BUILDING YOU CODE

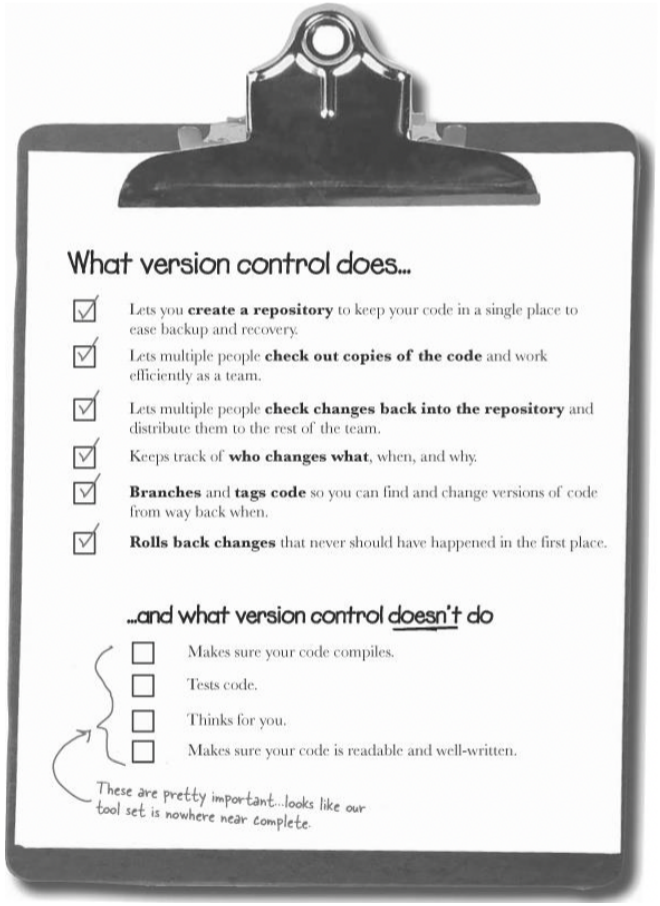
BUILD TOOLS AND CONSTRUCTION

Content from Chapter 6.5 of "Head First Software Development", Pilone et al.

Miami University Software Technology & Analysis Group (MUSTANG)
Computer Science & Software Engineering
Miami University, Oxford, Ohio, USA

Questions from last classes?

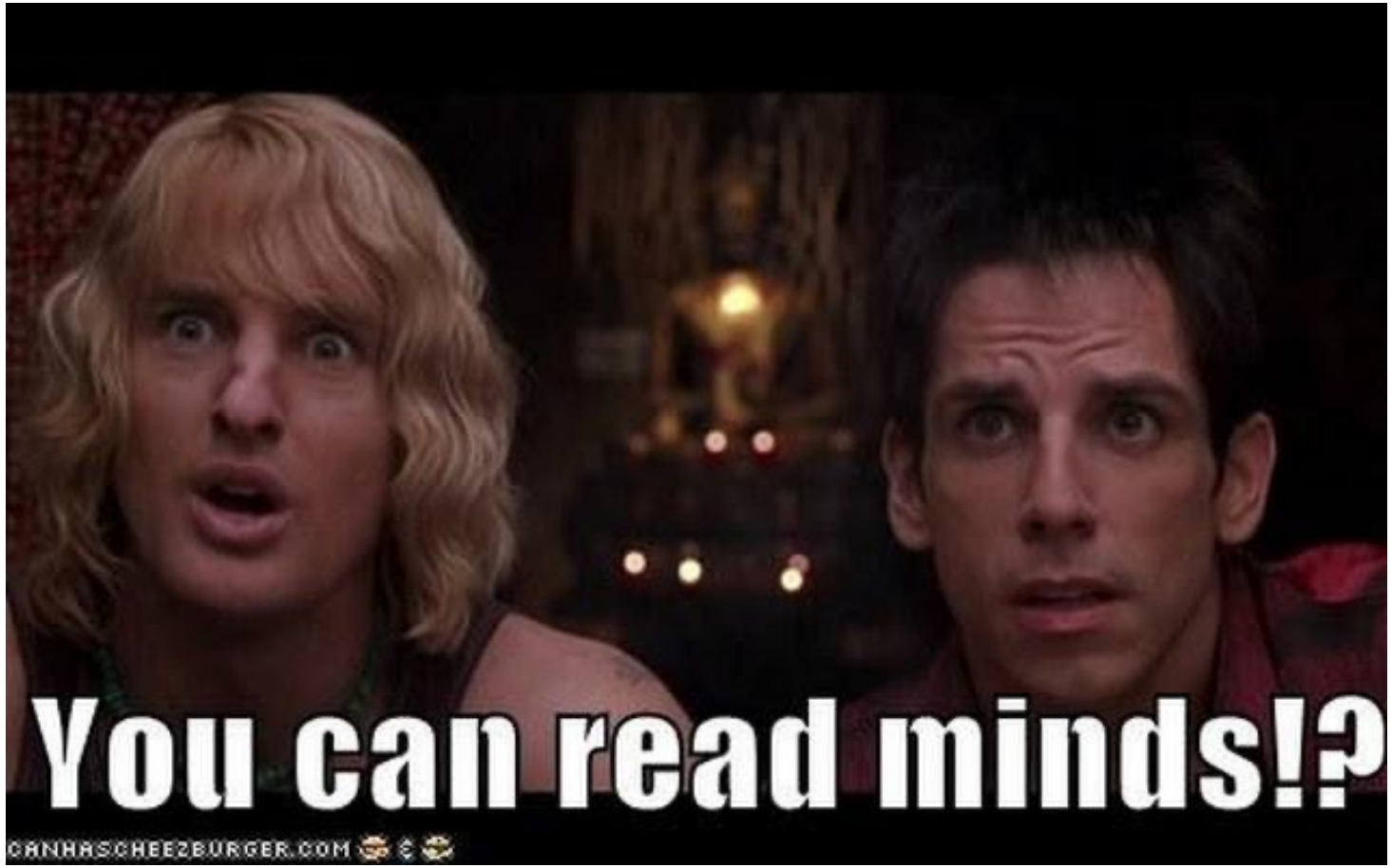
- TDD
  - What is it?
  - Benefits?
  - Tool suite for Java?
- Version Control: VCS is Not the "Silver Bullet"
  - SVN
  - GIT Hub/Lab
  - Differences?
  - Gitlab to Agile Mappings



CODE CONSTRUCTION

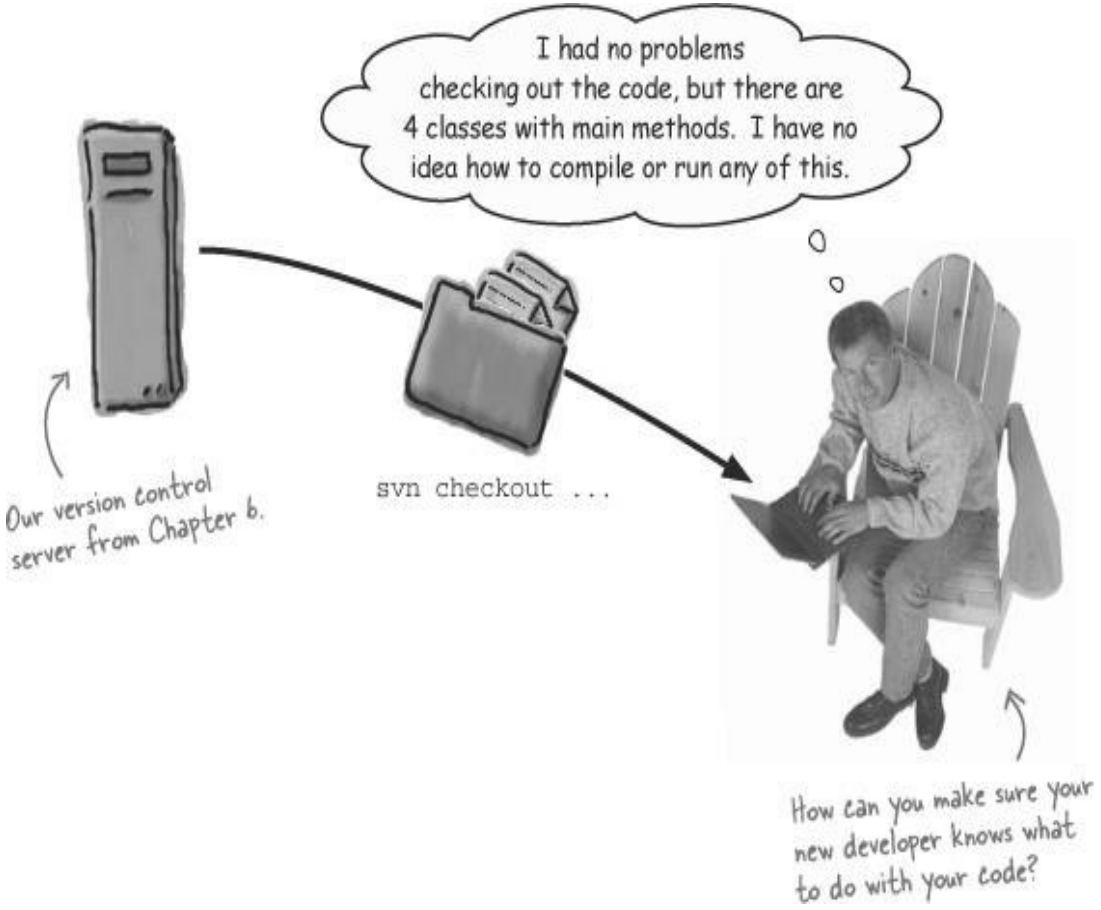
- It **pays to follow the instructions... ...especially when you write them yourself.**
- It's not enough to use version control to ensure your code stays safe.
- You've also got to worry about **compiling your code** and packaging it into a deployable unit.

