ant
```

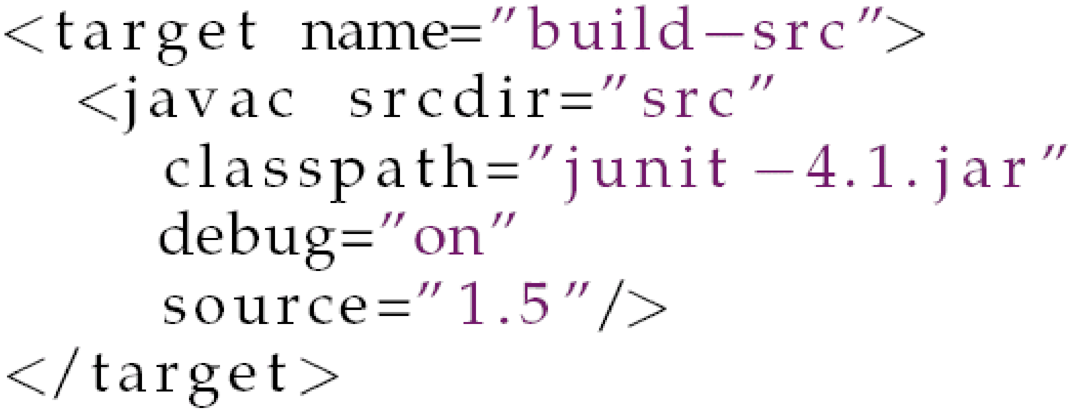
and the reply is:

```
Buildfile: build.xml

help:
     [echo] Pay station build management.

BUILD SUCCESSFUL
Total time: 0 seconds
```

Target

Procedure

javac task

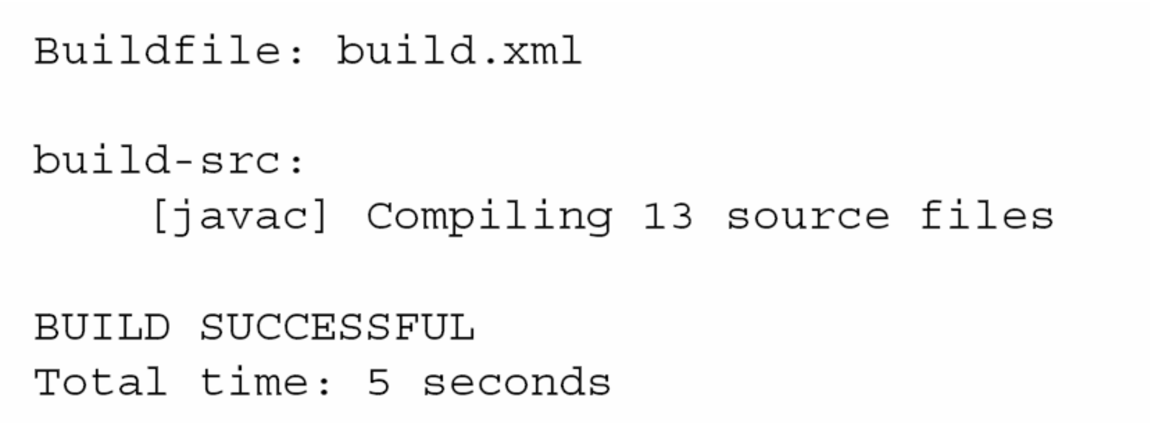


