Updated: 2021-03-31

6.6
Software Engineering

Northeastern
University

Some topics—but we will only touch a few of these

- Teamwork:
  - Project planning
  - Requirements (use cases)
  - Agile development
  - Acceptance Criteria
  - Source-code management
  - Practical dependency management
  - Technical Debt
  - Peer reviews/programming
  - Issue Tracking (JIRA, etc.)
- Eco-systems:
  - Platforms (JDK vs. .Net, etc.)
  - O/S (Windows, Unix, …)
  - Open source vs. proprietary
  - Licensing
  - Messaging/streaming: pub/sub
  - Persistence
- Complexity
  - Managing dependencies, coupling, etc.
  - Optimization: performance vs. memory
  - Code analysis
  - Privacy, security, cryptography
  - Big data/parallel programming/concurrent programming

Agile Development

The *Agile Manifesto* is based on twelve principles:

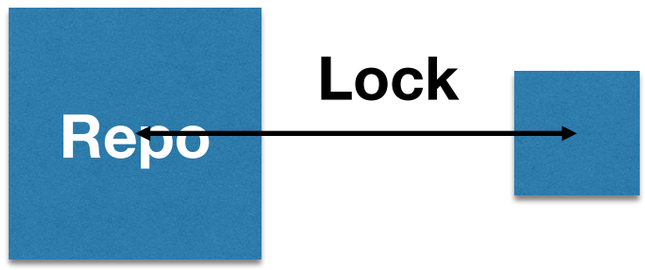
1. **Customer satisfaction by early/continuous delivery of valuable software**
2. **Welcome changing requirements, even in late development**
3. Working software is delivered frequently (weeks rather than months)
4. Close, daily cooperation between business people and developers
5. Projects are built around motivated individuals, who should be trusted
6. Face-to-face conversation is the best form of communication (co-location)
7. **Working software is the principal measure of progress**
8. Sustainable development, able to maintain a constant pace
9. Continuous attention to technical excellence and good design
10. Simplicity—the art of maximizing the amount of work not done—is essential
11. **Best architectures, requirements, and designs emerge from self-organizing teams**
12. Regularly, the team reflects on how to become more effective, and adjusts accordingly

Agile is built on…

- User Stories (c.f. requirements):
  - **Who**, **What**, **Why**?
  - As a *<role>*, I want *<goal>* so that *<benefit>.*
- Iterative Development:
  - Sprints
  - Scrum
  - Velocity
- Continuous integration:
  - User-centric source control (git, Mercurial)
  - **Unit testing (without unit testing, CI cannot work!!!!!)**
  - Continuous integration server, e.g. CI, Jenkins