DEVELOPERS CAN'T READ MINDS



DEVELOPERS CAN'T READ MINDS

- A new developer joins your team
  - They can check out code from your VCS
  - Not concerned about overwriting since VCS allows roll back.
- How does new team member know about dependencies or classes to run?



Software Must Be Usable

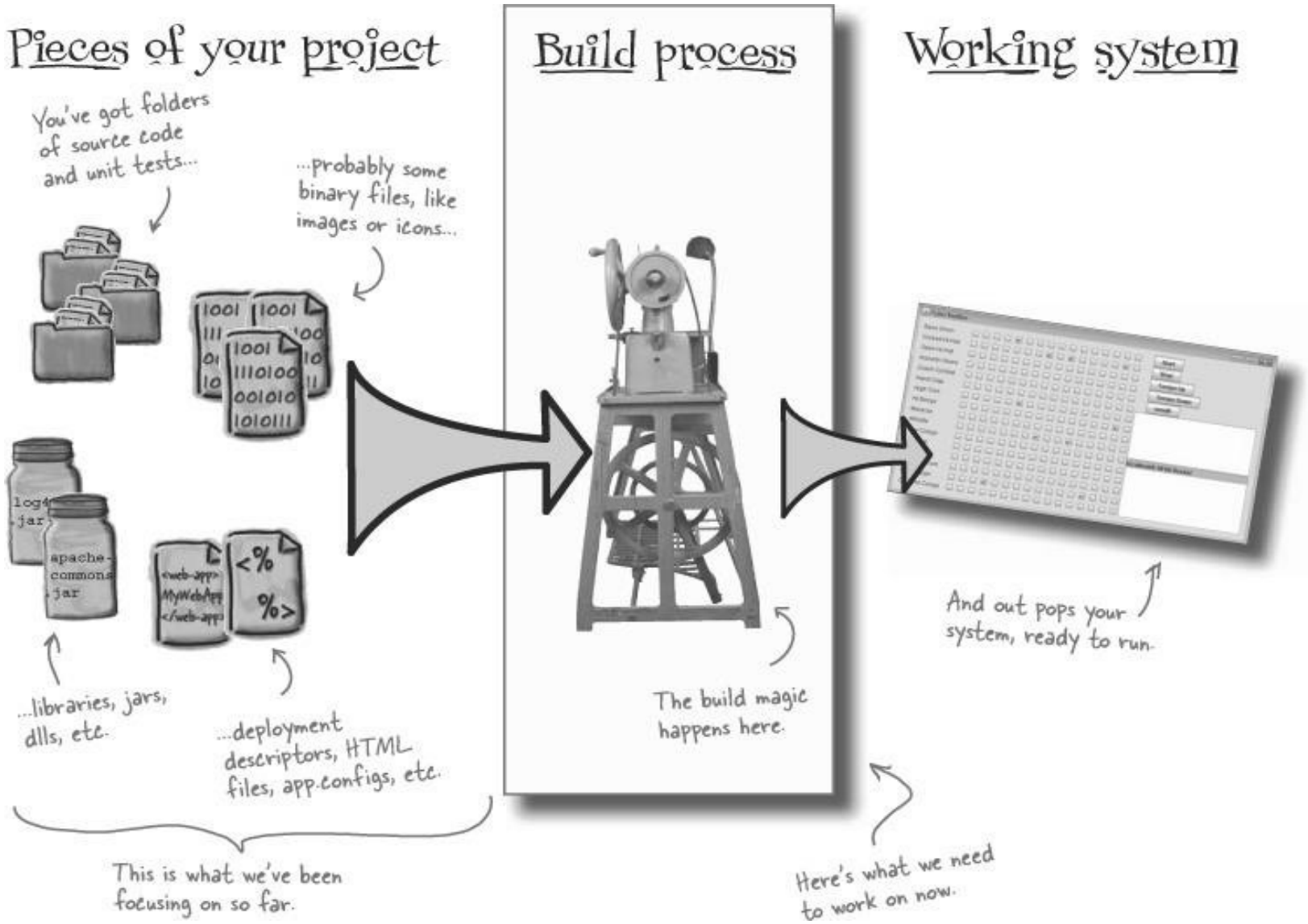
- It doesn't do you much good to put in a version control server if you can't also be sure your code is used properly once it's checked out.
- And that's where **build scripts** come in.
- Good code is easy to use, as well as easy to get (retrieve).

Building Our Projects

- To run your project, it involves more than just compile source code.
- REQUIRES BUILDING THE PROJECT
  - Finding dependencies
  - Packaging up project
  - More

Building Our Projects

- Tasks are the same each time
- Perfect candidate for automation.
  - Tool to handle repetitive work for you
  - We are engineers, after all.
- Think about IDE, like Eclipse. All of this is done through the "Build" button/option.


Pieces of your project
Build process
Working system
You've got folders of source code and unit tests...
...probably some binary files, like images or icons...
log4j.jar
apache-commons.jar
<web-app>
MyWebApp
</web-app>
<%
%>
...libraries, jars, dlls, etc.
...deployment descriptors, HTML files, app.configs, etc.
The build magic happens here.
And out pops your system, ready to run.
This is what we've been focusing on so far.
Here's what we need to work on now.

ANT: A BUILD TOOL FOR JAVA PROJECTS

- Ant is a build tool for Java that can compile code, create and delete directories, and even package up files for you.
- It all centers around a build script.
  - That's a file you write, in XML for Ant, that tells the tool what to do when you need to build your program.
- http://ant.apache.org/