```
<target name="build-src">
  <javac srcdir="src"
     classpath="junit-4.1.jar"
     debug="on"
     source="1.5"/>
</target>
```

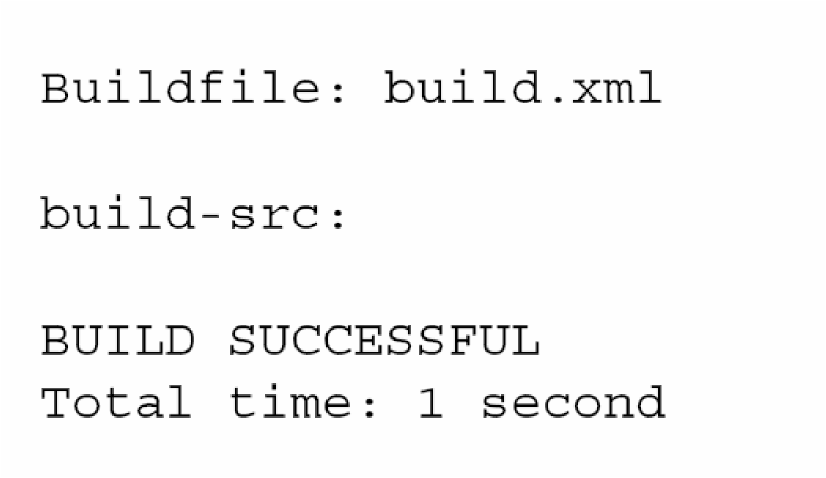
javac does

- recursive descent in full source code tree
- smart compilation

Smart Compilation



```
Buildfile: build.xml

build-src:
    [javac] Compiling 13 source files

BUILD SUCCESSFUL
Total time: 5 seconds
```

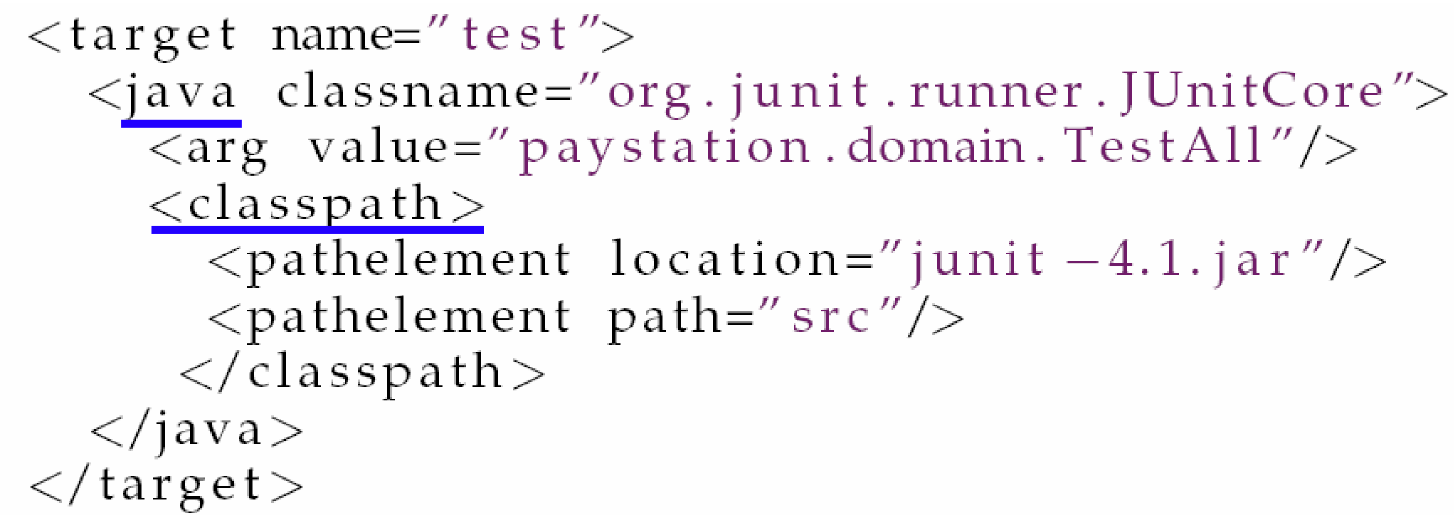


```
Buildfile: build.xml

build-src:

BUILD SUCCESSFUL
Total time: 1 second
```

Iteration 3

Running the tests

java task