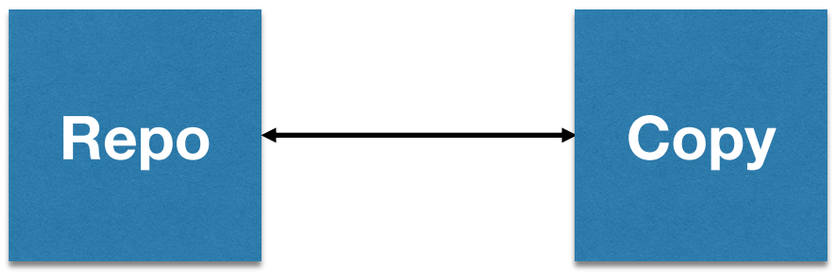
Source Code Management

- "Old" SCM:
  - Subversion
  - CVS



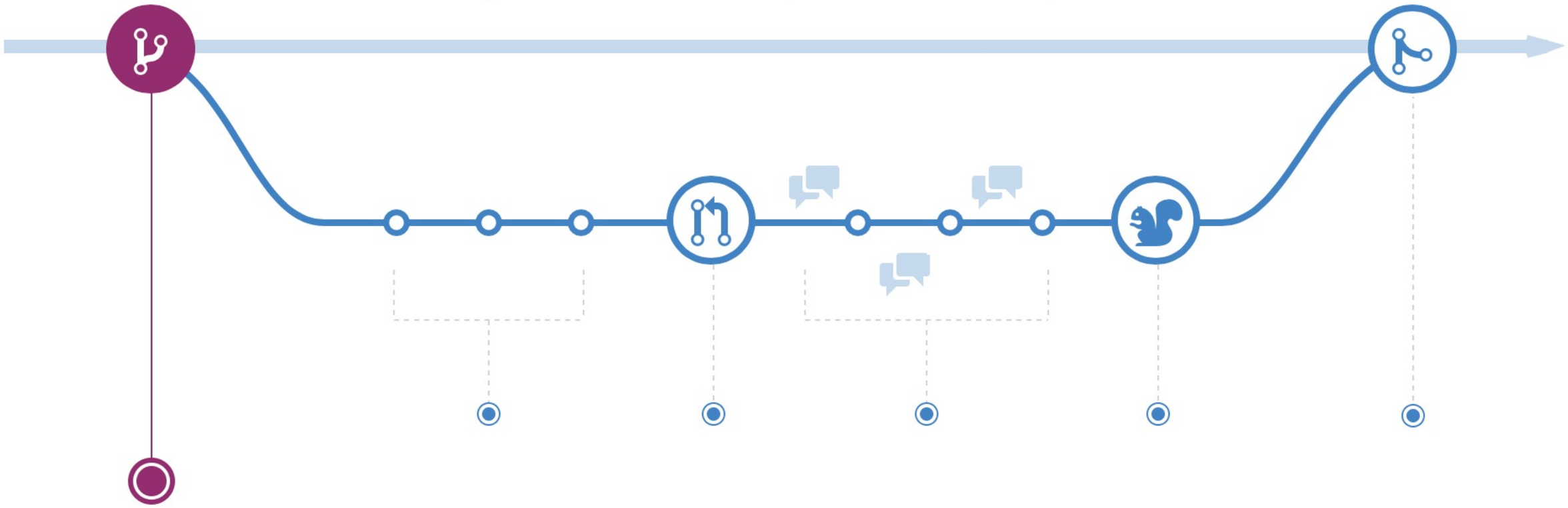
User downloads only those files to be edited

- User-centric SCM:
  - Git
  - Mercurial



User downloads entire repository

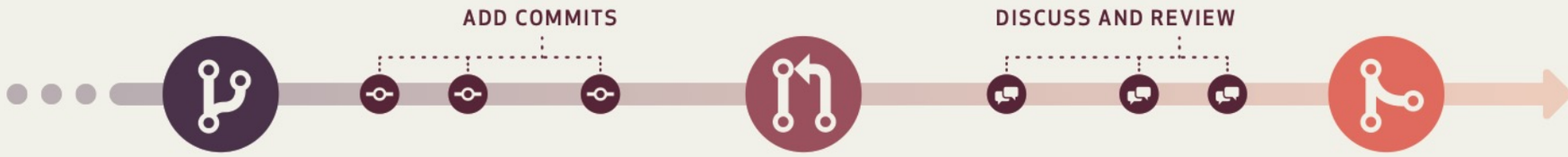
Git workflow



- GitHub Flow
  - Development branches
  - Pull Requests

WORK FAST
WORK SMART
THE GITHUB FLOW

The GitHub Flow is a lightweight, branch-based workflow that's great for teams and projects with regular deployments. Find this and other guides at **http://guides.github.com/**.



CREATE A BRANCH

Create a branch in your project where you can safely experiment and make changes.

OPEN A PULL REQUEST

Use a pull request to get feedback on your changes from people down the hall or ten time zones away.

MERGE AND DEPLOY

Merge your changes into your master branch and deploy your code.

