MIAMI UNIVERSITY

TESTING OVERVIEW & CONTINUOUS INTEGRATION

Content from Chapter 7 of "Head First Software Development", Pilone et al.

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

Project's Due Dates ?

- ~~April 2nd → 5th week deliverables~~
- ~~April 6th → Iteration 1 presentation → weeks 3 + 4 → 90 mints~~
- April 16th → 6th week deliverables
- April 20th → Iteration 2 presentation → weeks 5 + 6 → 90 mints
- April 23rd → 7th week deliverables
- April 30th → 8th week deliverables
- May 4th → Iteration 3 (final) presentation → weeks 7 + 8 → 90 mints

MIAMI UNIVERSITY

Quiz 4 Review

Q1 & Q2

- Version control systems may attempt to automatically merge changes from two or more sources. Automatic merging is not always possible and not always desirable. Describe the following two situations:
  - Describe a situation that would cause automatic merging to fail. You must use the term "conflict" in your description.
  - Describe a situation that would cause automatic merging to succeed but produce unintended results.

Q1 & Q2: Sample Solution

- Two authors check out the same file, then modify the same line, and then both try to check in the change. The second person will generate a conflict.
- Two authors check out the same file, make modifications to different lines. Say one person changed the signature of a function and the other person creates a call to the same function. Merging will succeed but the code won't compile.

Q3

- Following are commands that are common to many version control systems. Define, in your own words, each command in the space provided. Include in your definition an example of when/how you would use the command and use the terms "working copy" and "repository".
  - Check out:
  - Update:
  - Check in:

Q3: Sample Solution

- Check out: copy code from repository to my local working copy
- Update: update my working copy with most recent version from repository
- Check in: commit changes from my working copy to the repository

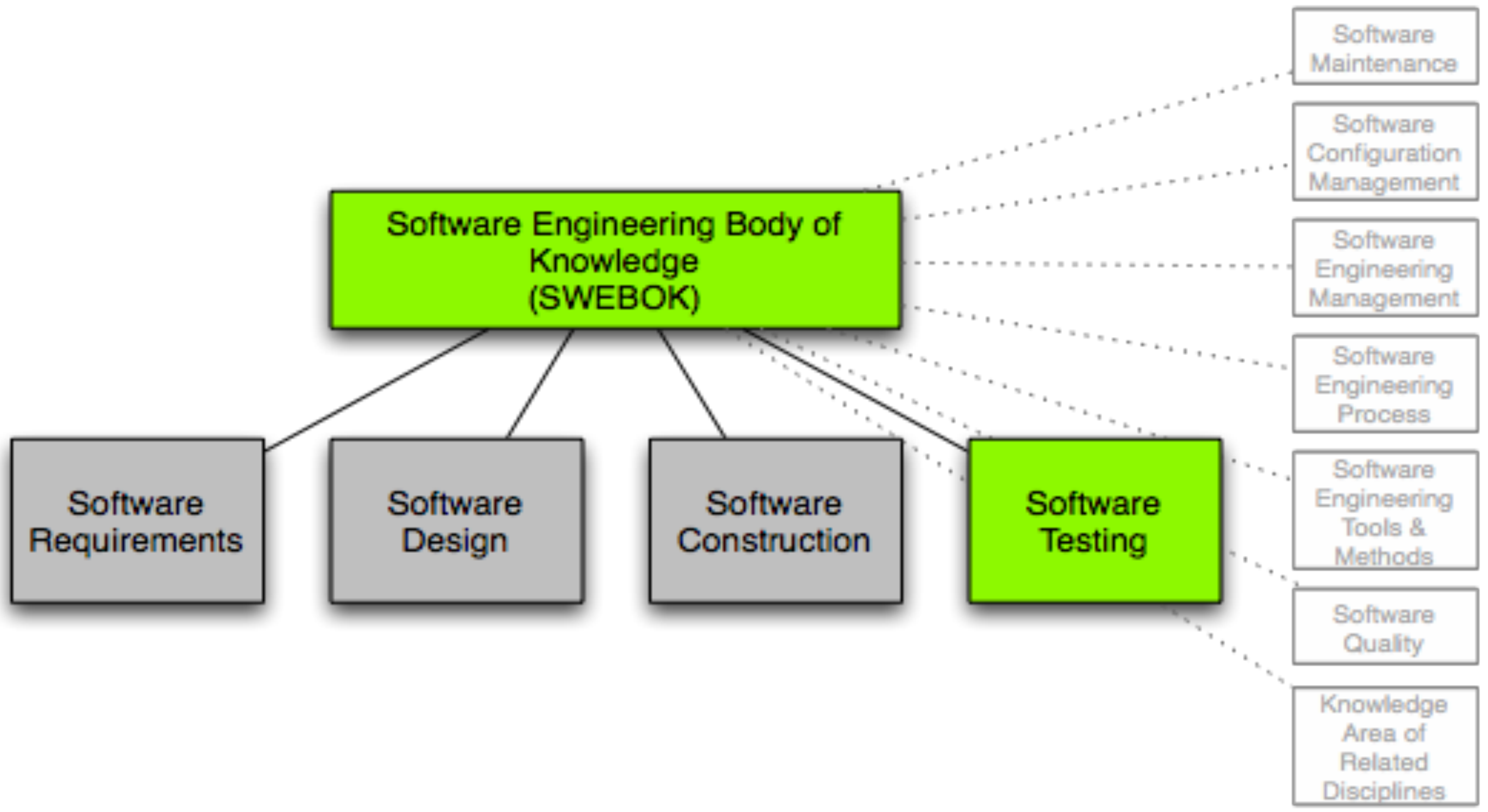
Q4

- Describe the difference between a centralized and a distributed version control system. Include in your description an example of each.

Q4: Sample Solution

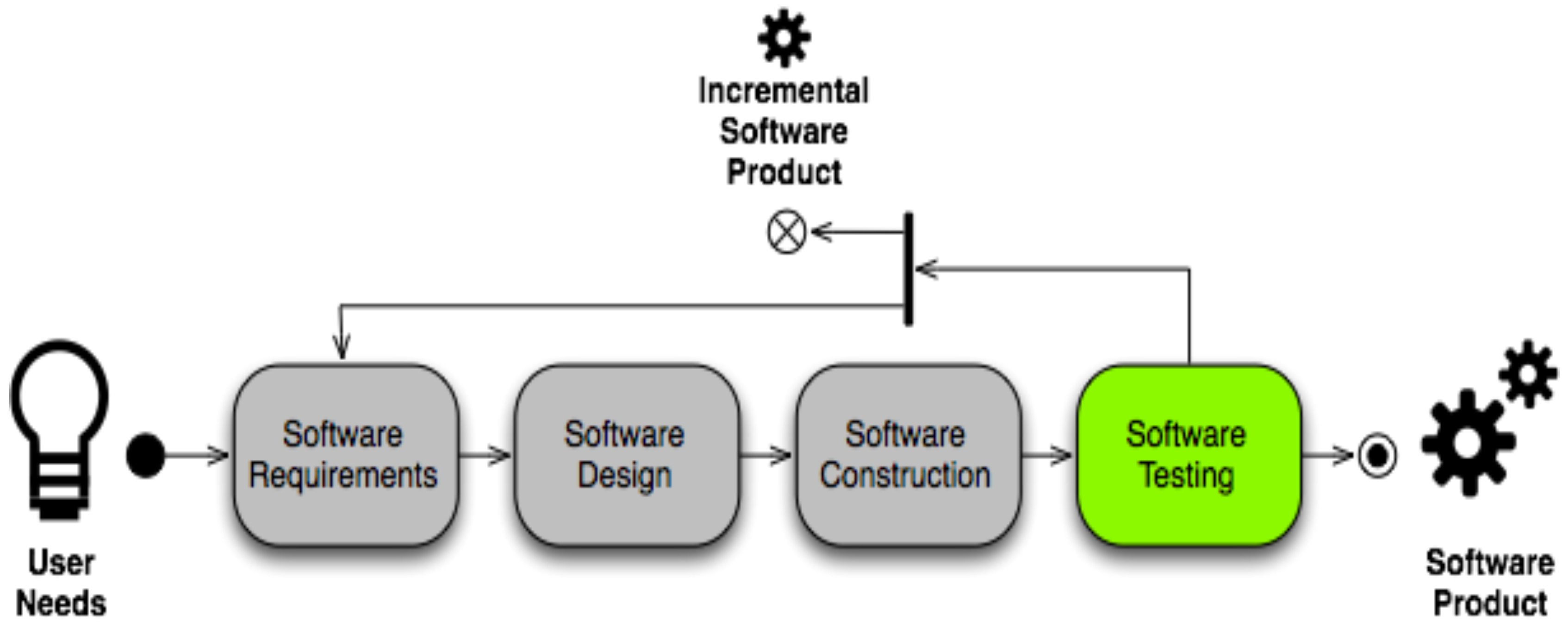
- Subversion or cvs is an example of centralized VCS. Centralized VCS stores the source code in one location. Developers check-out and check-in code from/to one location.
- Git or Mercurial is an example of distributed VCS. Every clone of the repository is itself a full repository complete with history.

