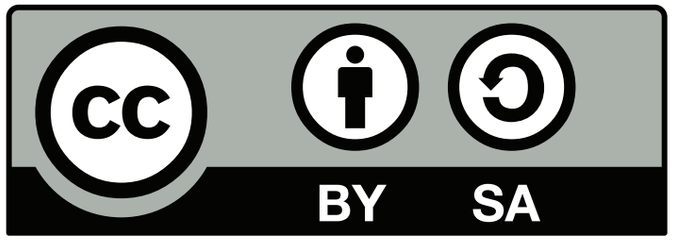
Tecnologia e Applicazioni Internet 2010/11

Lezione 0 - Modern Application Design

Matteo Vaccari
http://matteo.vaccari.name/
matteo.vaccari@uninsubria.it



Le vostre aspettative?

Argomenti

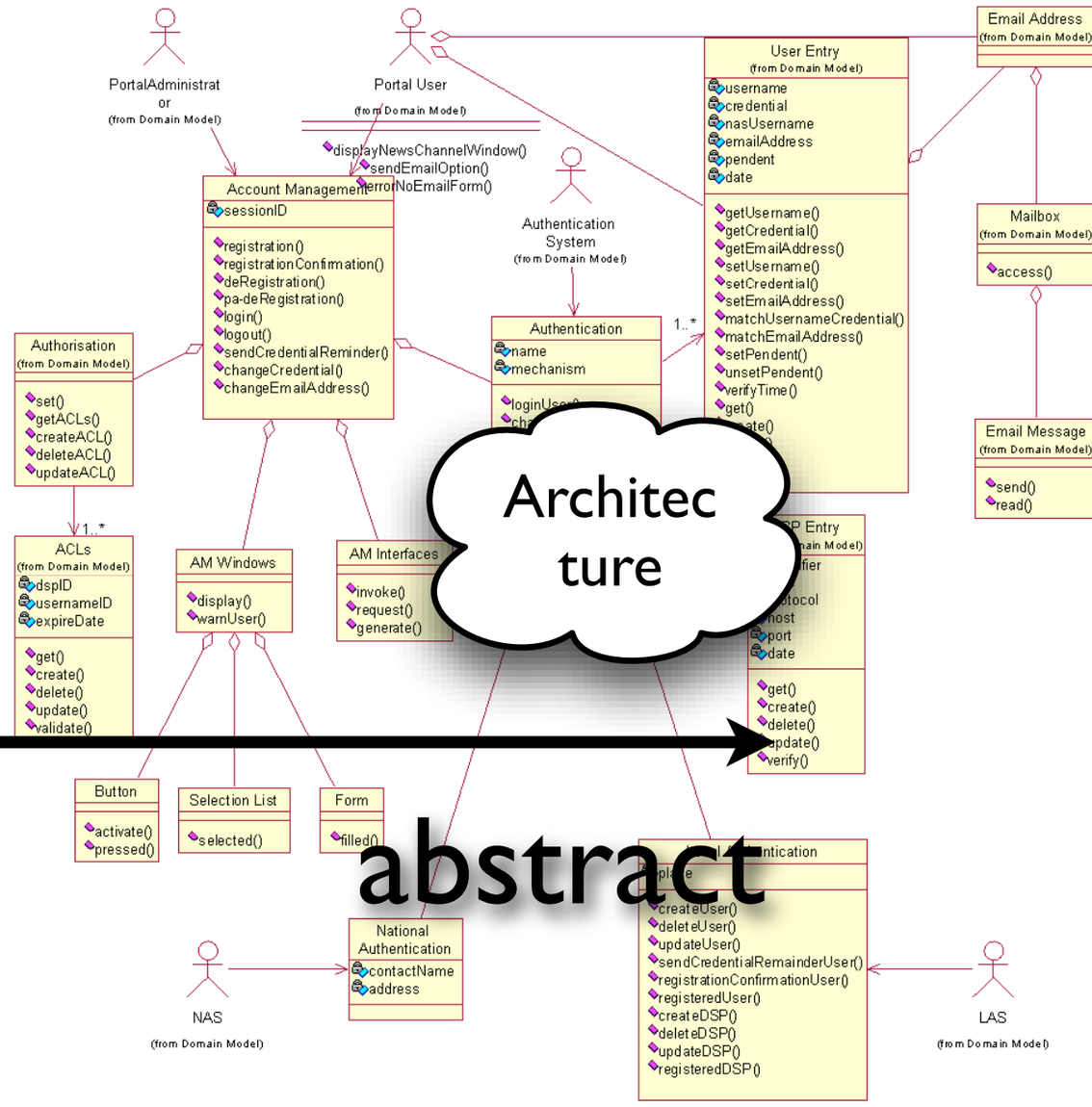
- Progettazione applicativa moderna
- Test unitario e funzionale di applicazioni web
- Uso del database in Java
- Java Servlet API
- JavaScript
- Ajax
- Architetture REST

