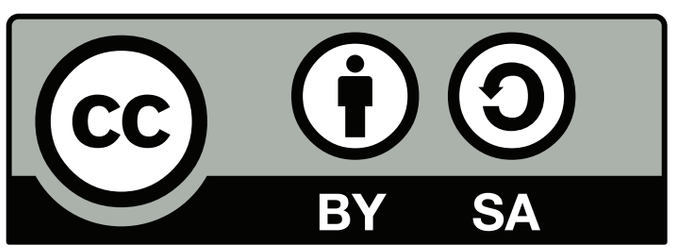
Tecnologia e Applicazioni Internet 2011/12

Lezione 0 - Test-Driven Development

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



Le vostre aspettative?

Argomenti

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

Esame

- 7 punti: laboratorio
- 7 punti: orale
- 16 punti: elaborato

Application Design

Obiettivo: software *manutenibile* e *affidabile*

Affidabile = ?

Assenza di difetti :-)

Working software

Come ottenere software affidabile?

- Testing
- Review
- Design

Terminologia

Definition: **Testing**

Testing is the process of executing software in order to find failures.

Definition: **Failure**

A failure is the situation in which the behavior of the executing software deviates from what is expected.

Definition: **Defect**

A defect is the algorithmic cause of a failure: some code logic that is incorrectly implemented.

Terminologia

Definition: Test case

A test case is a definition of input values and expected output values for the unit under test.

Definition: Unit under test

The unit under test is some part of the system that we consider to be a whole.

```
public class Date {
  public enum Weekday {
      MONDAY, TUESDAY, WEDNESDAY,
      THURSDAY, FRIDAY,
      SATURDAY, SUNDAY };

  /**
   * Construct a date object.
   * @param year the year as integer, i.e. year 2010 is 2010.
   * @param month the month as integer, i.e. januar is 1, december is 12.
   * @param dayOfMonth the day number in the month, range 1..31.
   * PRECONDITION: The date parameters must represent a valid date.
  */
  public Date(int year, int month, int dayOfMonth) {}

  /**
  * Calculate the weekday that this Date object represents.
  * @return the weekday of this date.
  */
  public Weekday dayOfWeek() {
    // Fake implementation, only for demonstrating testing.
    return Weekday.SATURDAY;
  }
}
```

Terminologia

Definition: Manual Testing

Manual testing is a process in which suites of test cases are executed and verified manually by humans.

Definition: Automated Testing

Automated testing is a process in which test suites are executed and verified automatically by computer programs.

Terminologia

Definition: **Production Code**

The production code is the code that defines the behavior implementing the software's requirements.

Definition: **Test Code**

The test code is the source code that defines test cases for the production code.

