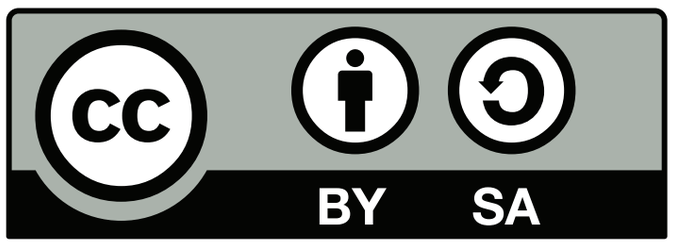
Tecnologia e Applicazioni Internet 2009/10

Lezione 0 - Test-Driven Development

Matteo Vaccari
http://matteo.vaccari.name/
vaccari@pobox.com



Argomenti del corso

- Progettazione applicativa moderna
- Test unitario e funzionale di applicazioni web
- Applicazioni web basate su database
- Java Servlet API
- JavaScript
- Ajax
- Architetture REST

Obiettivo del corso

Diventare programmatori più efficaci

Software come *Lego*

Object-oriented programming

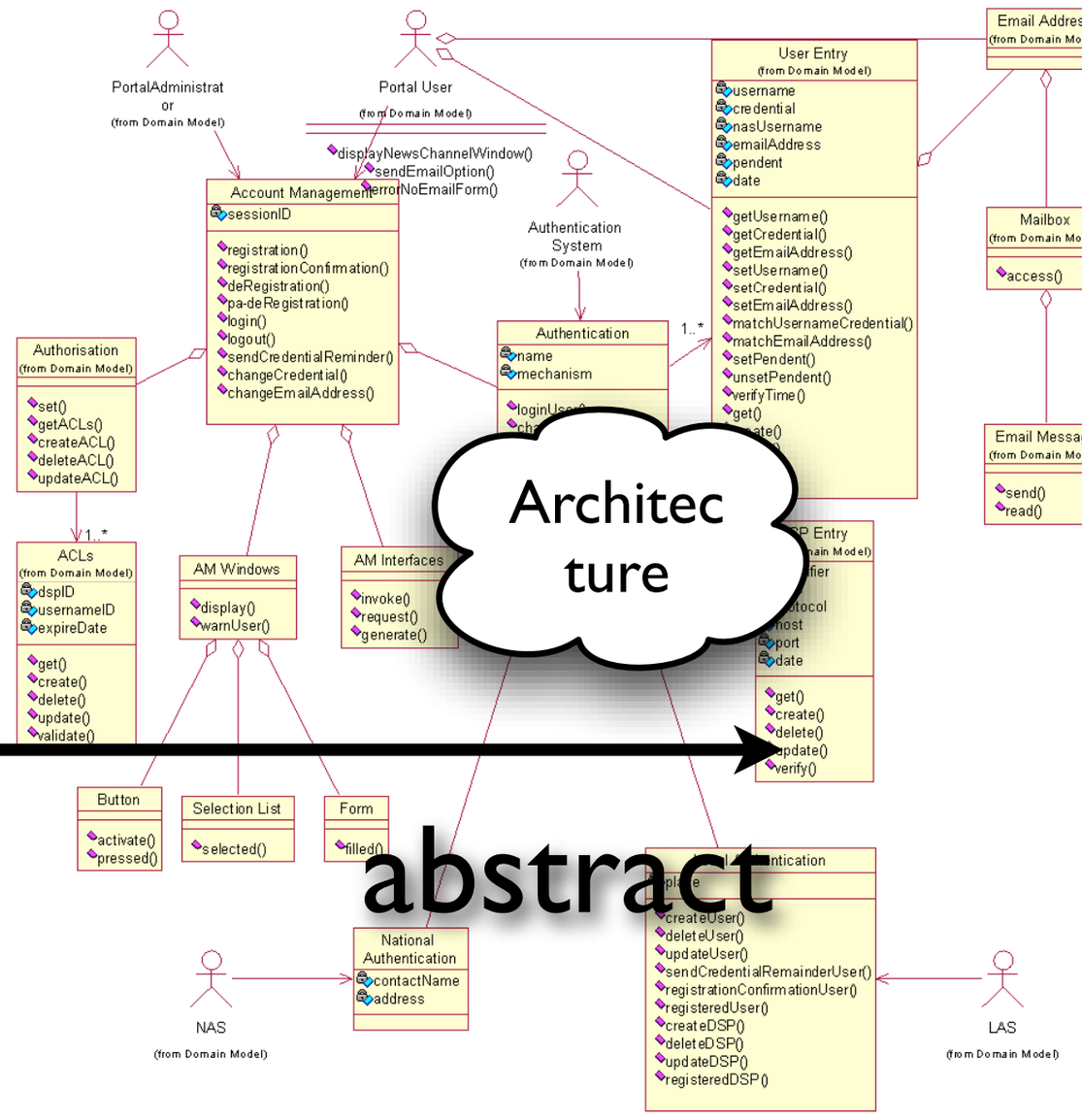
Che cos'è la programmazione a oggetti?