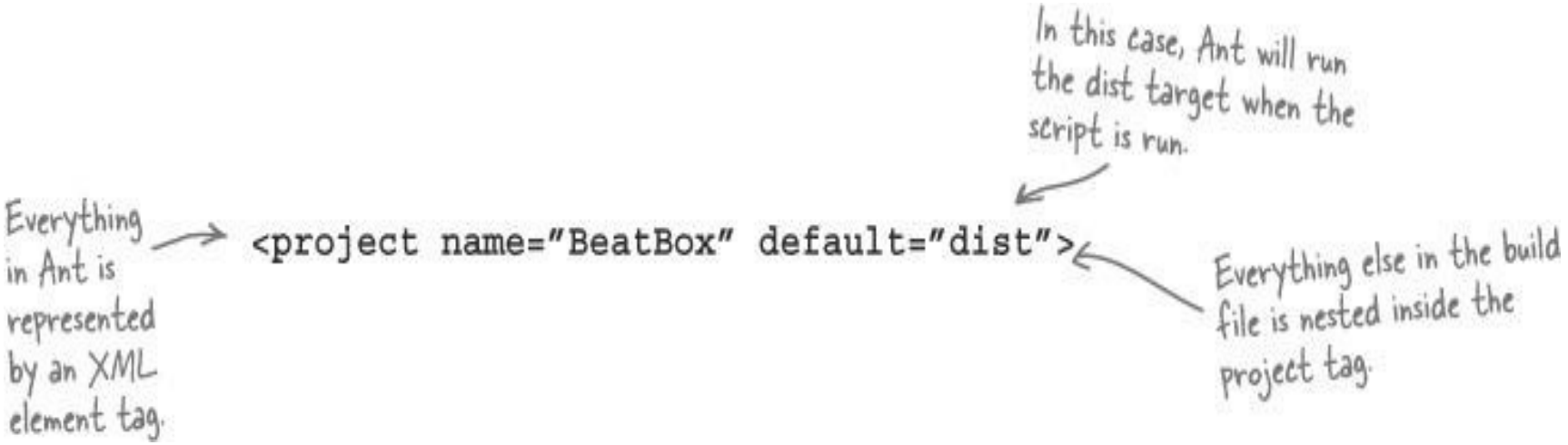

The steps to build your project are stored in an XML file, usually named build.xml.
This whole file is called a build script.
Each build file represents a single project
<project name="HFSDCoverage" default="dist" basedir=".">
<property name="src" location="src"/>
<property name="bin" location="bin"/>
<property name="bin-instrumented" location="bin-instrumented"/>
<property name="reportdir" location="reportdir"/>
<target name="clean">
<delete dir="${bin}"/>
<delete dir="${reportdir}"/>
<delete dir="${bin-instrumented}"/>
<delete file="cobertura.ser"/>
</target>
<target name="init">
<mkdir dir="${bin}"/>
<mkdir dir="${reportdir}"/>
What's needed to build your project is broken up into steps called targets. Each target can have more than one task.
You just kick off a build with a single command. Ant runs the default target in build.xml, and follows your instructions.
File Edit Window Help Dolt
hfsd> ant
Buildfile: build.xml
In this case, Ant creates a directory structure...
init:
[mkdir] Created dir: C:\Users\Developer\workspaces\HFSD\BeatBox\bin
[mkdir] Created dir: C:\Users\Developer\workspaces\HFSD\BeatBox\dist
...compiles code into that structure...
compile:
[javac] Compiling 4 source files to C:\Users\Developer\workspaces\HFSD\BeatBox\bin
dist:
[jar] Building jar: C:\Users\Developer\workspaces\HFSD\BeatBox\dist\BeatBox.jar
...and builds a JAR file.
BUILD SUCCESSFUL
Total time: 16 seconds
hfsd>

PROJECTS, PROPERTIES, TARGETS, AND TASKS

- **Ant Build Files** are comprised of 4 aspects
  - Project
  - Properties
  - Tasks
  - Targets

PROJECTS, PROPERTIES, TARGETS, AND TASKS

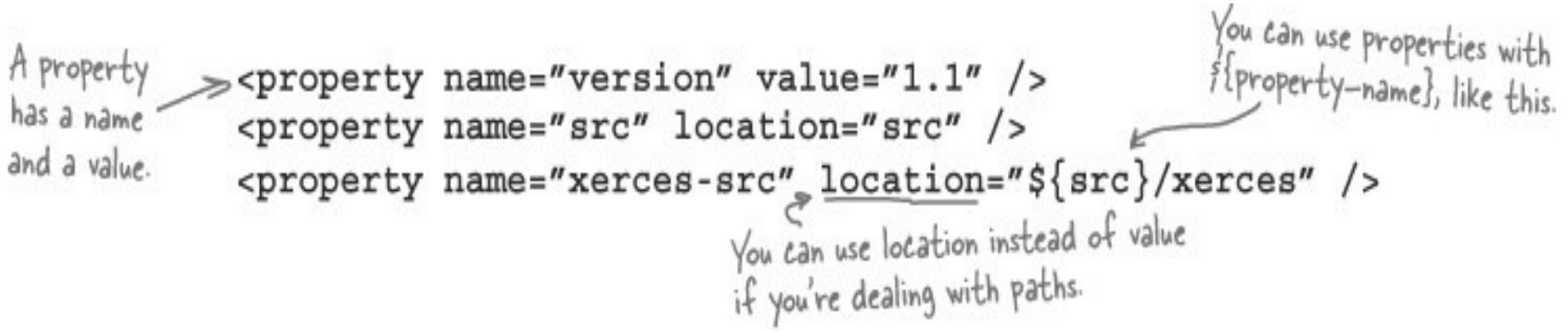
- Ant Build Files are comprised of 4 aspects
- 1) Projects
  - Everything in your build file is part of a single project:



  - Each project should have a name and default target to run.

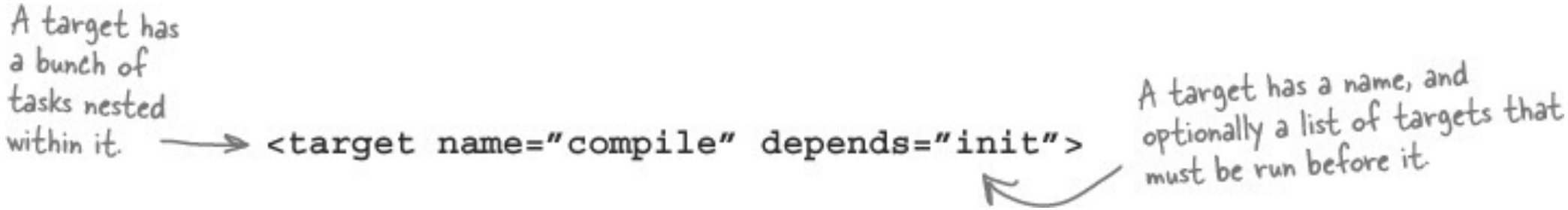
PROJECTS, PROPERTIES, TARGETS, AND TASKS

- Ant Build Files are comprised of 4 aspects
- 2) Properties
  - Constants in your project. Lets you refer to values in the scripts that you can change in a single place.



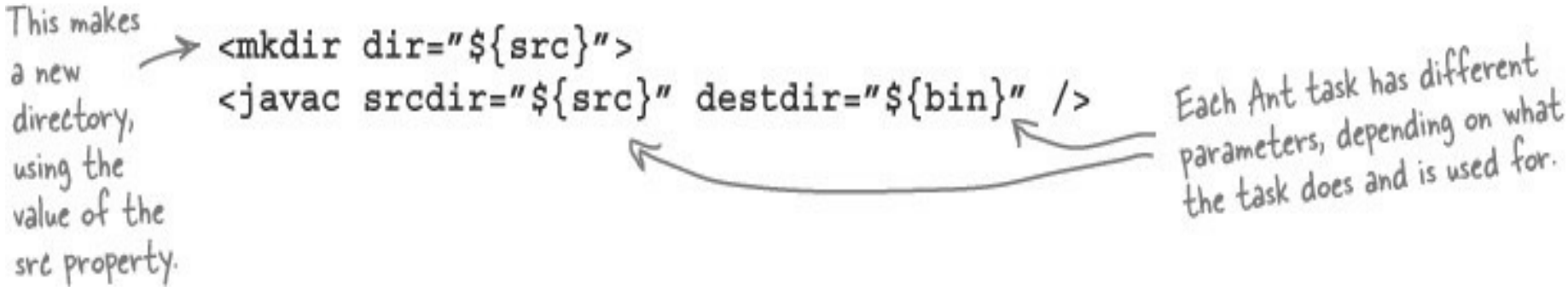
PROJECTS, PROPERTIES, TARGETS, AND TASKS

- Ant Build Files are comprised of 4 aspects
- 3) Targets
  - You can group different actions into a target, which is a set of work.
  - For example, you might have a `compile` target for compilation, and an `init` target for setting up your project's directory structure.



PROJECTS, PROPERTIES, TARGETS, AND TASKS

- Ant Build Files are comprised of 4 aspects
- 4) Tasks
  - Tasks are the work horses of your build script.
  - A task in Ant usually maps to a specific command, like **javac**, **mkdir**, or even **javadoc**:



No Java For You?

- **The syntax here is particular to Ant, but the principles work with all build tools, in any language.**
- A good build tool gives you: a way to manage projects, constants, and specific tasks.
- Other build tools that work with other languages, like PHP, Ruby, and C#.

Strength Of Build Tool

- While it is a new language/tool
  - It is easy to learn. Just a quick syntax lesson required
  - It's a tool that helps get things done faster over many projects
- It is for your team not just you.
  - Helps explain dependencies to your team
  - Everyone uses same process
  - Takes just one command

GOOD BUILD SCRIPTS?

- Captures the details that developers don't need to know from the start.
- Information is no longer kept in one person's head
  - Version-controlled, repeatable process

- But what exactly should a standard build script do?

GOOD BUILD SCRIPTS?