Testing (Some From Ch7)

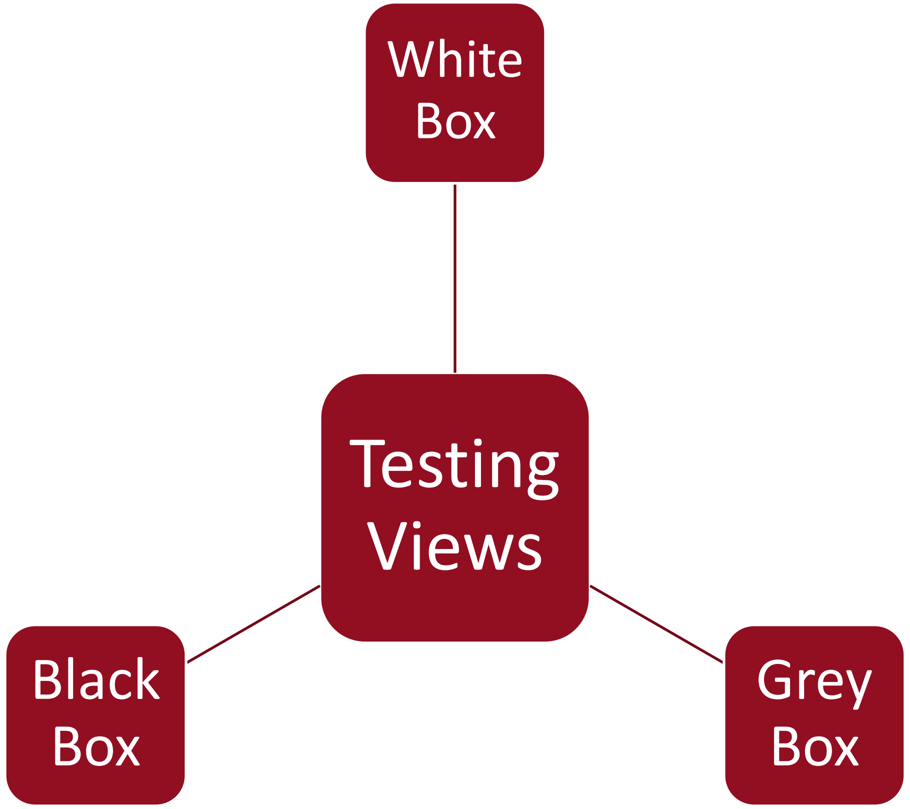

Software Engineering Body of Knowledge (SWEBOK)
Software Requirements
Software Design
Software Construction
Software Testing
Software Maintenance
Software Configuration Management
Software Engineering Management
Software Engineering Process
Software Engineering Tools & Methods
Software Quality
Knowledge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

MIAMI UNIVERSITY

Testing (Some From Ch7)



3 Ways Of Looking At Our System



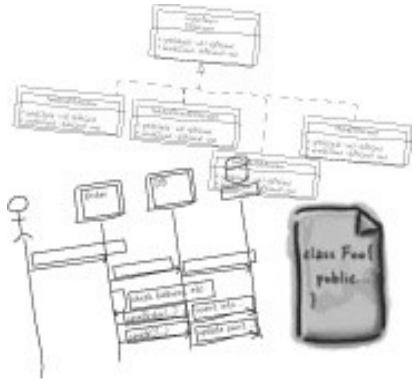
BLACK BOX VIEW

- Users see the system from the outside
- Your users don't see your code,
  - they don't look at the database tables, they don't evaluate your algorithms...and generally they don't want to.
- Your system is a black box to them; it either does what they asked it to do, or it doesn't.
- Users are all about functionality.

Grey Box View

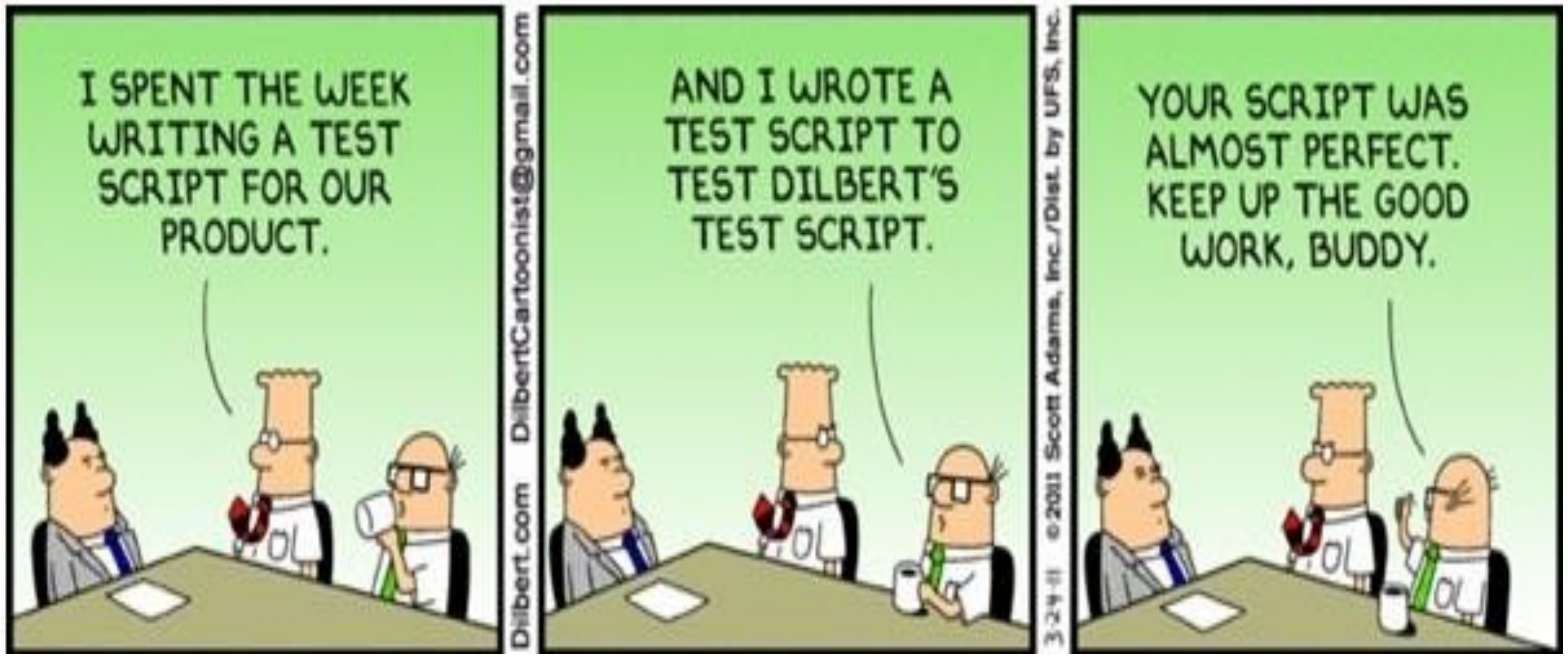
- Testers peek under the covers a little
- Testers are a different breed.
  - They're looking for functionality, but they're usually poking underneath to make sure things are really happening the way you said.
- Your system is more of a grey box to them.
  - Testers are probably looking at the data in your database to make sure things are being cleaned up correctly;
  - They might be checking that memory usage is staying steady.