Application Design

What is design?

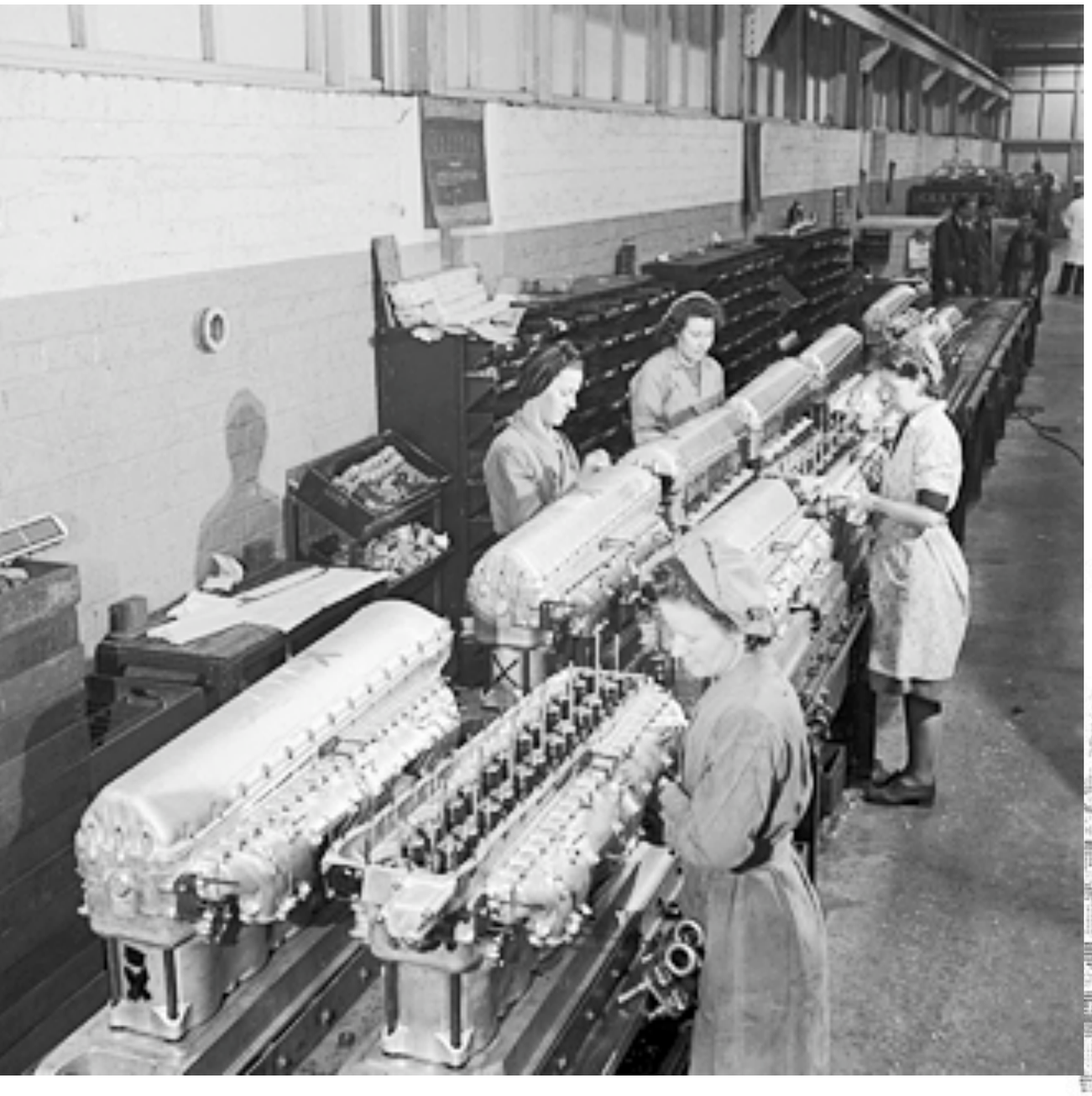
```java
public class XpathTest {

    @Test
    public void testXpath() throws Exception {
        Document doc = documentFromString("<a><b><c>foo</c></b></a>");
        assertEquals(0, getNodeList(doc, "//foo").getLength());
        assertEquals(1, getNodeList(doc, "//c").getLength());
        assertEquals("foo", getNodeContent(doc, "/a/b/c"));
    }

    private String getNodeContent(Document doc, String xpath) throws XPathExpressionException {
        NodeList nodes = getNodeList(doc, xpath);
        assertEquals("expected exactly 1 node at xpath " + xpath, 1, nodes.getLength());
        return nod
    }

    private NodeLi                                    ing xpath) throws XPathExpressionException {
        // see htt                                    ibrary/x-javaxpathapi.html
        XPathFacto                                    tance();
        XPath xp =
        return (No                                    XPathConstants.NODESET);
    }

    private Document documentFromString(String string) throws Exception {
        DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
        factory.setV                                  ing(false);
        factory.setNamespaceAware(true);
        return factory.newDocumentBuilder().parse(new ByteArrayInputStream(string.getBytes()));
    }

    @Test
    public void testTemplate() throws Exception {
        StringTemplate template = new StringTemplate("Hello, $name$!");
        template.setAttribute("name", "world");
        assertEquals("Hello, world!", template.toString());
    }
}
```