```
<target name="test">
  <java classname="org.junit.runner.JUnitCore">
    <arg value="paystation.domain.TestAll"/>
    <classpath>
      <pathelement location="junit-4.1.jar"/>
      <pathelement path="src"/>
     </classpath>
  </java>
</target>
```

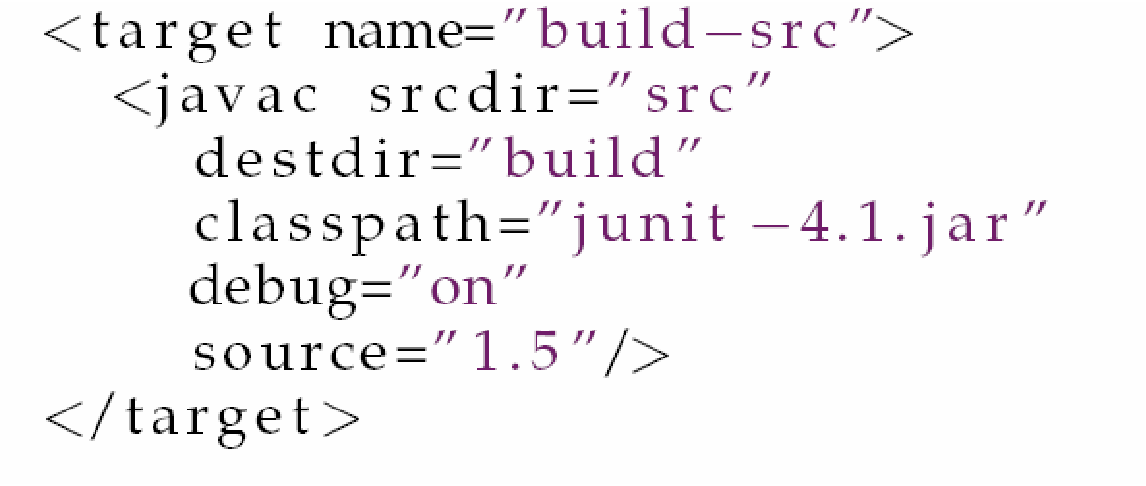
Iteration 4

Split Build Tree

Avoid pollution

I like to keep things separate!

- source code trees should not contain .class files

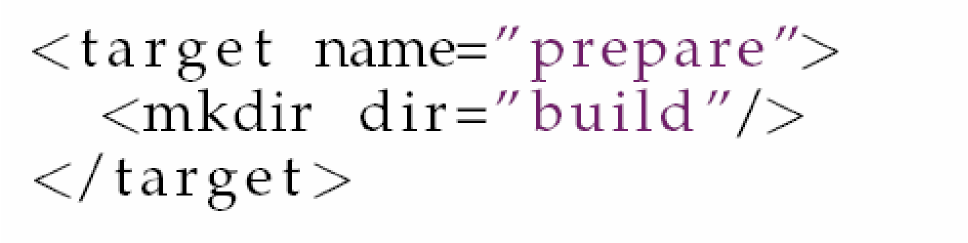


```
<target name="build-src">
  <javac srcdir="src"
      destdir="build"
      classpath="junit-4.1.jar"
      debug="on"
      source="1.5"/>
</target>
```

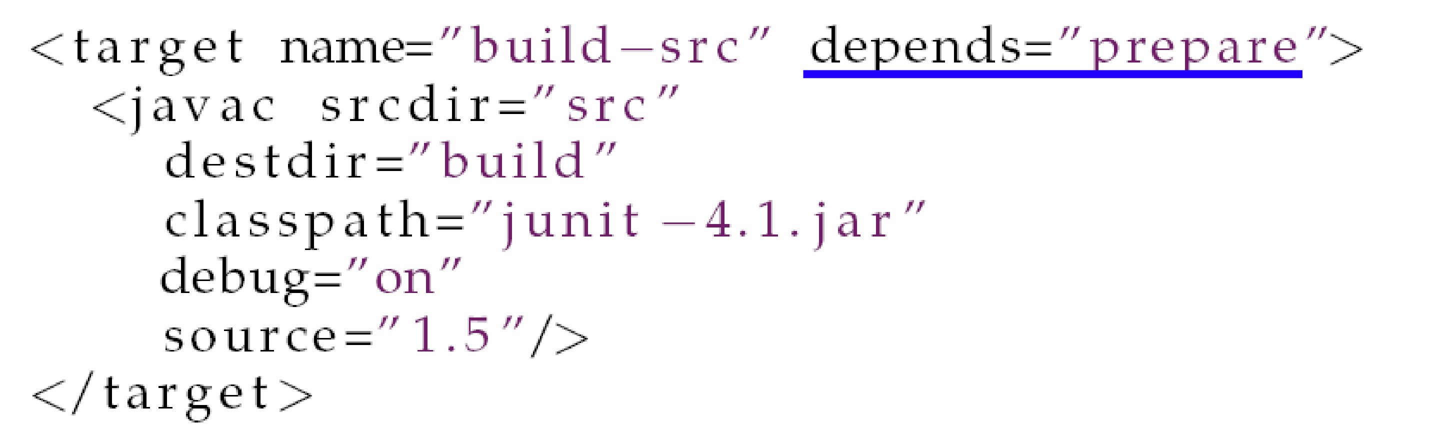
AARHUS UNIVERSITET

Dependencies

It however fails because no ‘build’ directory exists. Let us resolve that using a *prepare* target.