WHITE BOX VIEW



- Developers are deep in the trenches
- Developers are in the weeds.
  - They see good (and sometimes bad) class design, patterns, duplicated code, inconsistencies in how things are represented. The system is wide open to them.
- If users see a system as a closed black box, developers see it as an open white box.
  - But sometimes because developers see so much detail, it's possible for them to miss broken functionality or make an assumption that a tester or end user might not.

Introduction to Testing


I SPENT THE WEEK WRITING A TEST SCRIPT FOR OUR PRODUCT.
AND I WROTE A TEST SCRIPT TO TEST DILBERT'S TEST SCRIPT.
YOUR SCRIPT WAS ALMOST PERFECT. KEEP UP THE GOOD WORK, BUDDY.
Dilbert.com DilbertCartoonist@gmail.com
3-29-11 ©2011 Scott Adams, Inc./Dist. by UFS, Inc.

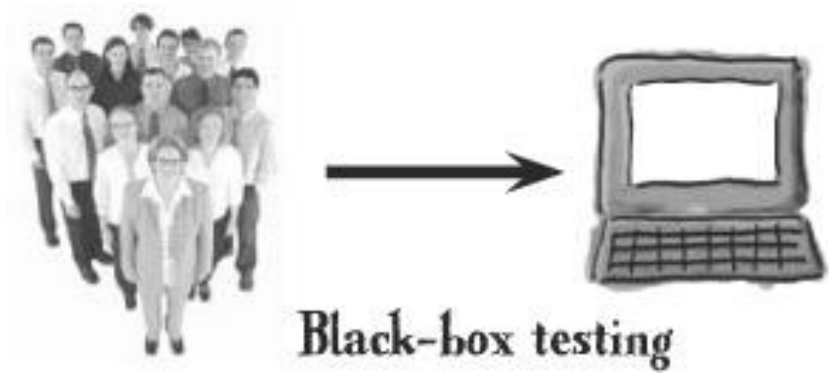
Types of Testing

- Black box testing
- Grey Box [TEXTBOOK]
- Glass box testing (aka White box)
- Unit testing
- Regression testing

Black Box Testing

- black box: "a system or component whose inputs, outputs, and general function are known but whose contents or implementation are unknown or irrelevant" [GLOSSARY]

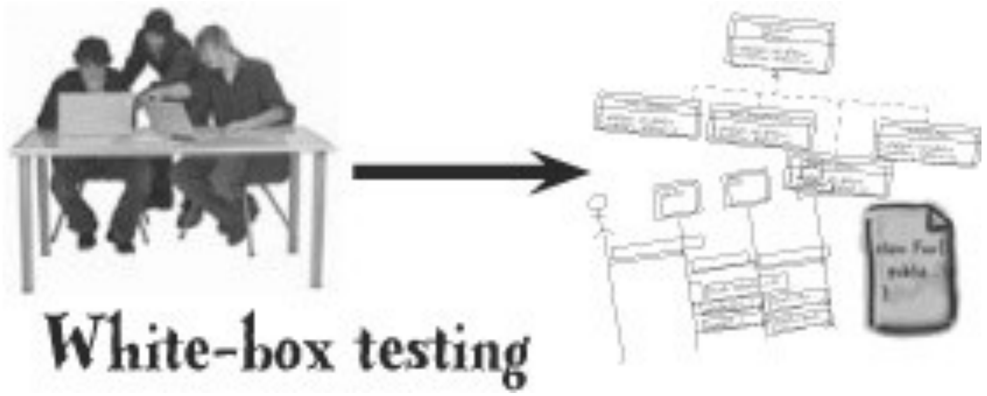
- "test cases rely only on the input/output behavior" [SWEBOK]



Glass (White) Box Testing

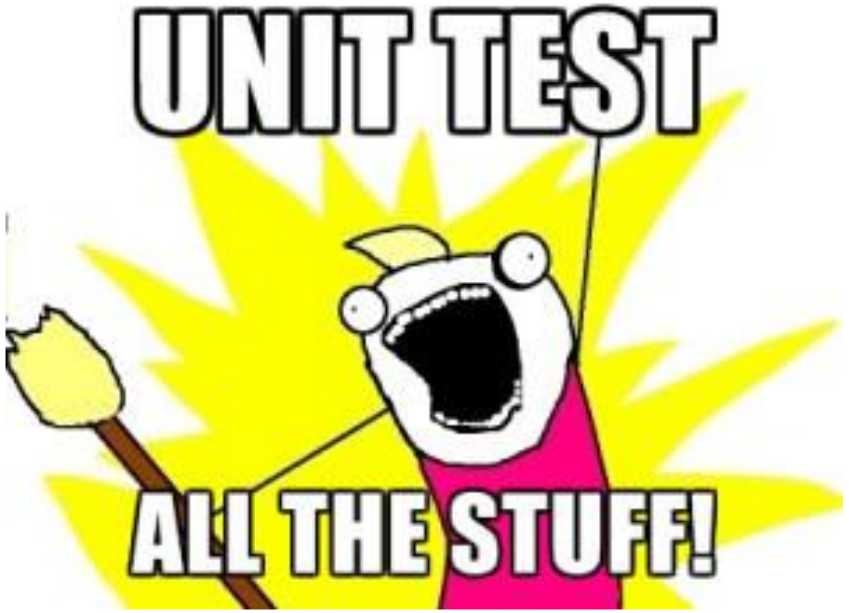
- glass box: "a system or component whose internal contents or implementation are known" [GLOSSARY]

- "tests rely on information about how the software has been designed or coded"



Unit Test

- unit test: "testing of individual routines and modules by the developer or an independent tester" [GLOSSARY]

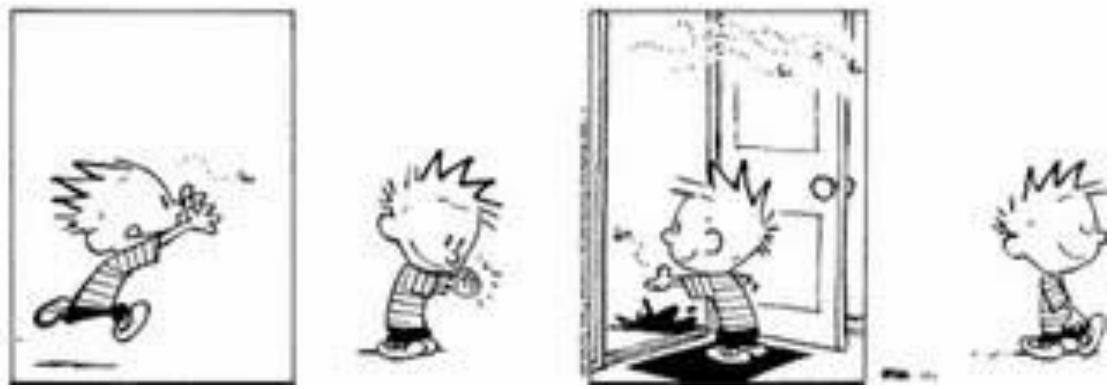


MIAMI UNIVERSITY

REGRESSION TESTING

- regression test: "retesting to detect faults introduced by modification" [GLOSSARY]

Regression:
"when you fix one bug, you introduce several newer bugs."