Application Design

What is design?

```
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
        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

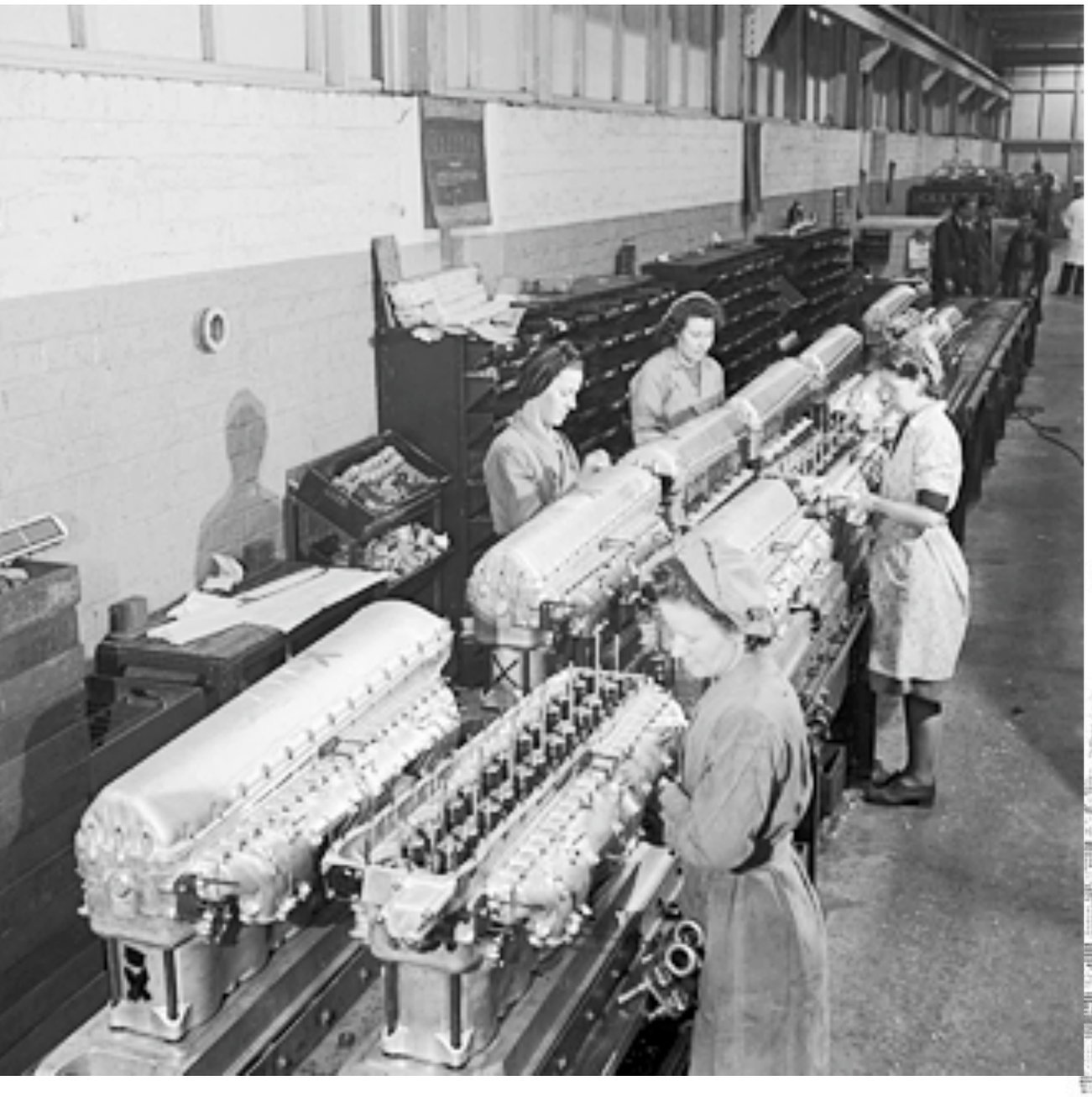


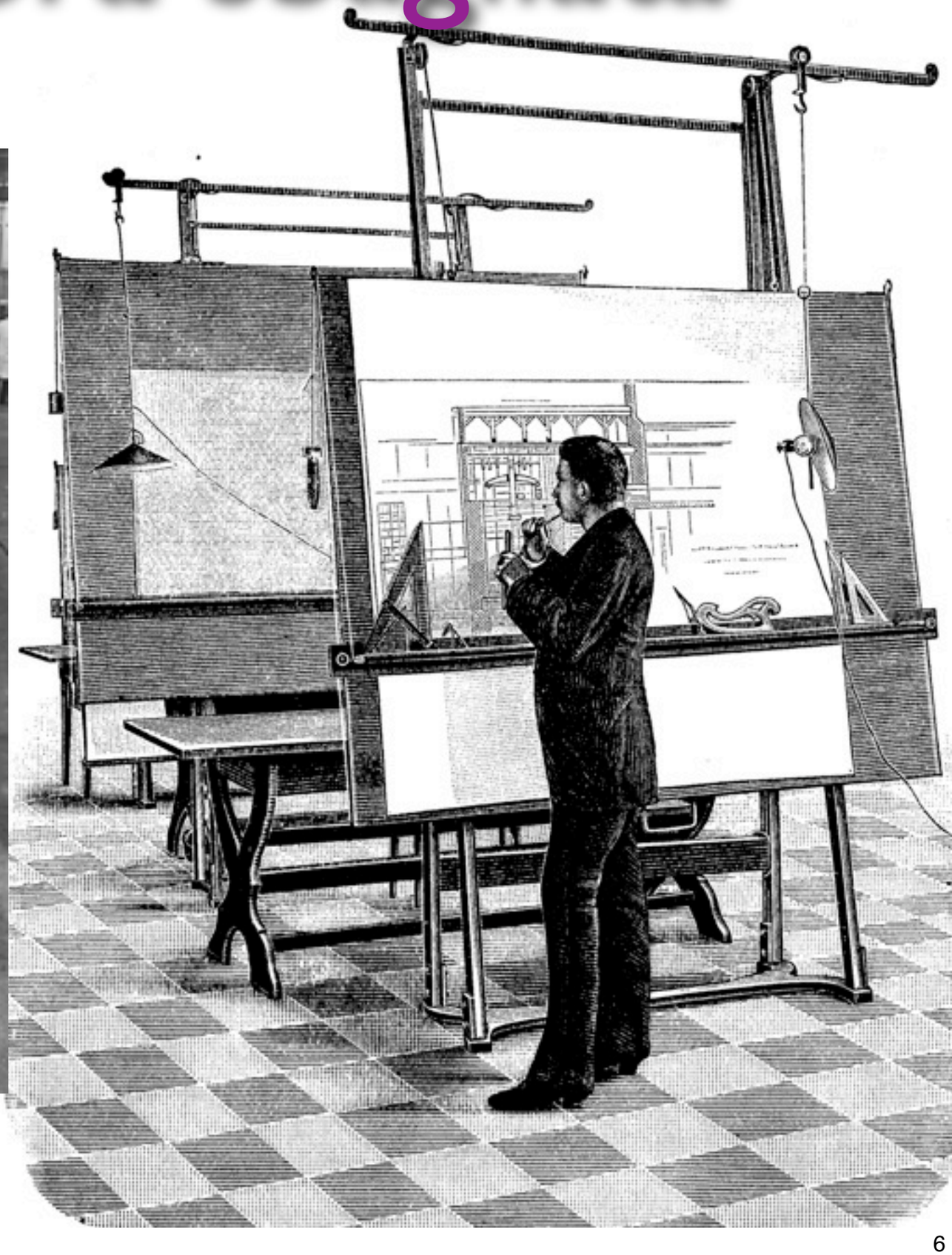
Architecture

detailed

abstract

Una metafora sbagliata





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