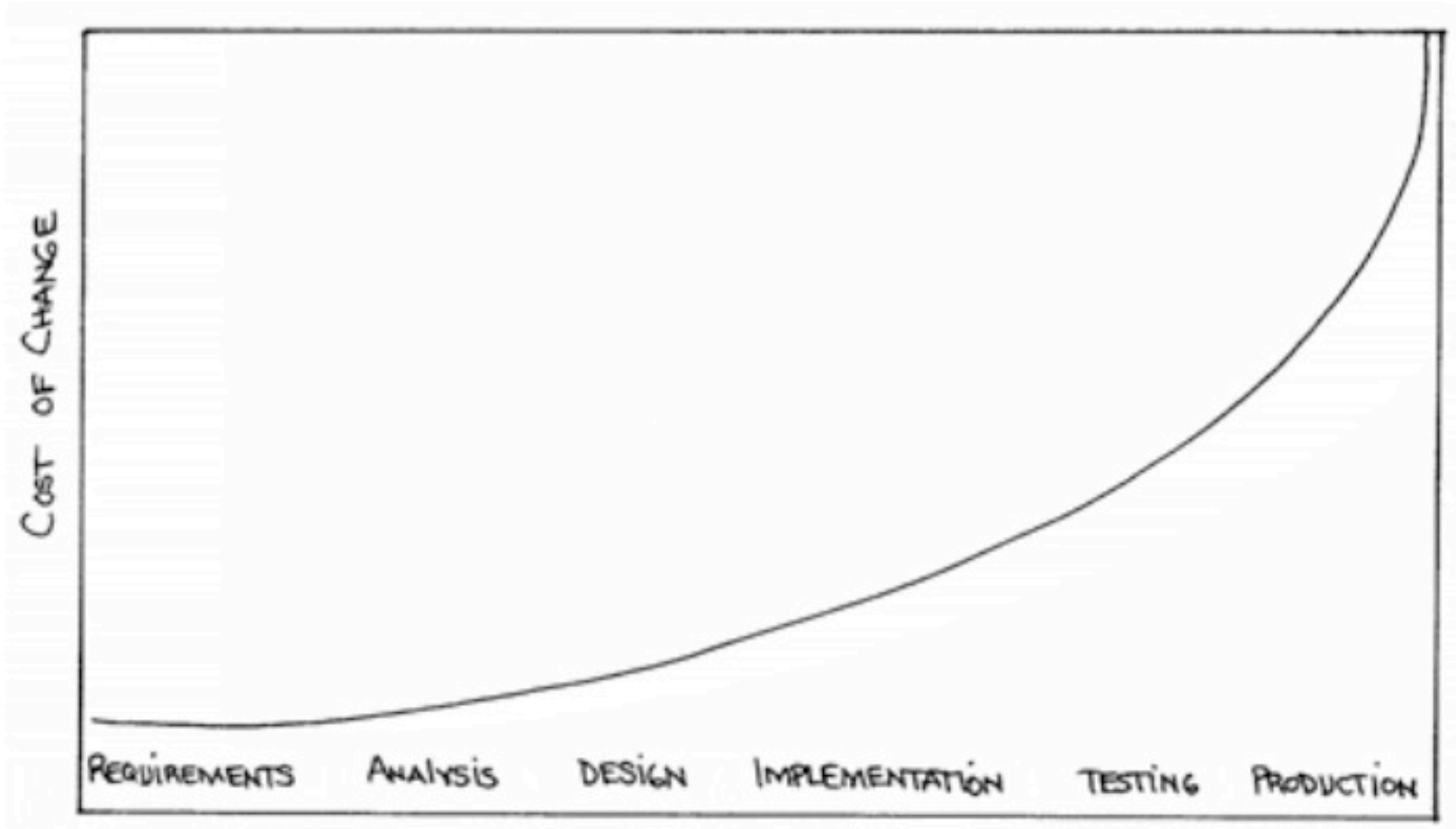
Tabella di casi di test

Unit under test: dayOfWeek	
Input	Expected output
year=2008, month=May, dayOfMonth=19	Monday
year=2008, month=Dec, dayOfMonth=25	Thursday
year=2010, month=Dec, dayOfMonth=25	Saturday

- Unit under test
- Input values
- Expected output

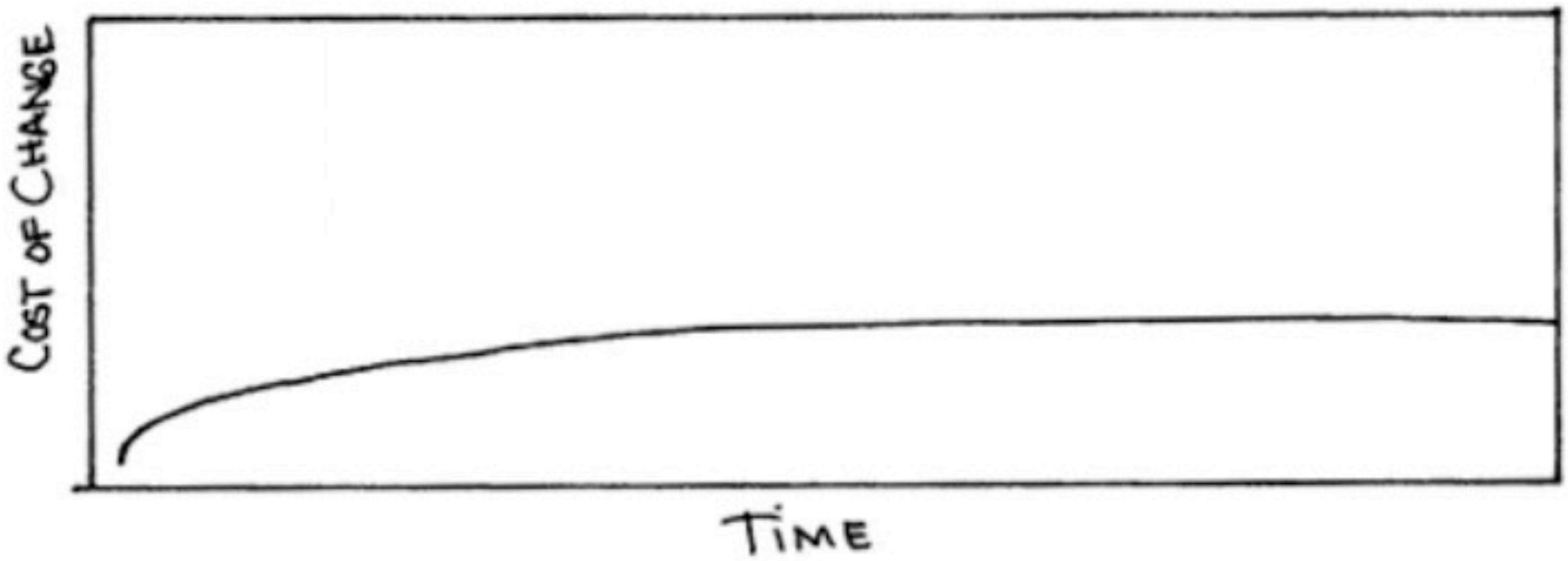
Manutenibile = ?