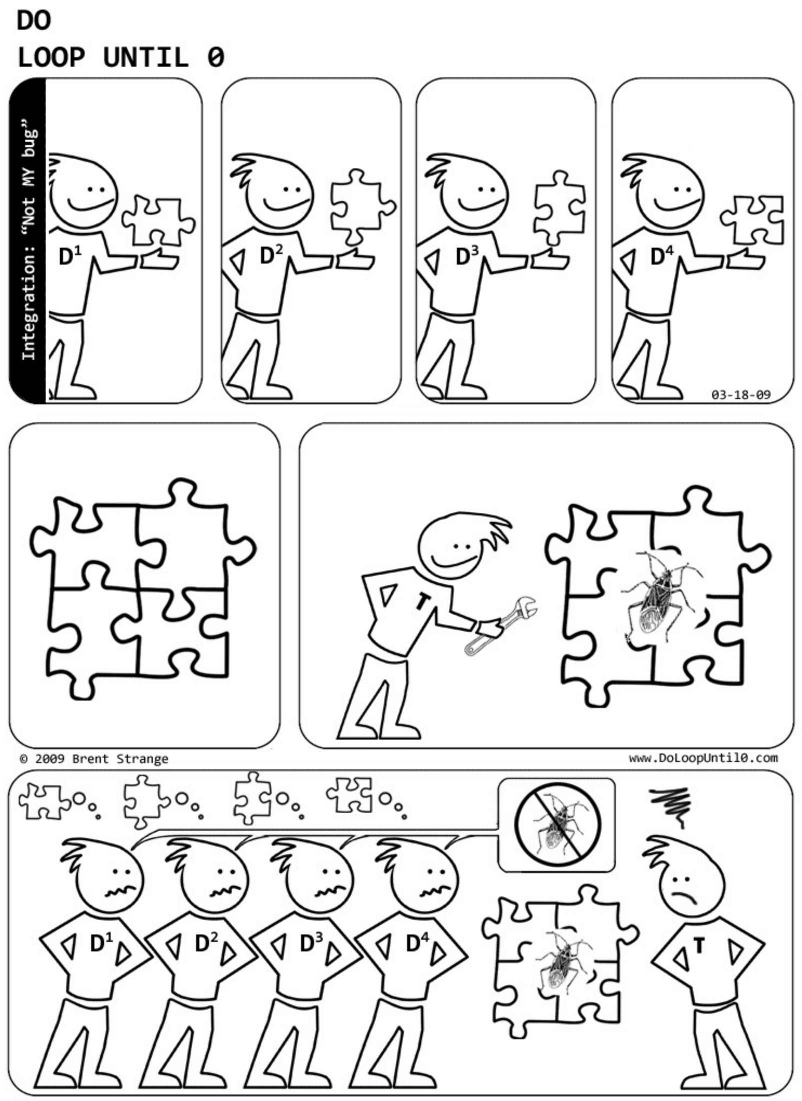
TEST AUTOMATION

- Test Suite:
  - Library of unit tests
  - All tests run at once
  - One command
- Testing Frameworks
  - Hundreds?
    - JUnit
      - JUnit is built into Eclipse

WHERE TO TEST

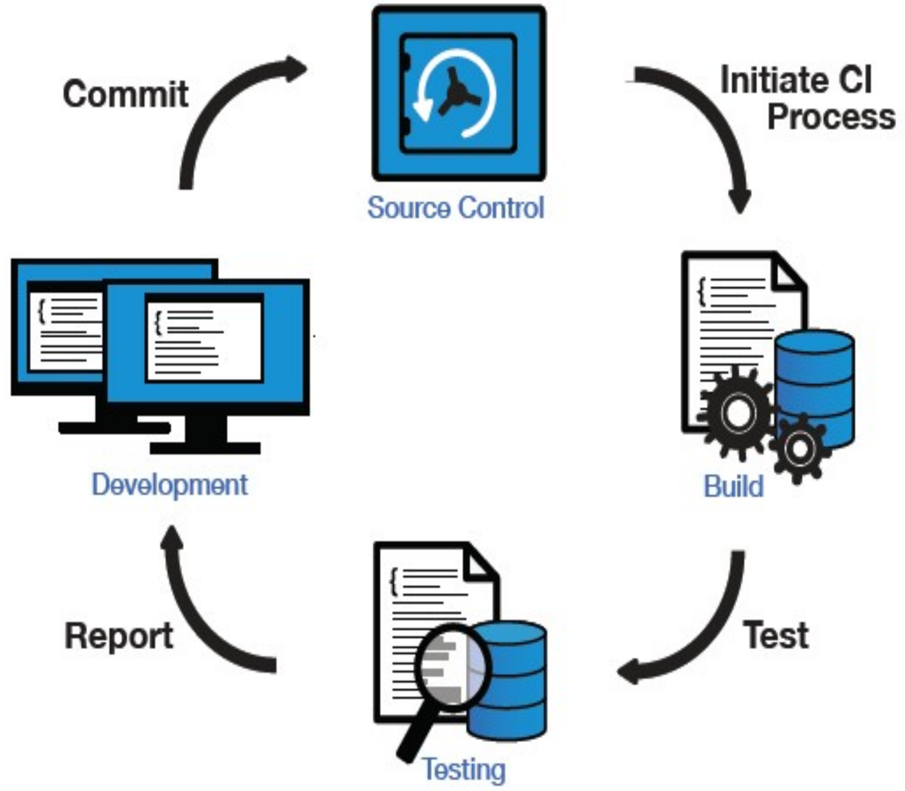
- Testing Considered at Every Phase
  - Requirements (verification)
  - Design
  - Construction

Integration (Testing)