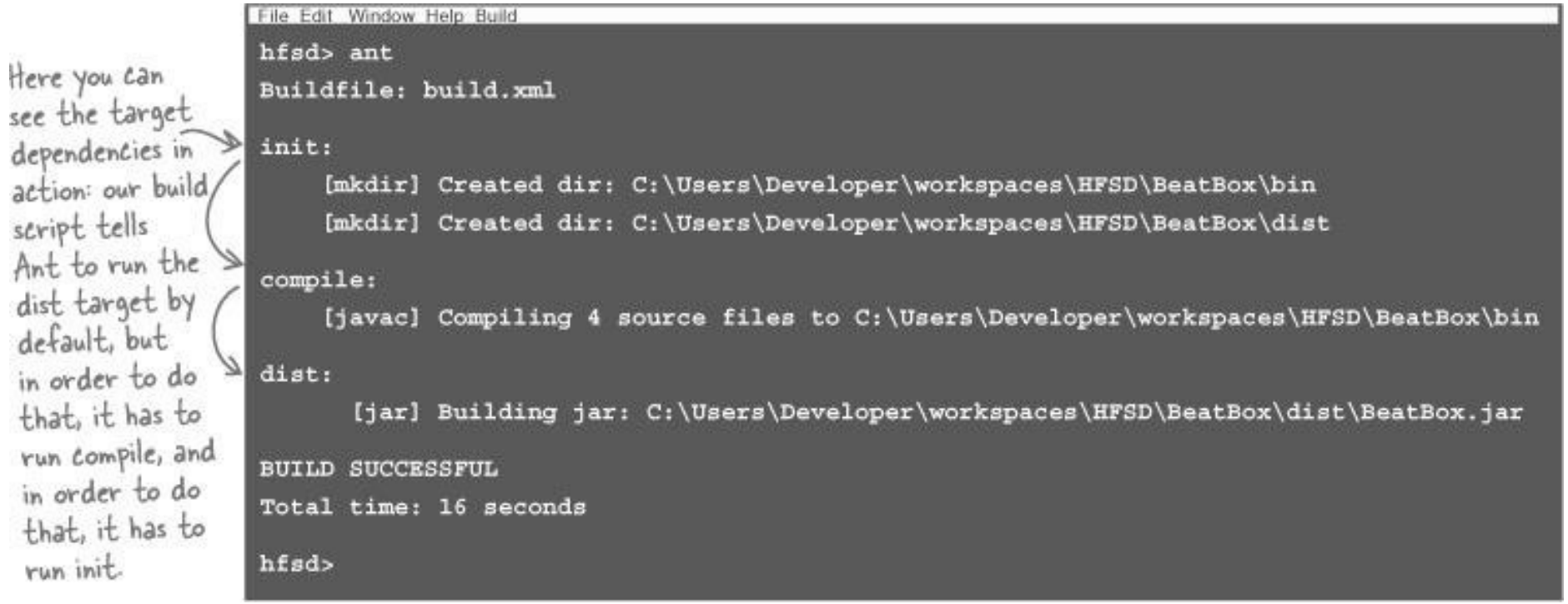
- Generate Documentation

```
File Edit Window Help Huh?
hfsd> ant -projecthelp
Buildfile: build.xml

Main targets:

 clean    Cleans up the build and dist directories.
 compile  Compiles the source files to the bin directory.
 dist     Packages up BeatBox into BeatBox.jar
 init     Creates the needed directories.
Default target: dist

hfsd>
```

Your build tool probably has a way to generate documentation about itself and your project, even if you're not using Ant and Java.

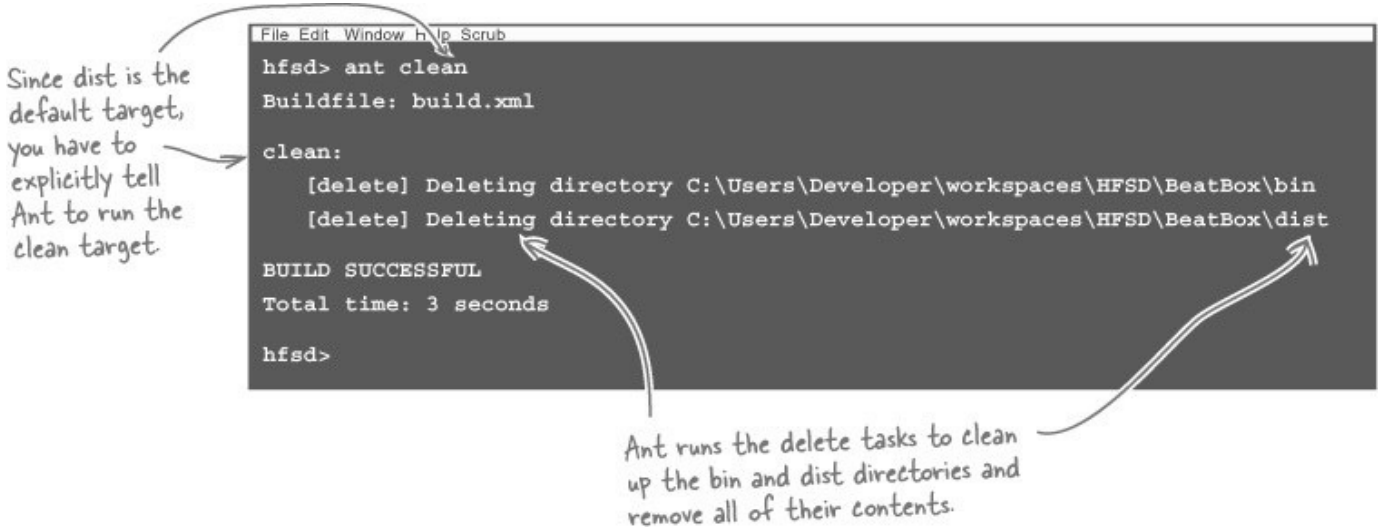
GOOD BUILD SCRIPTS?

- Compile your project code



GOOD BUILD SCRIPTS?

- Clean up the scraps of things that compiling leaves laying around.
- It's important to have a target that will get the project back to what it would look like if you checked the project out from the repository.
- That way, you can test things from a new developer's perspective.



GOOD BUILD SCRIPTS?

- Reference libraries your project needs

```
<javac srcdir="src" destdir="bin">
  <classpath>
    <pathelement location="libs/junit.jar"/>
    <pathelement location="libs/log4j.jar"/>
  </classpath>
</javac>
```

Each pathelement points to a single JAR to add to the classpath. You can also point to a directory if you need to.

GOOD BUILD SCRIPTS?

- Run your application

```
<exec executable="cmd">
  <arg value="/c"/>
  <arg value="iexplorer.exe"/>
  <arg value="http://www.headfirstlabs.com/"/>
</exec>
```

Executing something on the system directly is obviously going to be platform-dependent. Don't try to run iexplorer.exe on Linux.

(but do go to Head First Labs)

```
<java classname="headfirst.sd.chapter6.BeatBox">
  <arg value="HFBuildWizard"/>
  <classpath>
    <pathelement location="dist/BeatBox.jar"/>
  </classpath>
</java>
```

If you wrap this in a target then you won't ever have to type "java -cp blahblah..." again to launch BeatBox.

GOOD BUILD SCRIPTS?

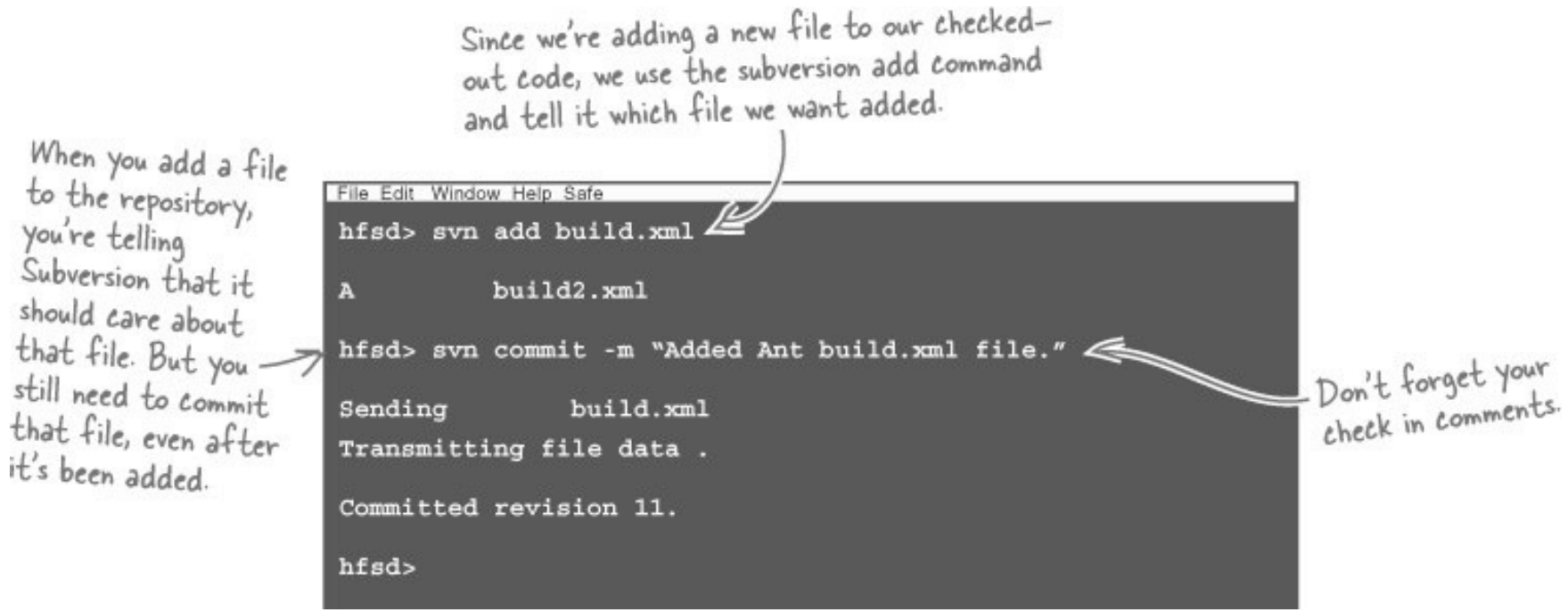
- Generate documentation

```
<javadoc packagenames="headfirst.sd.*"
  sourcepath="src"
  destdir="docs"
  windowtitle="BeatBox Documentation"/>
```