```
<target name="prepare">
  <mkdir dir="build"/>
</target>
```

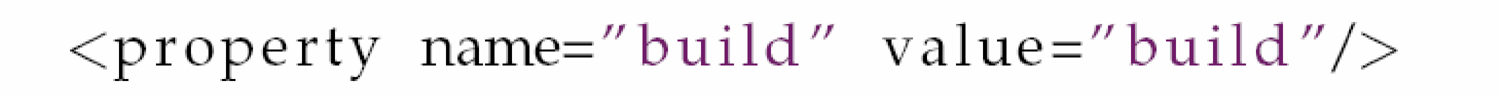


```
<target name="build-src" depends="prepare">
  <javac srcdir="src"
     destdir="build"
     classpath="junit-4.1.jar"
     debug="on"
     source="1.5"/>
</target>
```

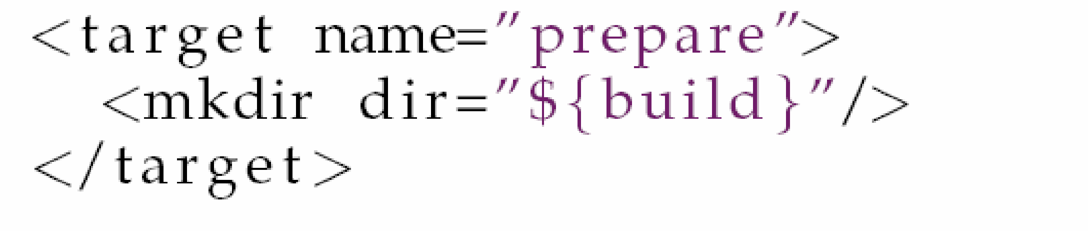
Refactoring

Refactor to clean up!

- ‘build’ as string literal all over the place



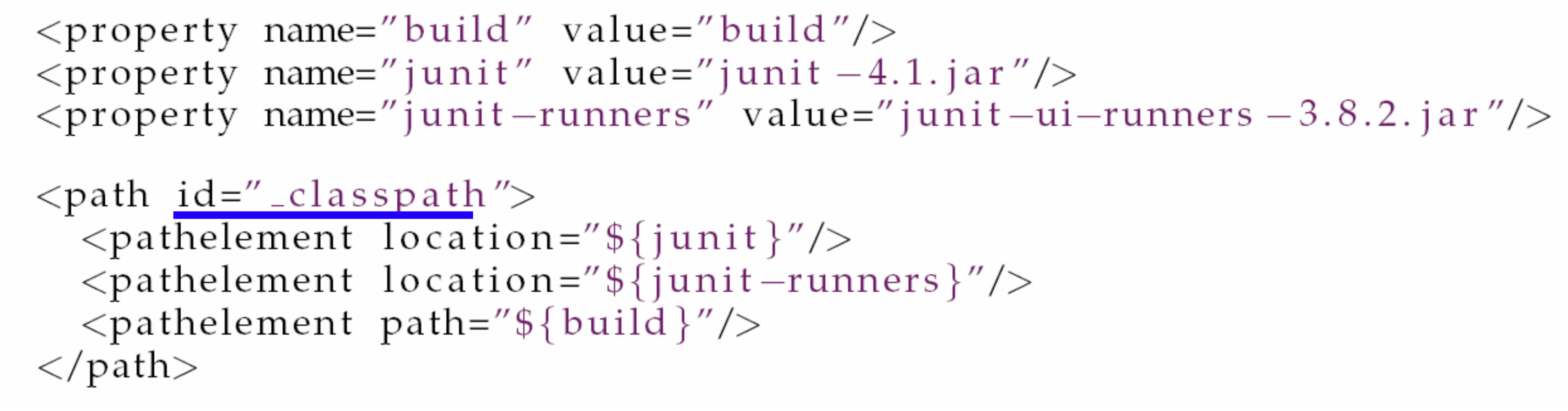
```
<property name="build" value="build"/>
```