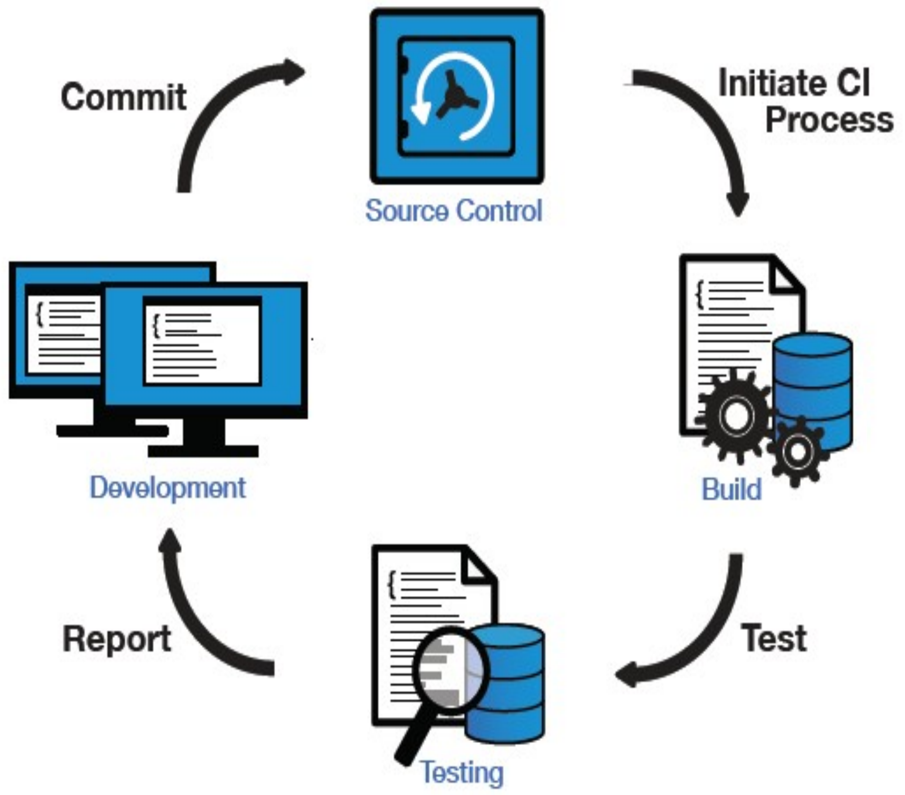
Continuous Integration (CI)

- CI Wraps into one process:
  - version control
  - compilation
  - testing



CONTINUOUS INTEGRATION

- Not a new idea (Martin Fowler wrote about it in 2006 for example)
  - https://martinfowler.com/articles/continuousIntegration.html
- However, continually growing and "desired"/"required" skill in industry
  - Frequently brought up by our Miami Alumni and Industry Advisers as skills they want their developers to know/use/have.
  - Google, Kroger, EliLilly, andmore.

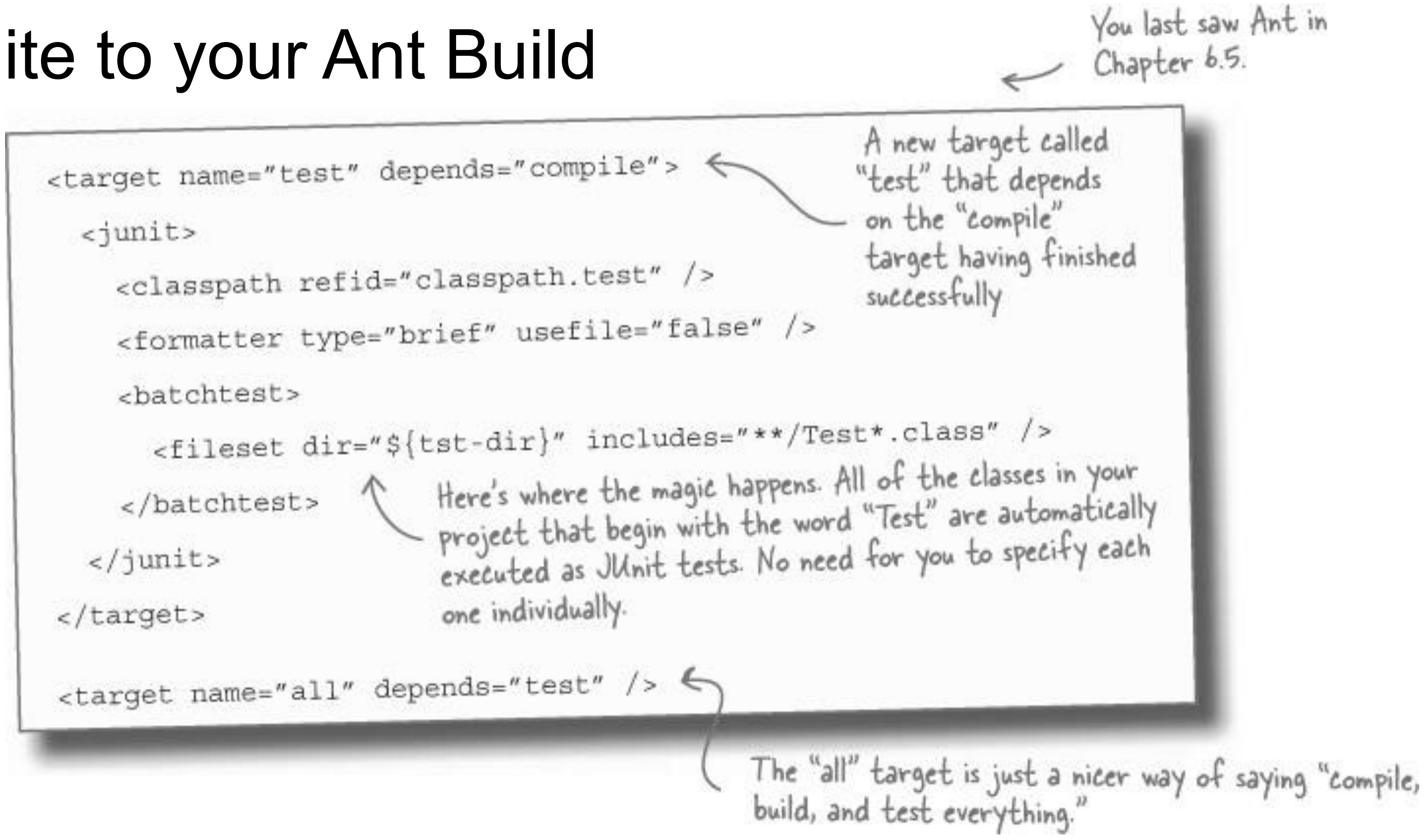


Continuous Integration

- Cruise Control
  - http://cruisecontrol.sourceforge.net/

Example: Cruise Control

- Add your Junit test suite to your Ant Build



You last saw Ant in Chapter 6.5.

```
<target name="test" depends="compile">
  <junit>
    <classpath refid="classpath.test" />
    <formatter type="brief" usefile="false" />
    <batchtest>
      <fileset dir="${tst-dir}" includes="**/Test*.class" />
    </batchtest>
  </junit>
</target>

<target name="all" depends="test" />
```

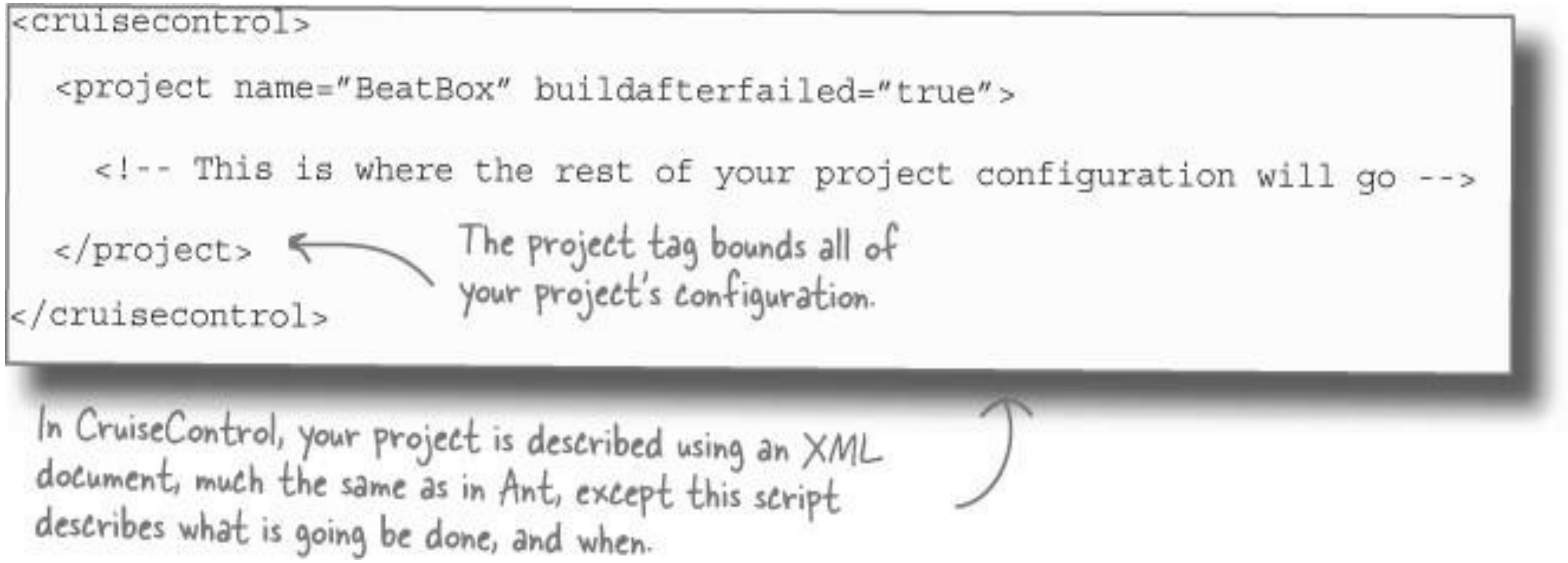
A new target called "test" that depends on the "compile" target having finished successfully