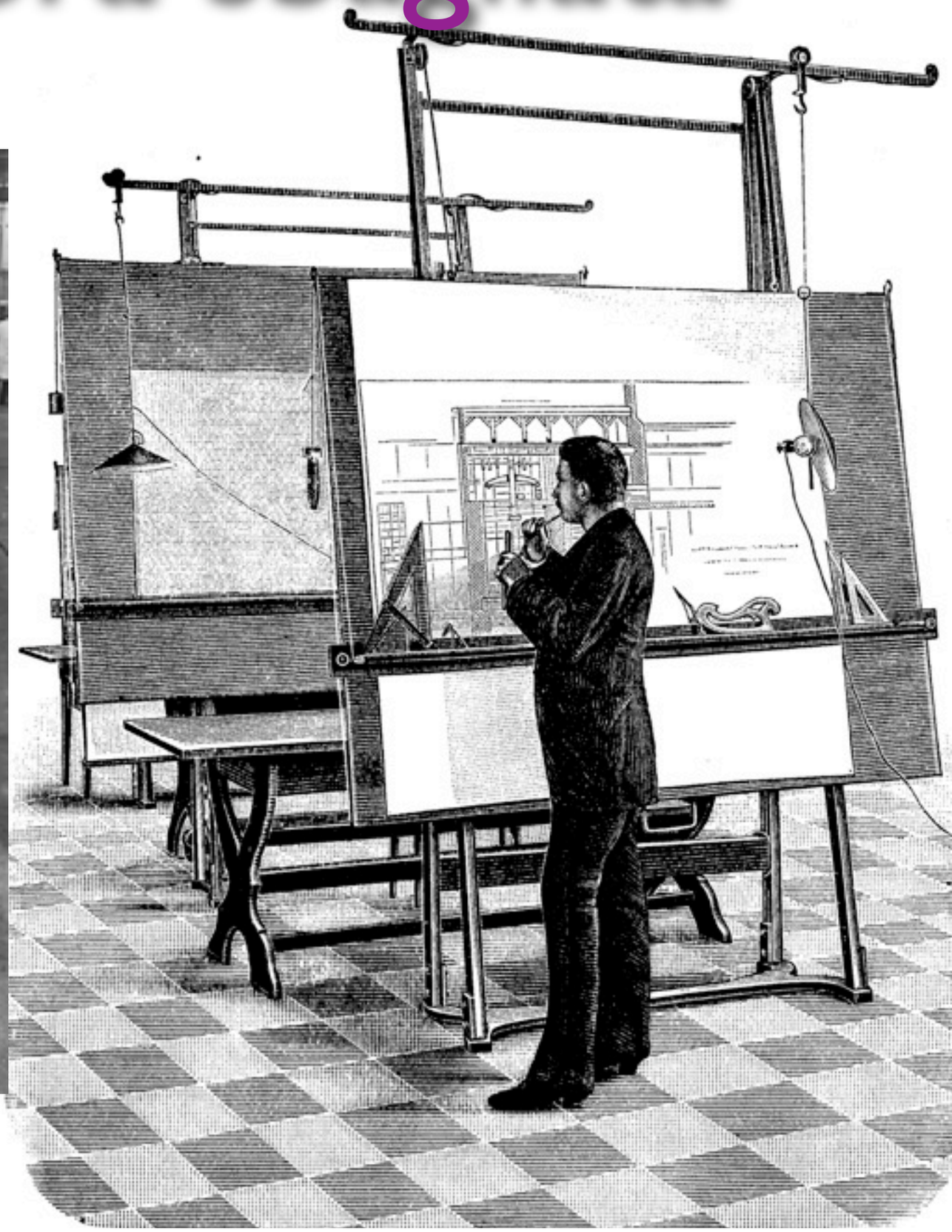
Source code



Il **design** del software è un continuo, che va dai modelli architetturali astratti fatti di diagrammi e descrizioni a parole, via via fino al design più dettagliato, che è rappresentato dal codice sorgente. Il codice sorgente **è** il design più preciso.

Una metafora sbagliata





La produzione di software non assomiglia alla manifattura. In particolare è perniciosa l'idea che i programmatori siano "operai" poco educati che eseguono i progetti prodotti da "architetti" più capaci. Nella produzione di software i programmatori sono **progettisti** a tutti gli effetti. Gli unici "operai" nel mondo del software sono i compilatori, che producono il codice eseguibile dai sorgenti.

SW Development != Manufacturing

```
    PreparedStatementData preparedStatementData = new PreparedStatementData(sql, pa
    executer.executeUpdateStatement(preparedStatementData);
}

protected static void execute(String sql) {
    execute(sql, list());
}

@Override
public boolean equals(Object other) {
    return EqualsBuilder.reflectionEquals(this, other);
}

@Override
public int hashCode() {
  return HashCodeBuilder.reflectionHashCode(this);
}
```

```
@Override
public String toString() {
    return ReflectionToStringBuilder.reflectionToString(this);
}

protected void insert(String tableName, List columnNames, List parameters) {
    String sql = JdbcQueryBuilder.createInsertStatement(tableName, columnNames);
    PreparedStatementData data = new PreparedStatementData(sql , parameters);
    getExecuter().executeInsertStatement(data );
}

public abstract void save();
```

The most accurate *design model* of software is *the code*

Modular programming

- Decompose the system in modules: yes but how?

...it is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.

-- David Parnas, 1972

Perché imparare a fare design?