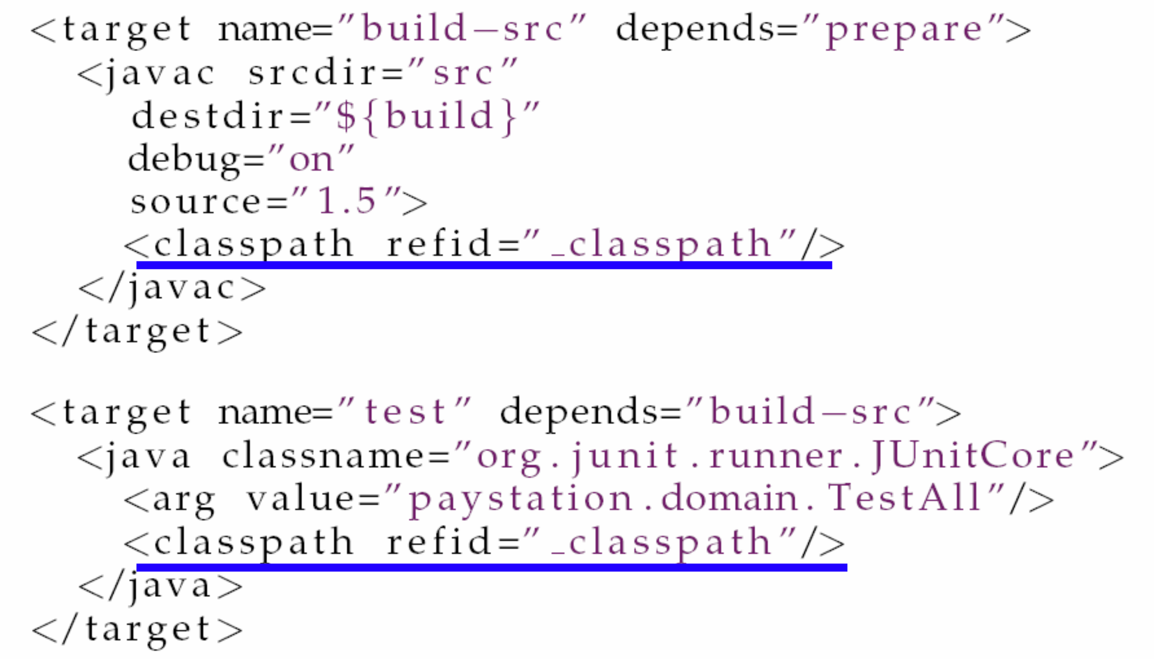
```
<target name="prepare">
  <mkdir dir="${build}"/>
</target>
```

(I think build-tools compete to introduce really *weird* syntax!!!)

Cleaning up classpath



```
<property name="build" value="build"/>
<property name="junit" value="junit-4.1.jar"/>
<property name="junit-runners" value="junit-ui-runners-3.8.2.jar"/>

<path id="_classpath">
  <pathelement location="${junit}"/>
  <pathelement location="${junit-runners}"/>
  <pathelement path="${build}"/>
</path>
```