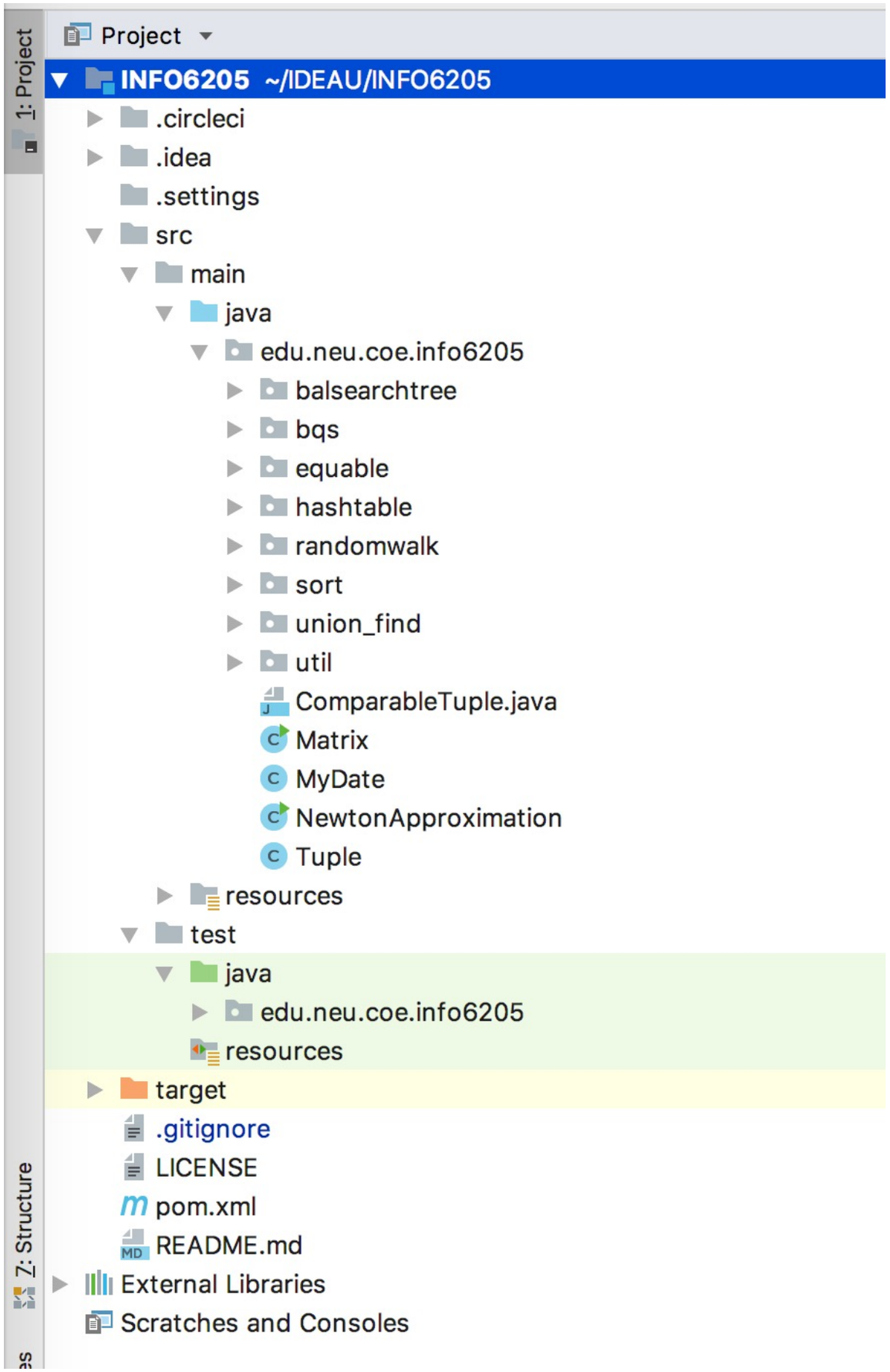
GitHub is the best way to build software together.

GitHub provides tools for easier collaboration and code sharing from any device. Start collaborating with millions of developers today!

Dependencies

- The days of building entire applications with all your own software are long gone
  - (yes, those days did exist!)
- Dependencies must be “managed” by something.
  - *Maven* is both a dependency management system and a build system; q.v. *Ivy, Gradle*, etc.
  - *sbt* (for Scala) is a build system which uses *Ivy* underneath
  - Dependencies are retrieved from an artifact repository (“artifactory”) which is either “central” or local (company, department, etc.).
  - The actual dependencies (with group/artifact/version details) are specified in a *POM* file (“Project Object Model”) or its equivalent.