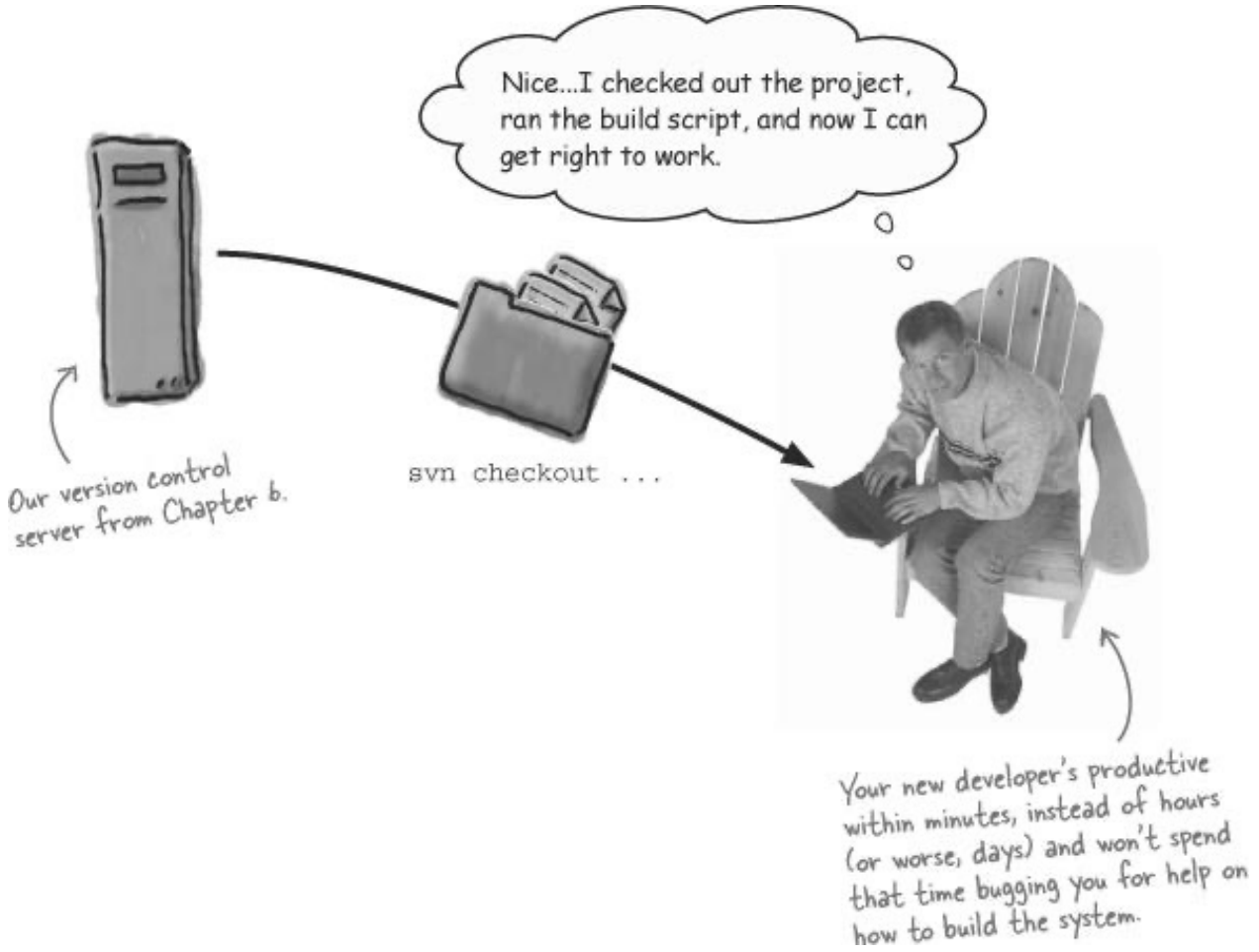
There are other elements you can include in the JavaDoc task to generate headers and footers for each page if you need to.

BUILD SCRIPTS ARE CODE, TOO!

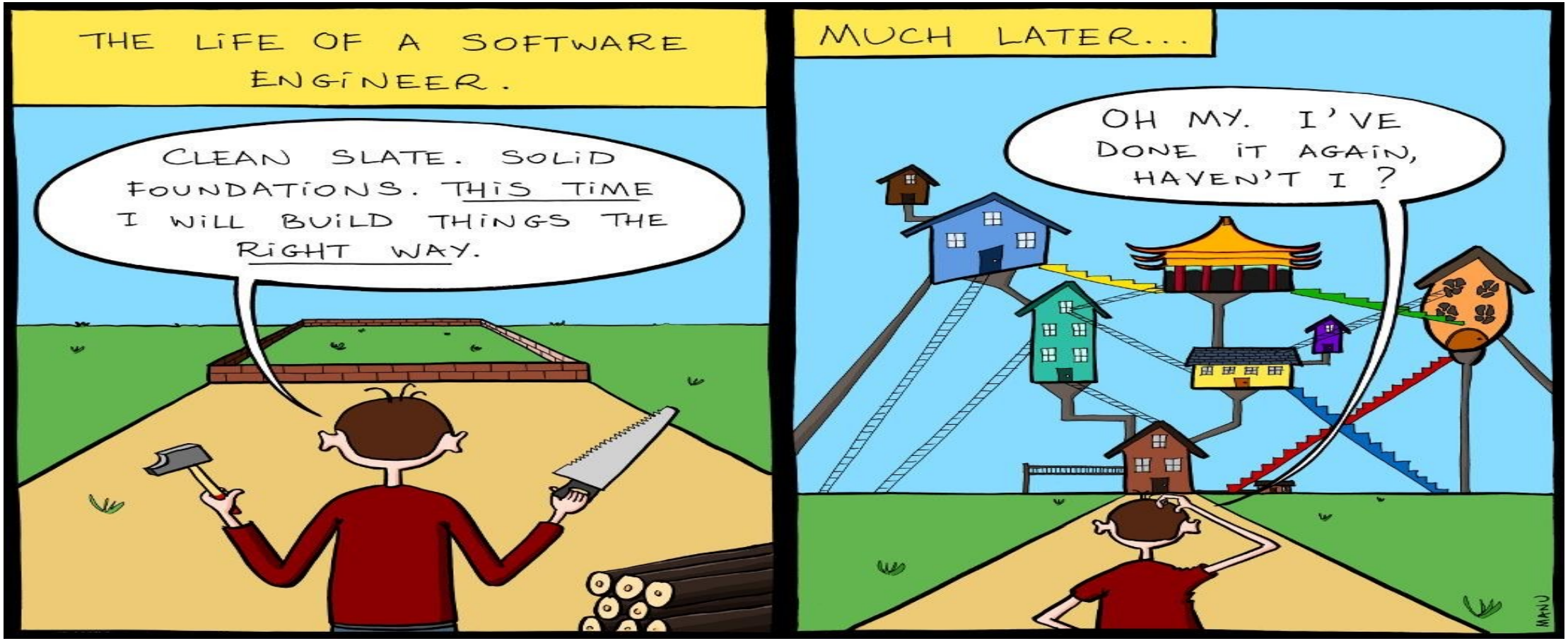
- So, build scripts should be checked in and subject to VCS.
  - Versioned, tagged, and saved.