- Modules should have:
  - a clear interface
  - low coupling
  - high cohesion
  - a single reason for change

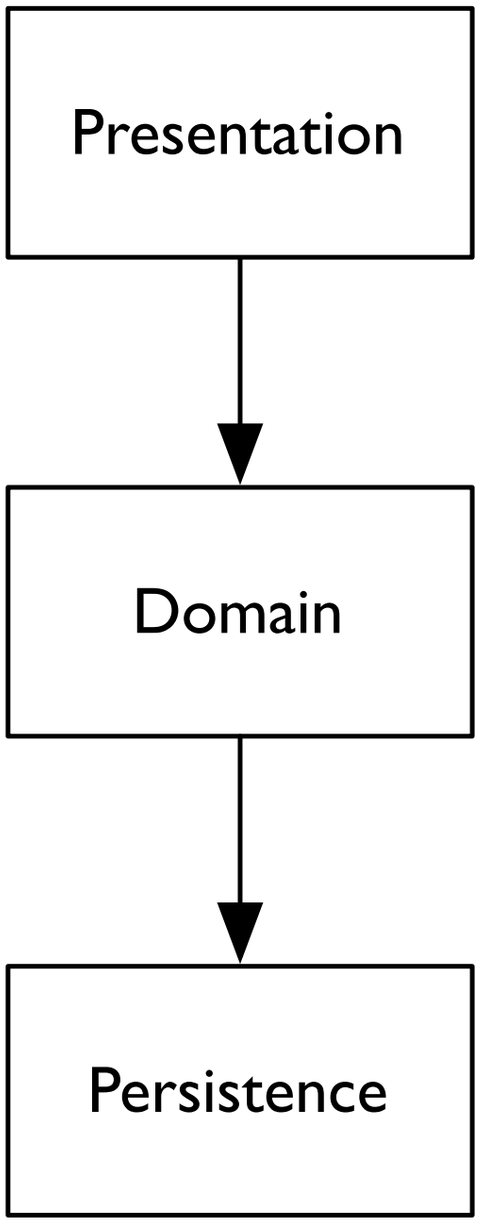
Modular programming

- 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.

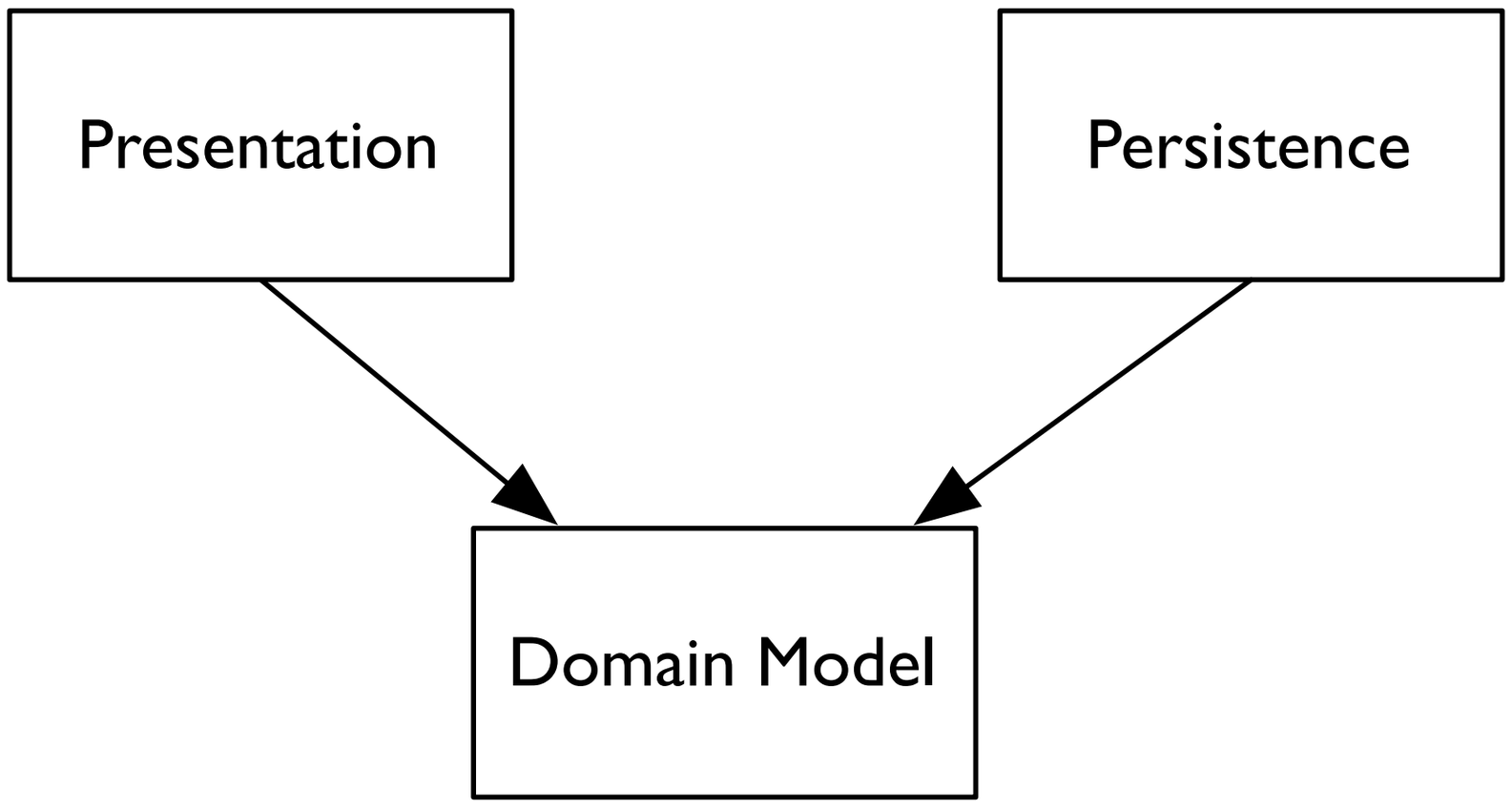
-- David Parnas, 1972

The standard three-layers architecture



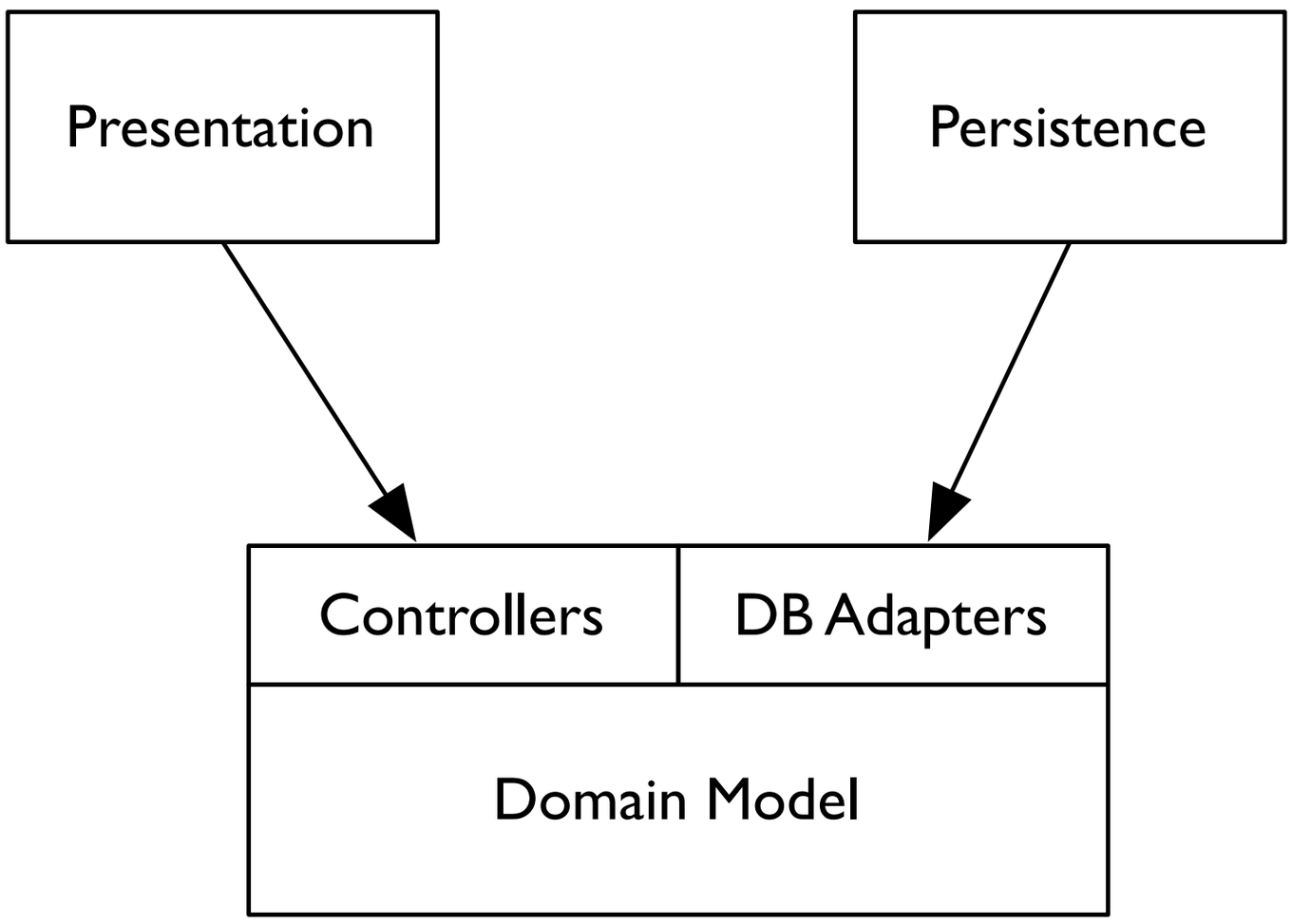
- Boxes are *layers* (sets of classes)
- Arrows mean "depends on"

A better variation



- Everything depends on the domain model
- The domain model depends on nothing

In greater detail



- Controllers are called by the presentation layers
- DB Adapters are called by the domain layer

Che cos'è il domain model?

Ma lo prence de li laièri, la più magna e nobile parte de lo projetto tutto, serà lo laièro mediano. Codesto, nomato la "logica de l'affari", contarrà tutte le parti complesse de lo sistema et sophisticate ancora. Et invero, hesso laièro spartirà lo programmatore probo et astuto de lo cacchione emprovvisato et stolto.

"Leonardo da Vinci" (in realtà Paolo Perrotta all'Italian Agile Day 2009 :-)

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 good design?

The code is simple enough when it:

1. Runs all the tests
2. Contains no duplication
3. Expresses every idea that we need to express
4. Has the minimum number of classes and functions

(In this order)

Adapted from *Extreme Programming Installed* by Ron Jeffries et al.

Come funziona JUnit?

Example test