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>sf.net.darwin</groupId>
    <artifactId>darwin</artifactId>
    <version>3.0.0-SNAPSHOT</version>
    <name>Darwin API</name>
    <description>Darwin Parent Project</description>
    <scm>
        <developerConnection>scm:svn:http://darwin.svn.sourceforge.net/svnroot/darwin</developerConnection>
        <url>http://darwin.svn.sourceforge.net/svnroot/darwin</url>
    </scm>
    <organization>
        <name>Rubecula Software, LLC</name>
        <url>http://www.rubecula.com/</url>
    </organization>
    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.3.2</version>
                <configuration>
                    <source>1.6</source>
                    <target>1.6</target>
                </configuration>
            </plugin>
                <plugin>
                    <groupId>org.codehaus.mojo</groupId>
                    <artifactId>buildnumber-maven-plugin</artifactId>
                    <version>1.1</version>
                    <executions>
                        <execution>
                            <phase>validate</phase>
                            <goals>
                                <goal>create</goal>
                            </goals>
                        </execution>
                    </executions>
                    <configuration>
                        <doCheck>false</doCheck>
                        <doUpdate>false</doUpdate>
                    </configuration>
                </plugin>
                <plugin>
                    <groupId>org.apache.maven.plugins</groupId>
                    <artifactId>maven-assembly-plugin</artifactId>
                    <version>2.2.1</version>
                    <configuration>
                        <descriptorRefs>
                            <descriptorRef>jar-with-dependencies</descriptorRef>
                        </descriptorRefs>
                        <finalName>${project.build.finalName}</finalName>
                        <archive>
                            <manifest>
                                <addClasspath>true</addClasspath>
                                <mainClass>net.sf.darwin.platform.main.DarwinPlatform</mainClass>
                            </manifest>
                        </archive>
                    </configuration>
                    <executions>
                        <execution>
                            <id>make-assembly</id> <!-- this is used for inheritance merges -->
                            <phase>package</phase> <!-- bind to the packaging phase -->
                            <goals>
                                <goal>single</goal>
                            </goals>
                        </execution>
                    </executions>
                </plugin>
        </plugins>
    </build>
    <packaging>pom</packaging>
</project>
```


1: Project
Project
INFO6205 ~/IDEAU/INFO6205
.circleci
.idea
.settings
src
main
java
edu.neu.coe.info6205
balsearchtree
bqs
equable
hashtable
randomwalk
sort
union_find
util
ComparableTuple.java
Matrix
MyDate
NewtonApproximation
Tuple
resources
test
java
edu.neu.coe.info6205
resources
target
.gitignore
LICENSE
pom.xml
README.md
External Libraries
Scratches and Consoles
7: Structure

Merging

- It happens sooner or later: you and another developer have changed the same module.
- One of you will have to resolve the conflicts. Your IDE or git client will help with this by presenting a three-way merge:
  - *common-ancestor—theirs—yours.*
- Unfortunately, these merge clients are still (after all these years!!!) line-oriented rather than token/block-oriented.
- Nevertheless, you should be able to resolve the conflict unless each of you has essentially changed the same part of the code in different ways.

What goes into source control?

- Put everything that you need—and nothing that you don't need—into the repository.
  - Guidelines for any (source) item:
    - represents IP (intellectual property—requires brainpower to create);
    - cannot be trivially derived from other sources;
    - experience versioning (typically);
    - relatively small (e.g. do not commit your entire data source(s) to git);
    - host-independent (builds should work on any machine);
    - usually in ASCII or Unicode or some other alphanumeric, human-readable encoding;
    - does not pertain to any particular IDE or is removed by clean:
      - files ending in, e.g., *.class*, *.jar*, *.classpath*, *.iml*; *.idea* plus anything under *project* or *target* should be ignored (using *.gitignore* assuming you're using git) and not be placed in the repo.

What other project models are there?

- The primary alternative to agile is “Waterfall”
  - This was the primary project management methodology for many years. It doesn’t work (never did!)
  - Why doesn’t it work?
    - Time lag, hysteresis induced by:
      - the development/testing cycle
      - changing requirements
    - Reality (*The Mythical Man Month*)
  - Why was it used? They didn’t know any better—no unit testing.

Testing:

Unit, integration and performance specifications

Our style of development

- What do you notice about the way you are being asked to do the assignments?

- We are essentially using a particular development style. Do you know what it’s called?
  - Hint: it has a TLA (three-letter acronym)

Why *do* we create specifications/unit tests?

- You can prove a *fragment* of code…
  - …but it's impractical to prove an entire code base…
  - …so, we aim for code coverage:
  - One of the ways of maximizing coverage is by using random inputs (that's why the random number generator is important)—*property-based testing*.
- Unit tests prevent you from making errors in:
  - bug fixing
  - feature implementation
  - re-factoring
- Unit tests serve as *documentation* (better than comments) on the behavior of software
- Integration tests help find:
  - sloppy APIs, contracts, etc.
  - unacceptable performance

Testing

- Unit Testing is one of the most important developments in software engineering
- Yet, it is still not done properly!!
  - When you go on an interview, ask *them* about their unit testing practices
  - Writing unit tests:
    - Test-driven development
    - Automated test writing
    - Mocking
    - Specifications and other styles
    - JUnit vs. Scalatest, etc.
  - Coverage
  - Functional testing

Property-based Testing

- There is another type of test specification called **property-based testing**.
  - This can save you a lot of time dreaming up suitable input values: a property-based tester will generate random values as well as corner cases.
  - In Scala, we typically use *Scalacheck* for this.