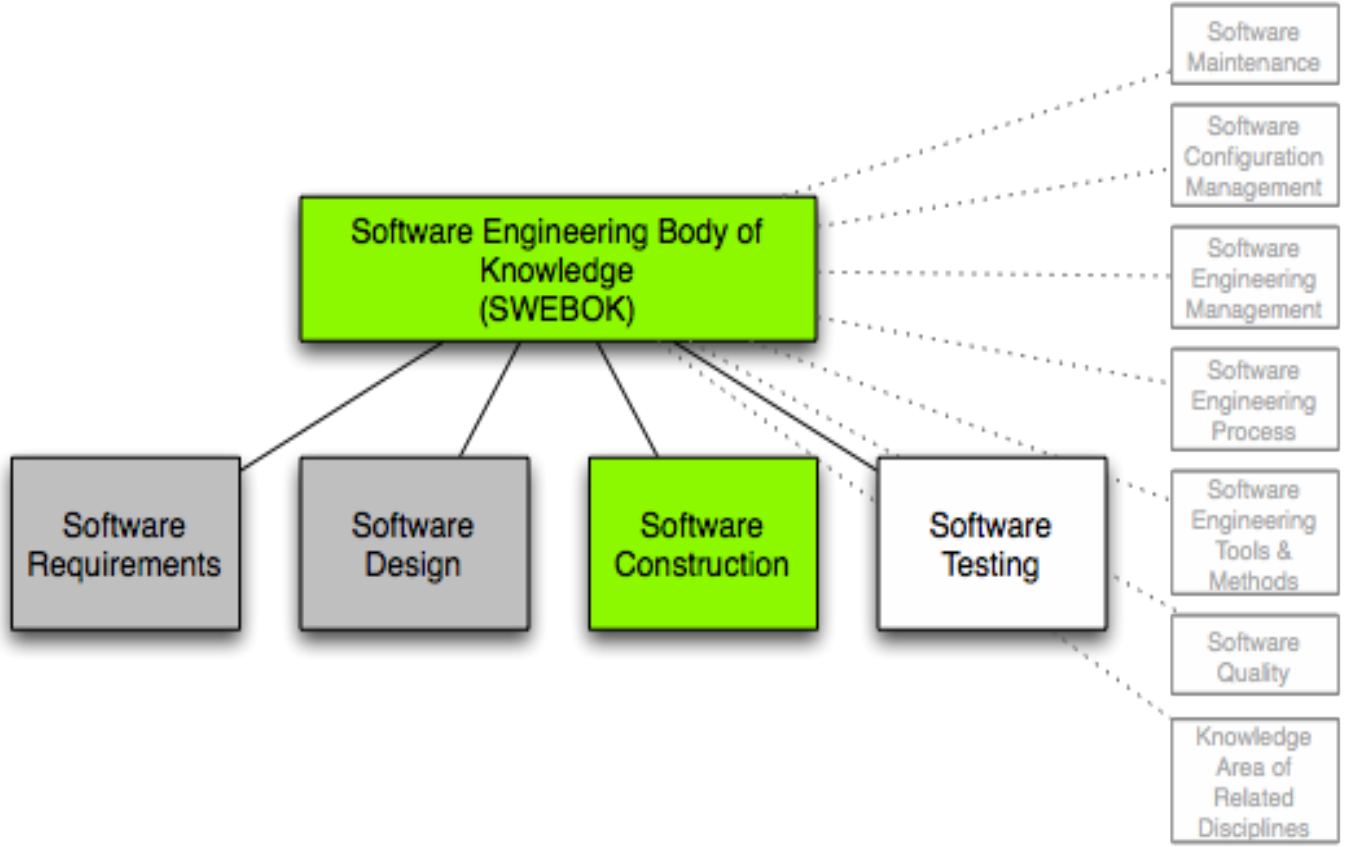


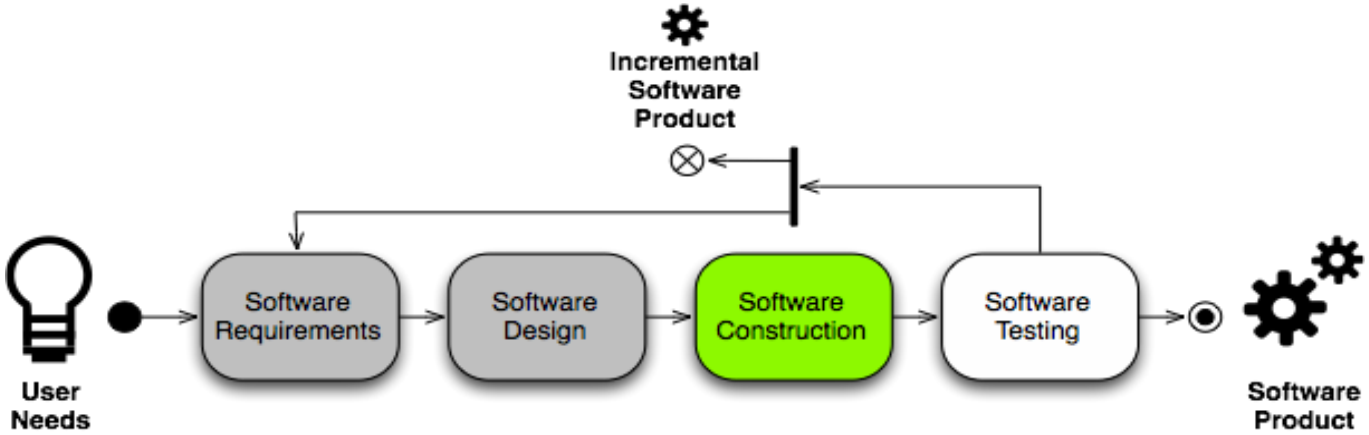
Now


Nice...I checked out the project, ran the build script, and now I can get right to work.
Our version control server from Chapter 6.
svn checkout ...
Your new developer's productive within minutes, instead of hours (or worse, days) and won't spend that time bugging you for help on how to build the system.

CONSTRUCTION




Software Engineering Body of Knowledge (SWEBOK)
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Tools & Methods
Software Quality
Knowledge Area of Related Disciplines

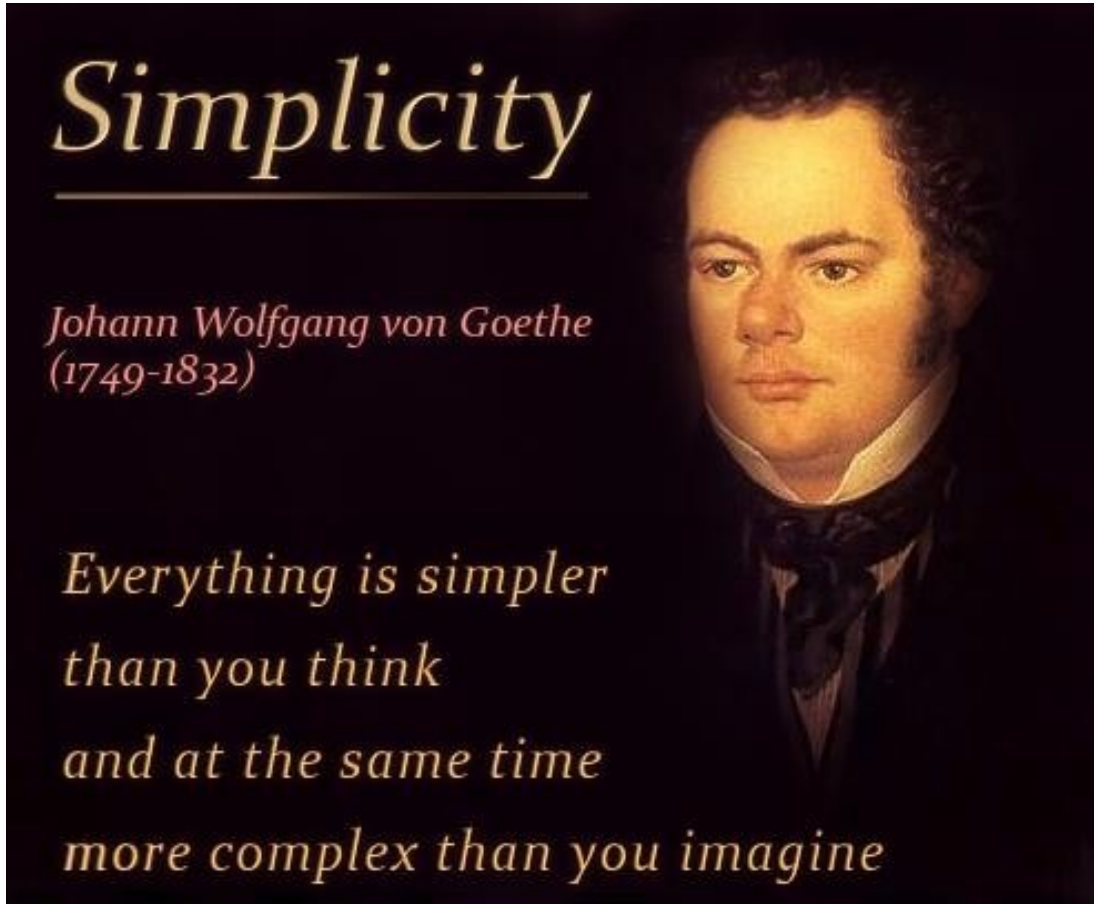

Incremental Software Product
Software Requirements
Software Design
Software Construction
Software Testing
User Needs
Software Product