```
import org.junit.Test;

import static org.junit.Assert.*;

public class SampleTest {

    @Test
    public void firstTest() throws Exception {
        assertEquals(42, someComputation());
    }
}
```

org.junit.Assert.

expected

actual

Many kinds of assertions

```
assertEquals(6, 3+3);
assertNull(someThing());
assertNotNull(someThingElse());
assertTrue(a() <= b());
assertFalse(foobar());

assertEquals("3+3!", 6, 3+3);
assertNull("was expecting null in someThing()", someThing());
assertNotNull("unexpected null someThingElse()", someThingElse());
assertTrue("a() should be <= b()", a() <= b());
assertFalse("we expected false in foobar", foobar());
```

Before and After

```
public class BlogTest {

    private Blog blog = new Blog();
    HttpUser user = new HttpUser();

    @Before
    public void setUp() throws Exception {
        blog.start(8080);
    }

    @After
    public void tearDown() throws Exception {
        blog.shutdown();
    }

    @Test
    public void answersToHttp() throws Exception {
        HttpResponse response = user.get("http://localhost:8080/");
        assertEquals(HttpResponse.OK, response.getStatus());
    }
```

Test-Driven Development

The TDD rhythm

1. Quickly add a test
2. Run all the tests and see the new one fail
3. Make a little change
4. Run all the tests and see them all succeed
5. Refactor to remove duplication

Esercizio...
Implementare le regole di valutazione del punteggio di Yahtzee

Yahtzee®

NAME anessa

UPPER SECTION	HOW TO SCORE	GAME #1	GAME #2	GAME #3	GAME #4	GAME #5	GAME #6
Aces • = 1	Count and Add Only Aces	1	4	4	1	2	2
Twos = 2	Count and Add Only Twos	4	2	6	6	4	6
Threes = 3	Count and Add Only Threes	9	3	6	9	12	9
Fours = 4	Count and Add Only Fours	16	8	8	8	16	8
Fives = 5	Count and Add Only Fives	10	20	20	—	10	15
Sixes = 6	Count and Add Only Sixes	18	18	24	18	24	18
TOTAL SCORE	→	58	55	68	42	68	57
BONUS If total score is 63 or over	SCORE 35	—	—	35	-	35	—
TOTAL Of Upper Section	→	58	55	103	42	103	57
LOWER SECTION							
3 of a kind	Add Total Of All Dice	25	29	25	28	26	28
4 of a kind	Add Total Of All Dice	14	29	27	28	X	25
Full House	SCORE 25	25	25	25	25	25	25
Sm. Straight Sequence of 4	SCORE 30	30	30	30	30	30	30
Lg. Straight Sequence of 5	SCORE 40	40	40	40	40	X	40
YAHTZEE 5 of a kind	SCORE 50	0	50	0	50	50	—
Chance	Score Total Of All 5 Dice	19	9	18	23	22	8
YAHTZEE BONUS	FOR EACH BONUS	—					
	SCORE 100 PER	—					
TOTAL Of Lower Section	→	155	212	165	224	153	156
TOTAL Of Upper Section	→	58	55	103	42	103	57
GRAND TOTAL	→	213	267	268	266	266	213

```java
public class YahtzeeScoreTest {

   @Test
   public void no_aces_with_aces_rule_gives_0() throws Exception {
      Throw dice = new Throw(new Integer[] {2, 2, 2, 2, 2});
      assertEquals(0, dice.score("Aces"));
   }

   @Test
   public void one_ace_with_aces_rule_gives_1() throws Exception {
      Throw dice = new Throw(new Integer[] {2, 1, 2, 2, 2});
      assertEquals(1, dice.score("Aces"));
   }

   @Test
   public void four_aces_with_aces_rule_gives_4() throws Exception {
      Throw dice = new Throw(new Integer[] {1, 1, 2, 1, 1});
      assertEquals(4, dice.score("Aces"));
   }

   @Test
   public void one_two_with_rule_of_twos_gives_two() throws Exception {
      Throw dice = new Throw(new Integer[] {1, 1, 2, 1, 1});
      assertEquals(2, dice.score("Twos"));
   }

   // ... etc ... implement all 6 basic rules from Aces to Sixes
}
```

```java
// Then implement the two significant cases for the Yahtzee rule

@Test
public void five_ones_with_yahtzee_rule_gives_50() throws Exception {
    Throw dice = new Throw(new Integer[] {1, 1, 1, 1, 1});
    assertEquals(50, dice.score("Yahtzee"));
}
```

```java
@Test
public void yahtzee_rule_gives_0_when_no_five_equals() throws Exception {
    Throw dice = new Throw(new Integer[] {1, 1, 1, 1, 2});
    assertEquals(0, dice.score("Yahtzee"));
}
```

```
// Then implement the 3-of-a-kind, 4-of-a-kind rules and the full-house rule
// Finally implement the Small and Large Straight

@Test
public void large_straight_gives_40() throws Exception {
   assertScoreIs(40, "Large Straight", new Integer[] {1, 2, 3, 4, 5});
   assertScoreIs(40, "Large Straight", new Integer[] {2, 1, 3, 5, 4});
}
```

```
@Test
public void large_straight_gives_0_if_you_dont_have_a_straight() throws Exception {
   assertScoreIs(0, "Large Straight", new Integer[] {2, 1, 3, 5, 4});
}

private void assertScoreIs(int expected, String ruleName, Integer[] rolls) {
   // define an abbreviation to remove duplication
}
```