Le tecniche di programmazione istintiva (cut&paste, code&fix, ecc...) danno all'inizio del progetto una falsa sensazione di velocità. Perché falsa? Perché:

1. La velocità è destinata a diminuire progressivamente. Queste tecniche sono degenerative, raggiungono velocemente degli obiettivi (implementazione di funzionalità) nel breve periodo, ma degradano la qualità della base di codice, il che comporta un rallentamento durante l'implementazione della funzionalità successiva, ecc... fino al raggiungimento del collasso del codice (ovvero il momento in cui i programmatori o scappano, o si impuntano per una riscrittura dell'intero progetto)

...

Gabriele Lana, http://milano-xpug.pbwiki.com/Velocita

Perché imparare a fare design?

...

2. La velocità iniziale non può essere aumentata. Le tecniche di cui sopra non sono fisicamente migliorabili attraverso l'esperienza, la pratica o l'impegno (forse solo leggermente). L'unico modo di aumentare la velocità è di aumentare il numero delle persone coinvolte nella scrittura del progetto; peccato che questo aumenti anche la velocità di degradazione del codice, che porta al peggioramento della situazione in breve tempo.

Gabriele Lana, http://milano-xpug.pbwiki.com/Velocita

Perché imparare a fare design?

The fantasy of cutting quality to deliver faster lives on in part because we do not measure that long period at the end where we are taking out the critical bugs that we put in. We assume, wrongly, that that period is necessary, part of the universe. ...

Low quality has an immediate and visible negative impact on delivering "fast enough". That negative impact is the stabilization period. That is lost time due directly to low quality. ...

To a first approximation, then, looking at a random project, if we want to speed up, we must increase quality.

Ron Jeffries, http://is.gd/kZcp

What is a good design?

- Simple
- No duplication
- Testable
- Expressive
- Easy to change
- Modular

Why testable?

- Test-driven-development is a *design* technique
- Automated tests enable refactoring

Come funziona JUnit?

Il pattern AAA

- *Arrange* creo una collezione di oggetti
- *Act:* eseguo un'operazione sull'oggetto da testare
- *Assert:* asserisco che sia successo quello che mi aspettavo

Test-Driven Development

Write a test

```
public class AdderTest {
   @Test
   public void testTwoPlusThree() {
      Adder a = new Adder();
      assertEquals(5, a.add(2, 3));
   }
}
```

m.vaccari@sourcesense.com

Now it compiles

```
public class AdderTest {
   @Test
   public void testTwoPlusThree() {
      Adder a = new Adder();
      assertEquals(5, a.add(2, 3));
   }
}
```

```
public class Adder {
  public int add(int a, int b) { return 0; }
}
```

m.vaccari@sourcesense.com

Red bar!

```
public class AdderTest {
   @Test
   public void testTwoPlusThree() {
      Adder a = new Adder();
      assertEquals(5, a.add(2, 3));
   }
}
```

```
public class Adder {
  public int add(int a, int b) { return 0; }
}
```

Expected 5, was 0

m.vaccari@sourcesense.com

Do the simplest thing

```
public class AdderTest {
   @Test
   public void testTwoPlusThree() {
      Adder a = new Adder();
      assertEquals(5, a.add(2, 3));
   }
}
```

```
public class Adder {
  public int add(int a, int b) { return 5; }
}
```

m.vaccari@sourcesense.com

Refactor

```
public class AdderTest {
   @Test
   public void testTwoPlusThree() {
      Adder a = new Adder();
      assertEquals(5, a.add(2, 3));
   }
}
```

```
public class Adder {
  public int add(int a, int b) { return a+b; }
}
```