Here's where the magic happens. All of the classes in your project that begin with the word "Test" are automatically executed as JUnit tests. No need for you to specify each one individually.

The "all" target is just a nicer way of saying "compile, build, and test everything."

Example: Cruise Control

- Create your CruiseControl project

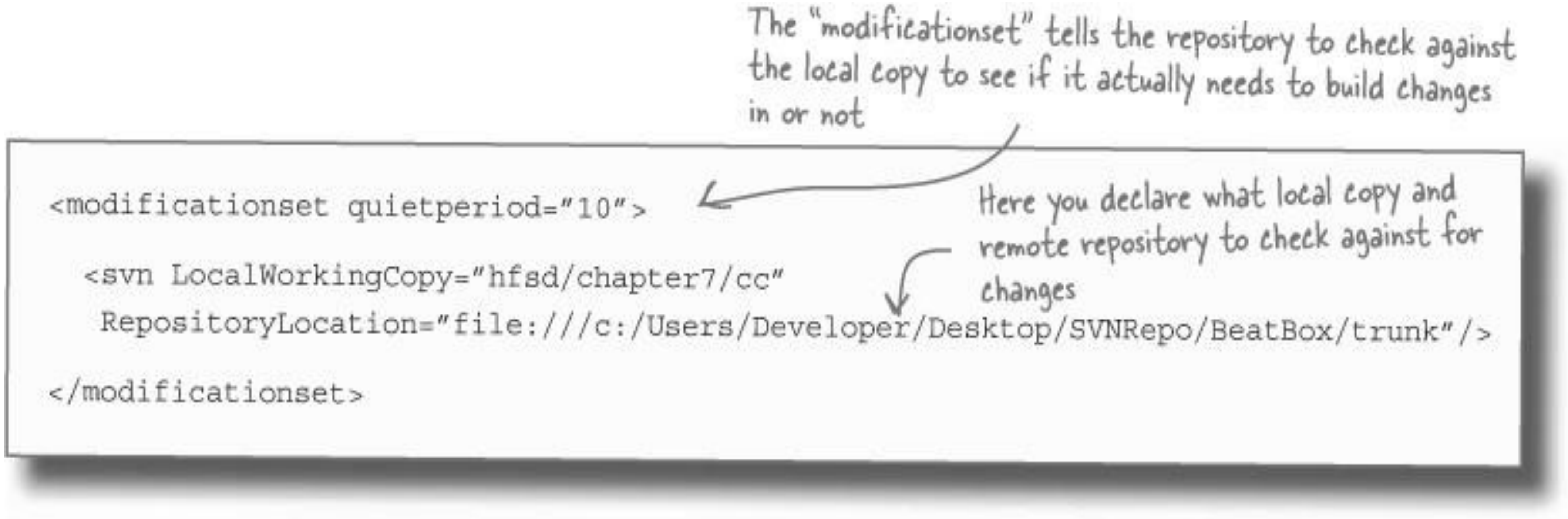


```
<cruisecontrol>

  <project name="BeatBox" buildafterfailed="true">

    <!-- This is where the rest of your project configuration will go -->

  </project>

</cruisecontrol>
```

The project tag bounds all of your project's configuration.
In CruiseControl, your project is described using an XML document, much the same as in Ant, except this script describes what is going be done, and when.

Example: Cruise Control

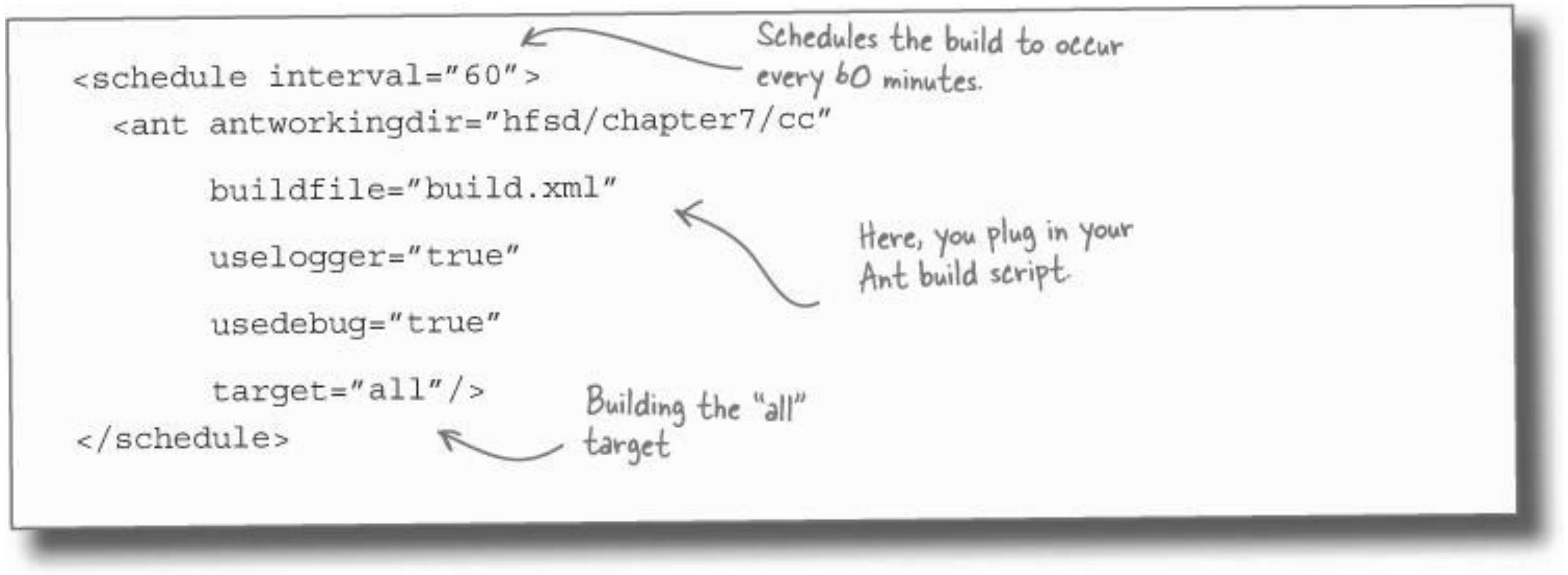
- Check Repository and get latest code


The "modificationset" tells the repository to check against the local copy to see if it actually needs to build changes in or not
Here you declare what local copy and remote repository to check against for changes

```
<modificationset quietperiod="10">
  <svn LocalWorkingCopy="hfsd/chapter7/cc"
   RepositoryLocation="file:///c:/Users/Developer/Desktop/SVNRepo/BeatBox/trunk"/>
</modificationset>
```