SOFTWARE CONSTRUCTION

- Fundamentals of Software Construction
  - Minimizing complexity
    - Keep it simple
    - Use standards
  - Anticipating change
  - Constructing for verification
  - Standards in construction

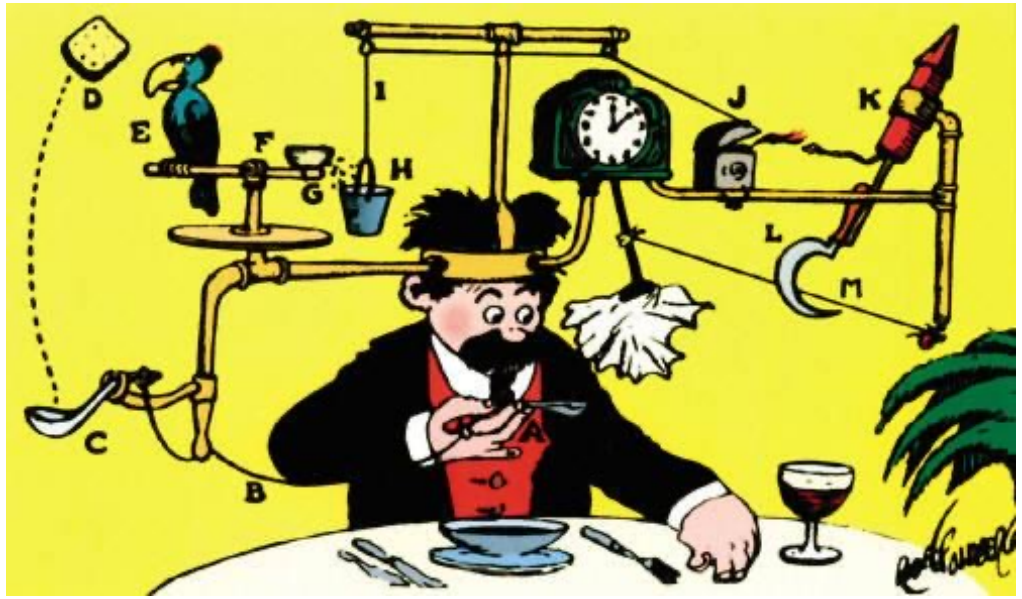
Minimizing Complexity

- Two Approaches
  - Keep it simple
  - Use standards

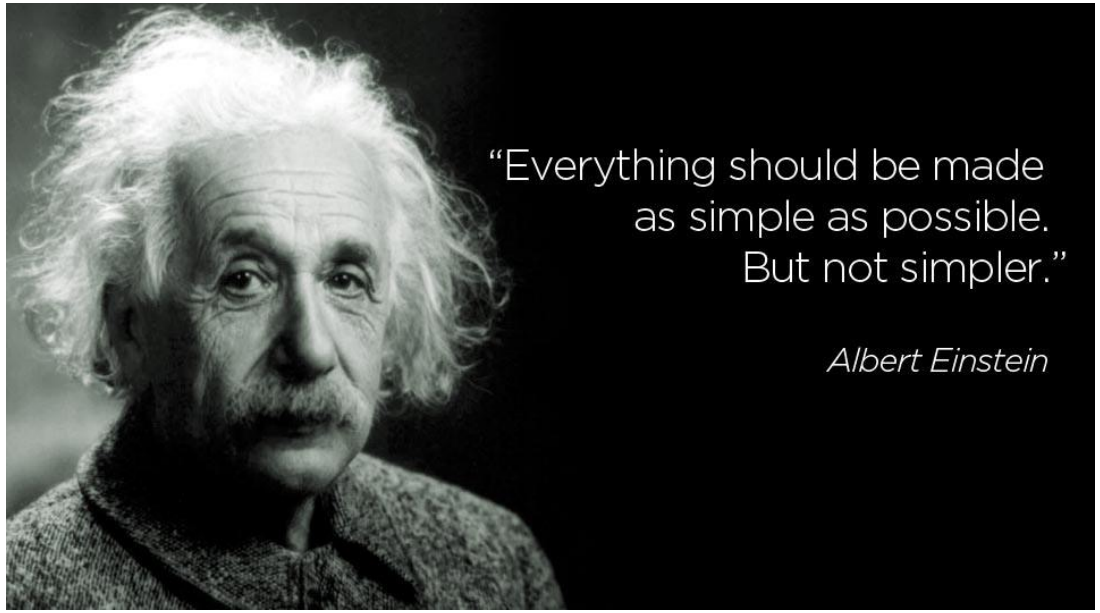


KEEP IT SIMPLE

IS THIS SIMPLE? SELF-OPERATING NAPKIN

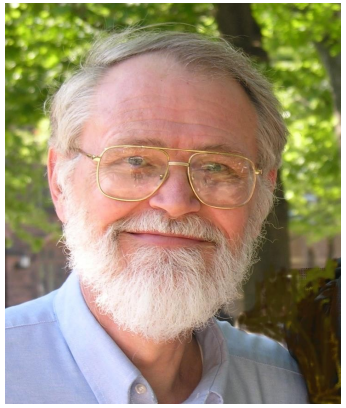


EINSTEIN



MINIMIZING COMPLEXITY

- "Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it." –
  - **Brian Kernighan**

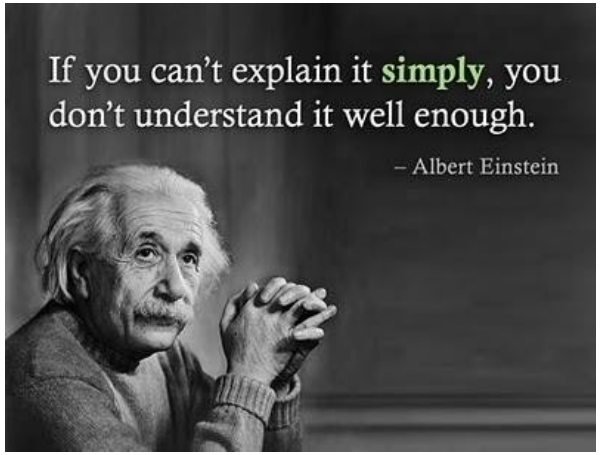


- **"clever" in this context has a negative connotation**

Clever (not really)	Simple
i ^= (i & ~-~i) | (~i & -~i);	i++;

AVOID "CLEVER" CODE

- Avoid language "tricks"
- Avoid cleverness
- Make it simple and readable
- Maintainability is more important than coding

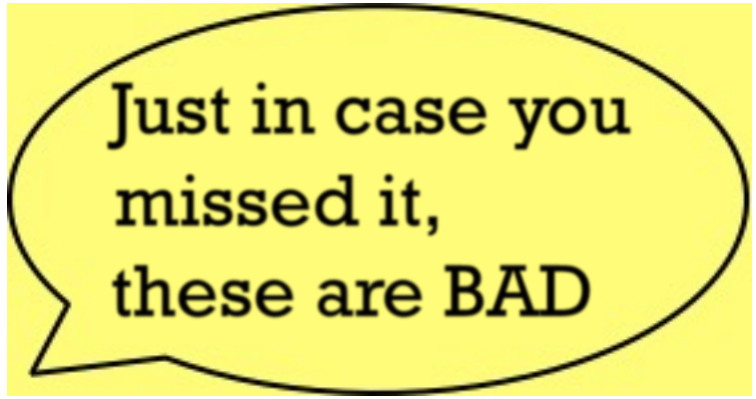


LEARN THROUGH BAD EXAMPLE

- Want to write maintainable code?
  - Study how to do it the wrong way:
  - How To Write Unmaintainable Code by: Roedy Green
    https://tinyurl.com/hfyvqjx