m.vaccari@sourcesense.com

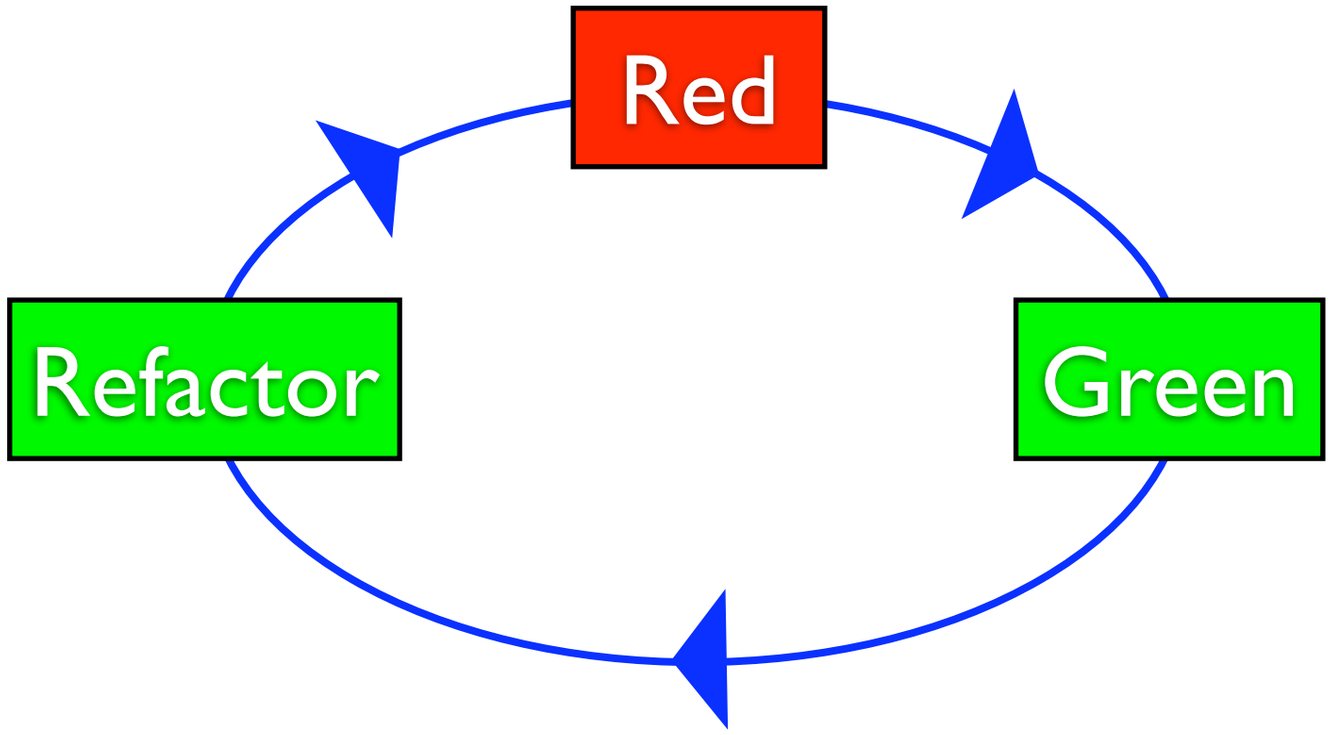
The procedure

1. Write a test
2. Make it compile

3. Make it pass

4. Refactor

m.vaccari@sourcesense.com

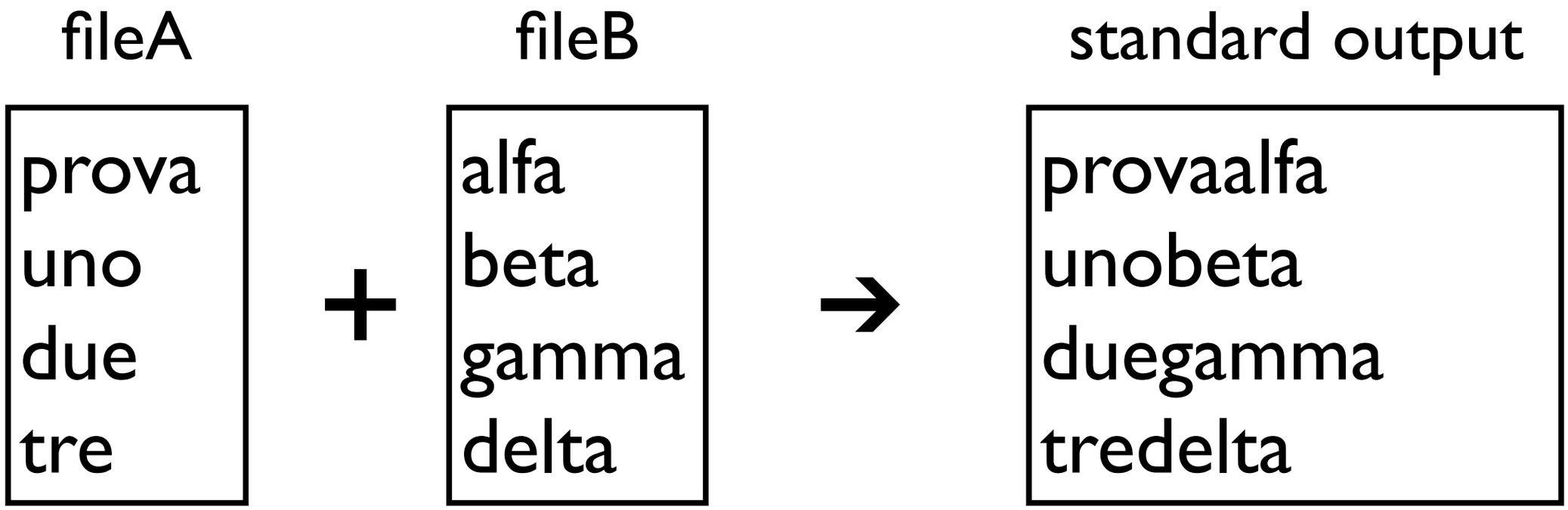


Repeat every 2-10 min.

m.vaccari@sourcesense.com

Esercizio

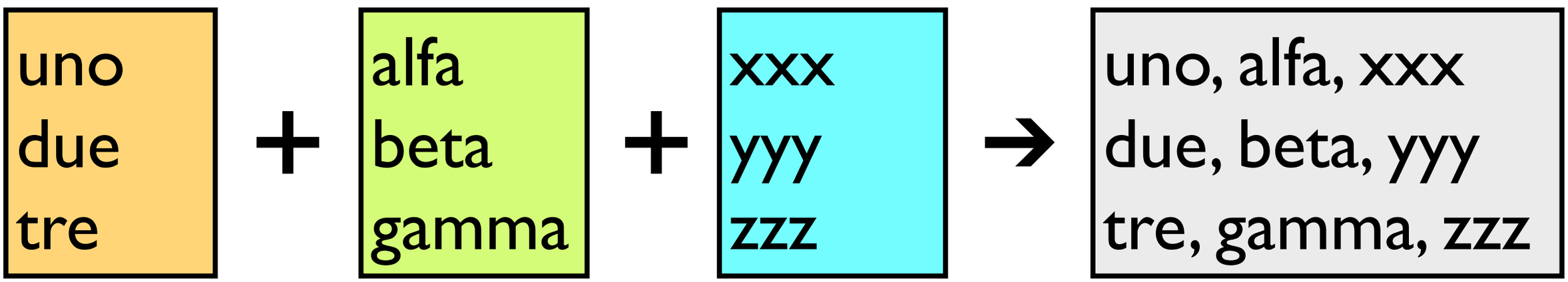
Esercizio: il programma “paste”



paste -- merge corresponding lines of files

$ paste fileA fileB > fileC
$ java -jar paste.jar fileA fileB > fileC

Nuovi requisiti!



- numero arbitrario di file in ingresso
- output separato da virgole

La soluzione “veloce” può avere risolto il problema originariamente. Ma quando dobbiamo, tempo dopo, risolvere un problema simile ma più complesso, adattare il vecchio software può risultare difficile.

Risolvere l'esercizio con Test-Driven Development.
Cerca di mantenere il codice il più
semplice, disaccoppiato e testabile possibile.