EXAMPLE: CRUISE CONTROL

- Schedule the build

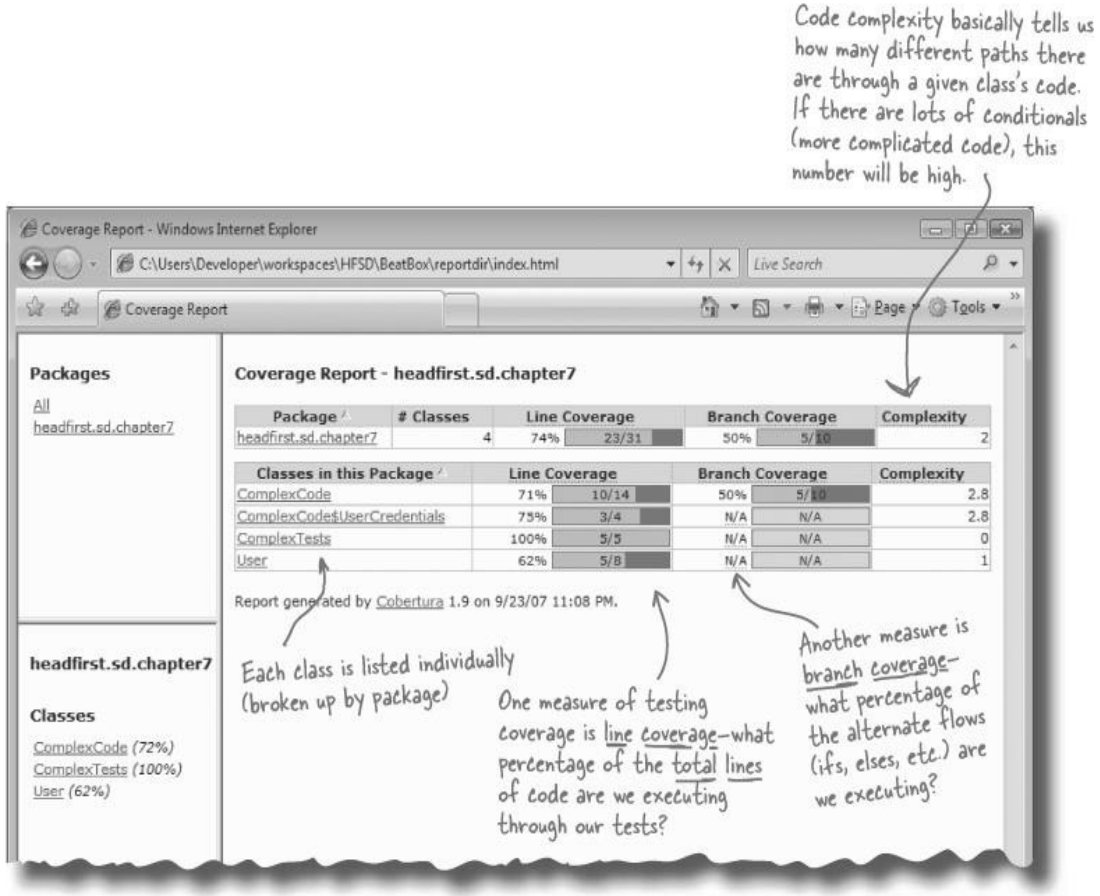


```
<schedule interval="60">
  <ant antworkingdir="hfsd/chapter7/cc"
       buildfile="build.xml"
       uselogger="true"
       usedebug="true"
       target="all"/>
</schedule>
```

Schedules the build to occur every 60 minutes.
Here, you plug in your Ant build script.
Building the "all" target

Test Coverage Tools

- Most coverage tools—especially ones like CruiseControl that integrate with other CI and version control tools—can generate a report telling you how much of your code is covered.


Coverage Report - Windows Internet Explorer
C:\Users\Developer\workspaces\HFSD\BeatBox\reportdir\index.html
Live Search
Coverage Report
Page
Tools

Code complexity basically tells us how many different paths there are through a given class's code. If there are lots of conditionals (more complicated code), this number will be high.

Packages

All
headfirst.sd.chapter7

Coverage Report - headfirst.sd.chapter7

Package	# Classes	Line Coverage		Branch Coverage		Complexity
headfirst.sd.chapter7	4	74%	23/31	50%	5/10	2

Classes in this Package	Line Coverage		Branch Coverage		Complexity
ComplexCode	71%	10/14	50%	5/10	2.8
ComplexCode$UserCredentials	75%	3/4	N/A	N/A	2.8
ComplexTests	100%	5/5	N/A	N/A	0
User	62%	5/8	N/A	N/A	1

Report generated by Cobertura 1.9 on 9/23/07 11:08 PM.

headfirst.sd.chapter7

Classes

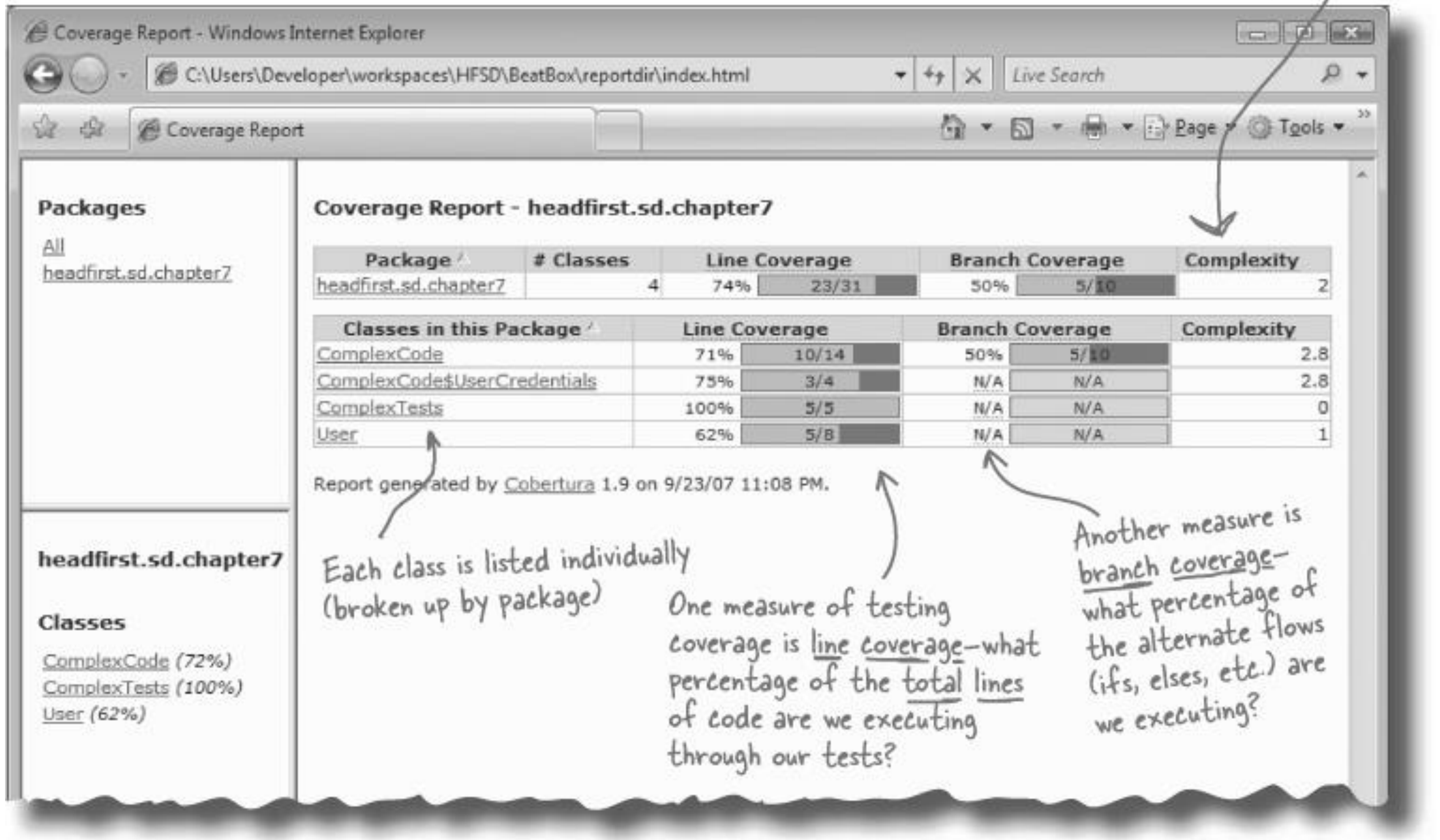
ComplexCode (72%)
ComplexTests (100%)
User (62%)