How to Write Unmaintainable Code

- Naming:
  - Buy a copy of a baby naming book, and you'll never be at a loss for variable names.
  - Choose variable names with irrelevant emotional connotation. e.g.: marypoppins = (superman + starship) / god;
- Comments:
  - Lie in the comment
- Testing:
  - Testing is for cowards



MINIMIZING COMPLEXITY

- KISS principle: keep it simple, stupid

Minimizing Complexity: Use Standards

- External standards
  - Coding standards
    - Brace style
    - Naming conventions
  - Technology standards
  - Security standards
- Internal standards
  - Company specific
  - Help with knowledge transfer
  - Creates vernacular (common language)

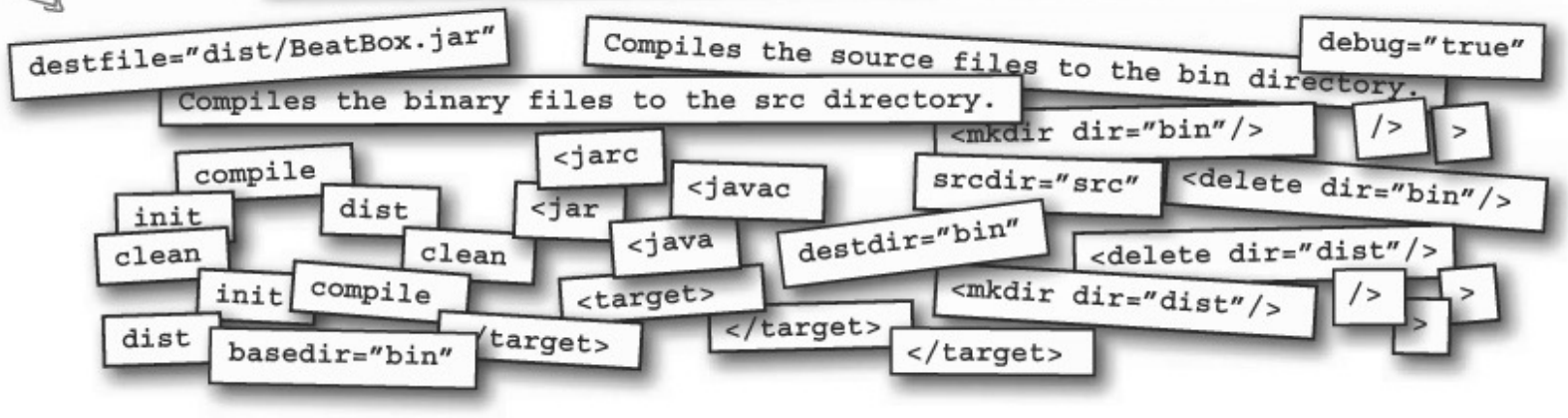
EXERCISE

- Ant files are easier to use—and write—than you think.
- Below is part of a build script, but lots of pieces are missing. It's up to you to use the build magnets at the bottom of the page to complete the build script.

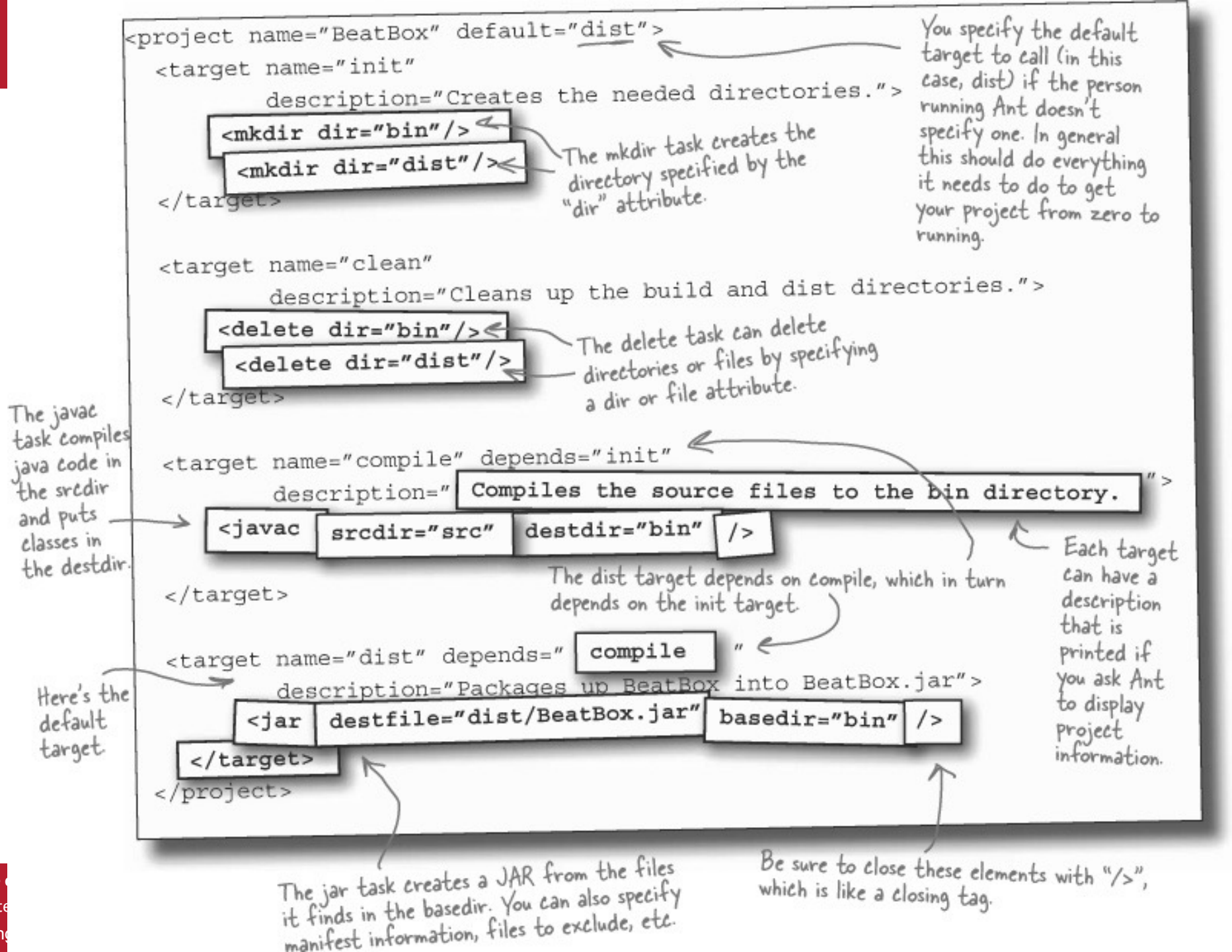
Put the magnets between the target elements to complete the build.xml file.

```
<project name="BeatBox" default="dist">
  <target name="init"
          description="Creates the needed directories.">
    _____
    _____
  </target>

  <target name="clean"
          description="Cleans up the build and dist directories.">
    _____
    _____
  </target>

  <target name="compile" depends="init"
          description="_____">
    _____
    _____
  </target>

  <target name="dist" depends="_____"
          description="Packages up BeatBox into BeatBox.jar">
    _____
    _____

</project>
```

Seems like there are a couple of extra magnets, so be careful.



SOLUTION


<project name="BeatBox" default="dist">
<target name="init"
description="Creates the needed directories.">
<mkdir dir="bin"/>
<mkdir dir="dist"/>
</target>
<target name="clean"
description="Cleans up the build and dist directories.">
<delete dir="bin"/>
<delete dir="dist"/>
</target>
<target name="compile" depends="init"
description="Compiles the source files to the bin directory.">
<javac srcdir="src" destdir="bin" />
</target>
<target name="dist" depends="compile"
description="Packages up BeatBox into BeatBox.jar">
<jar destfile="dist/BeatBox.jar" basedir="bin" />
</target>
</project>
You specify the default target to call (in this case, dist) if the person running Ant doesn't specify one. In general this should do everything it needs to do to get your project from zero to running.
The mkdir task creates the directory specified by the "dir" attribute.
The delete task can delete directories or files by specifying a dir or file attribute.
The javac task compiles java code in the srcdir and puts classes in the destdir.
Each target can have a description that is printed if you ask Ant to display project information.
The dist target depends on compile, which in turn depends on the init target.
Here's the default target.
The jar task creates a JAR from the files it finds in the basedir. You can also specify manifest information, files to exclude, etc.
Be sure to close these elements with "/>", which is like a closing tag.

Retrospective Questions

What does it mean to build your project?

What are build scripts?

What is an ant build file comprised of?

- 4 elements

What do we mean by “clever” code?

- Why is it bad (hint – debugging / finding problems)