```
<target name="build-src" depends="prepare">
  <javac srcdir="src"
    destdir="${build}"
    debug="on"
    source="1.5">
    <classpath refid="_classpath"/>
  </javac>
</target>

<target name="test" depends="build-src">
  <java classname="org.junit.runner.JUnitCore">
    <arg value="paystation.domain.TestAll"/>
    <classpath refid="_classpath"/>
  </java>
</target>
```

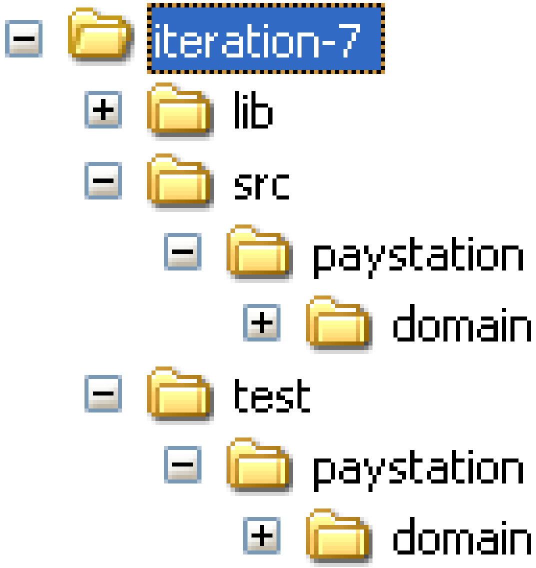
Iteration 6

Split production and test tree

Splitting the trees

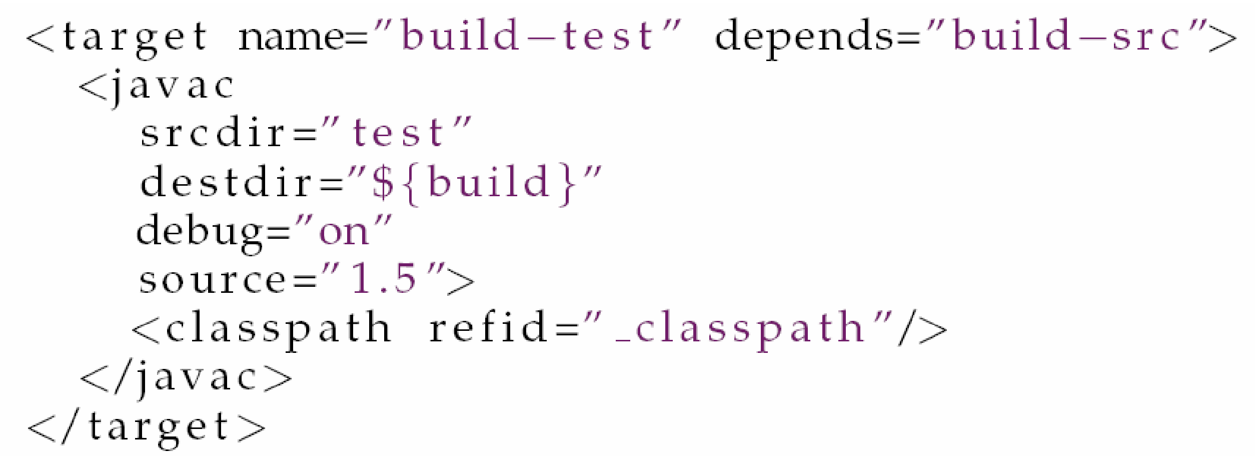
Advantages

- Make javadoc without refering to unit test
- Make jar of production code without unit test



Require New Task

Have to build the test tree as well



```
<target name="build-test" depends="build-src">
  <javac
     srcdir="test"
     destdir="${build}"
     debug="on"
     source="1.5">
     <classpath refid="_classpath"/>
  </javac>
</target>
```

Summary

Summary

Build-management automates many ‘household’ tasks

Build-management = tool + build script

Build scripts are documentation!

- tell me how to run servers and clients even when names and topologies have changed
  - as long as we agree on the task names, like “run-server”