```
      import org.scalacheck.Properties
import org.scalacheck.Prop.forAll

class RationalPropertySpec extends Properties("String") {

  import Rational.RationalHelper

  property("RationalFromString") = forAll { (a: Int, b: Int) =>
    val rat = r"$a/$b"
    (rat*b).toInt == a
  }
}
```

Integration Testing

- What's the difference between unit testing and integration testing?
  - A unit test is designed to test *exactly* one thing, no more, no less—an integration test is designed to test an entire module, or even an entire application.
  - A unit test runs instantaneously—an integration test may take some significant time to run.
  - A unit test, ideally, does not use mocking—integration tests use mocking a lot.
  - Unit tests are found in the "test" folder under "src" in a maven/sbt project—integration tests are found somewhere else, typically under "it."
  - Unit tests are run (or should be) whenever you change code—integration tests are typically run only when you are ready to commit.

Mocking

Then the Queen left off, quite out of breath, and said to Alice, "Have you seen the Mock Turtle yet?"
"No," said Alice. "*I don't even know what a Mock Turtle is*."
"It's the thing Mock Turtle Soup is made from", said the Queen.
— *Alice in Wonderland*, chapter 9

What is mocking?

- Let's say for example that you have developed a set of DAO classes—
  - in order to test them, you have to connect to a database;
  - this may be an expensive (slow) operation;
  - the database that you connect to may have changed since you last ran the test;
  - the database may not even be built yet—it may be under development as part of your overall project—or promised by some third-party developer.
- For these reasons, you "mock" the database:
  - The most obvious way is simply to define a mock class in your test module: typically, you will be extending a trait or abstract class and defining your own methods;
  - But if you want to do it "properly" (i.e. with the least inconvenience to yourself), use a mocker:
    - *scalamock*, although you can use Java test runners like mockito, etc.

Mock Example

- The following is from Majabigwaduce:

```
  class CountWordsSpec extends FlatSpec with Matchers with Futures with ScalaFutures with Inside
with MockFactory {
    "CountWords" should "work for http://www.bbc.com/ http://www.cnn.com/ http://default/" in {
      val wBBC = "http://www.bbc.com/"
      val wCNN = "http://www.cnn.com/"
      val wDef = "http://default/"
      val uBBC = new URI(wBBC)
      val uCNN = new URI(wCNN)
      val uDef = new URI(wDef)
      val hc = mock[HttpClient]
      val rBBC = mock[Resource]
      (rBBC.getServer _).expects().returning(uBBC)
      rBBC.getContent _ expects() returning CountWordsSpec.bbcText
      val rCNN = mock[Resource]
      rCNN.getServer _ expects() returning uCNN
      rCNN.getContent _ expects() returning CountWordsSpec.cnnText
      val rDef = mock[Resource]
      rDef.getServer _ expects() returning uDef
      rDef.getContent _ expects() returning CountWordsSpec.defaultText
      hc.getResource _ expects wBBC returning rBBC
      hc.getResource _ expects wCNN returning rCNN
      hc.getResource _ expects wDef returning rDef
      val nf = CountWords(hc, Array(wBBC, wCNN, wDef))
      whenReady(nf, timeout(Span(6, Seconds)))(i => assert(i == 556))
    }
}
```

Dependencies, versions, coupling, etc.

- Suppose you have a trait *X[T]* with method *x: T*
  - and a concrete class *X_String* that extends *X[String]* that defines a method *x1: String*.
  - Now, you have another class *Y* where you reference the *X_String* directly and even its method *x1*.
- You've introduced *coupling* into your code. Any new version of *X_String* is liable to require a change to *Y*.
- If you are disciplined and access *only* the method *x* of trait *X*, you will not be likely to need to change your code very much as traits tend to remain stable (assuming they were well-designed in the first place).