Each class is listed individually (broken up by package)

One measure of testing coverage is line coverage—what percentage of the total lines of code are we executing through our tests?

Another measure is branch coverage—what percentage of the alternate flows (ifs, elses, etc.) are we executing?

Code complexity basically tells us how many different paths there are through a given class's code. If there are lots of conditionals (more complicated code), this number will be high.



Coverage Report - Windows Internet Explorer

C:\Users\Developer\workspaces\HFSD\BeatBox\reportdir\index.html

Packages

All
headfirst.sd.chapter7

headfirst.sd.chapter7

Classes

ComplexCode (72%)
ComplexTests (100%)
User (62%)

Coverage Report - headfirst.sd.chapter7

Package	# Classes	Line Coverage		Branch Coverage		Complexity
headfirst.sd.chapter7	4	74%	23/31	50%	5/10	2

Classes in this Package	Line Coverage		Branch Coverage		Complexity
ComplexCode	71%	10/14	50%	5/10	2.8
ComplexCode$UserCredentials	75%	3/4	N/A	N/A	2.8
ComplexTests	100%	5/5	N/A	N/A	0
User	62%	5/8	N/A	N/A	1

Report generated by Cobertura 1.9 on 9/23/07 11:08 PM.

Each class is listed individually (broken up by package)

One measure of testing coverage is line coverage—what percentage of the total lines of code are we executing through our tests?

Another measure is branch coverage—what percentage of the alternate flows (ifs, elses, etc.) are we executing?

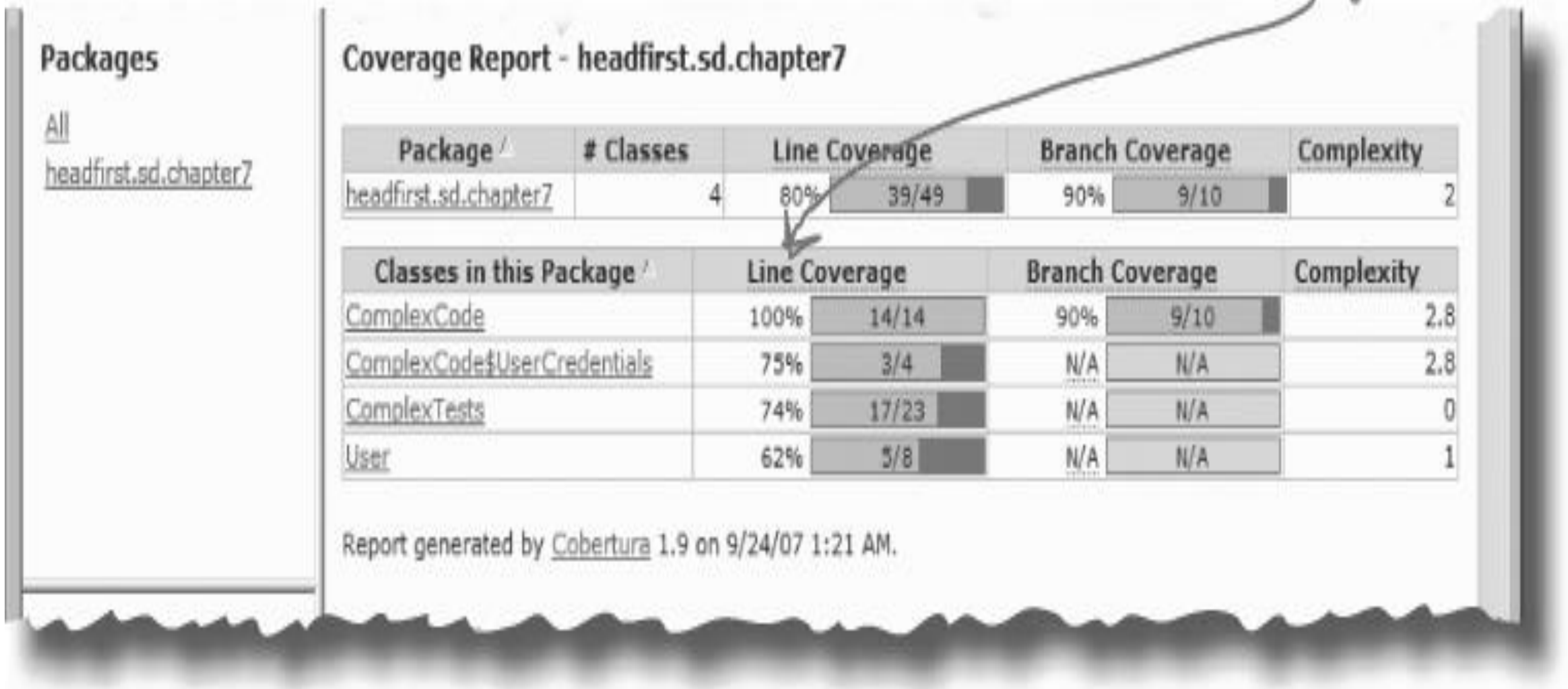
Reality

- In general, it’s not practical to always hit 100% coverage.
- You’ll get diminishing returns on your testing after a certain point.
  - For most projects, aim for about 85%–90% coverage.
  - More often than not, it’s just not possible to tease out that last 10%–15% of coverage.
  - In other cases, it’s possible but just far too much work to be worth the trouble.

MIAMI UNIVERSITY

Reality

Add in the failure cases and we're in much better shape with the ComplexCode class. Still need work on the User class though...



Packages

All
headfirst.sd.chapter7

Coverage Report - headfirst.sd.chapter7

Package	# Classes	Line Coverage		Branch Coverage		Complexity
headfirst.sd.chapter7	4	80%	39/49	90%	9/10	2

Classes in this Package	Line Coverage		Branch Coverage		Complexity
ComplexCode	100%	14/14	90%	9/10	2.8
ComplexCode$UserCredentials	75%	3/4	N/A	N/A	2.8
ComplexTests	74%	17/23	N/A	N/A	0
User	62%	5/8	N/A	N/A	1

Report generated by Cobertura 1.9 on 9/24/07 1:21 AM.

RETROSPECTIVE QUESTIONS

- Three different views of Testing? Definitions?
- Unit Test
- Regression Tests
- Integration Testing
- Continuous Integration (3 elements?)