Costo del cambiamento



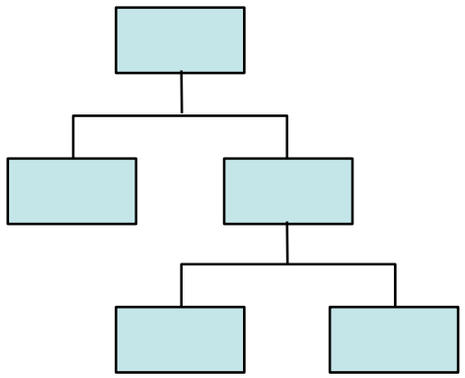
Kent Beck, *Extreme Programming Explained*

Può essere così?

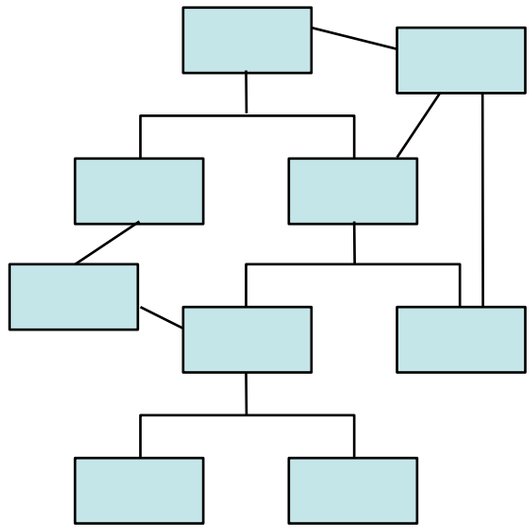


Kent Beck, *Extreme Programming Explained*

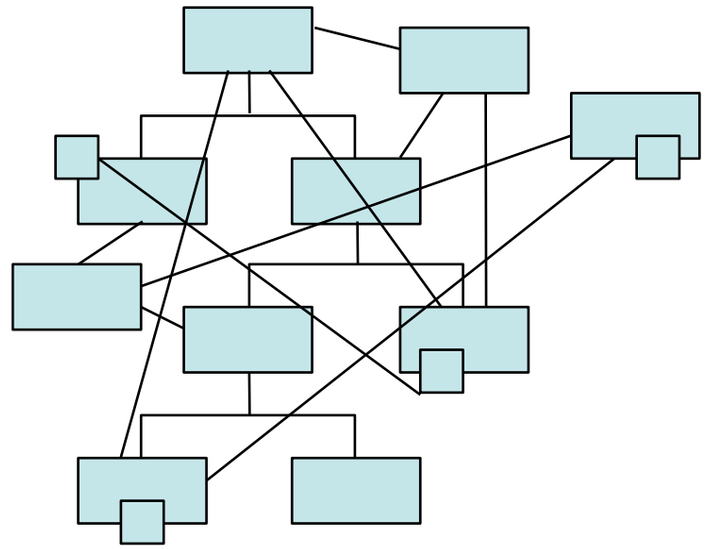
Perché il costo del cambiamento aumenta?



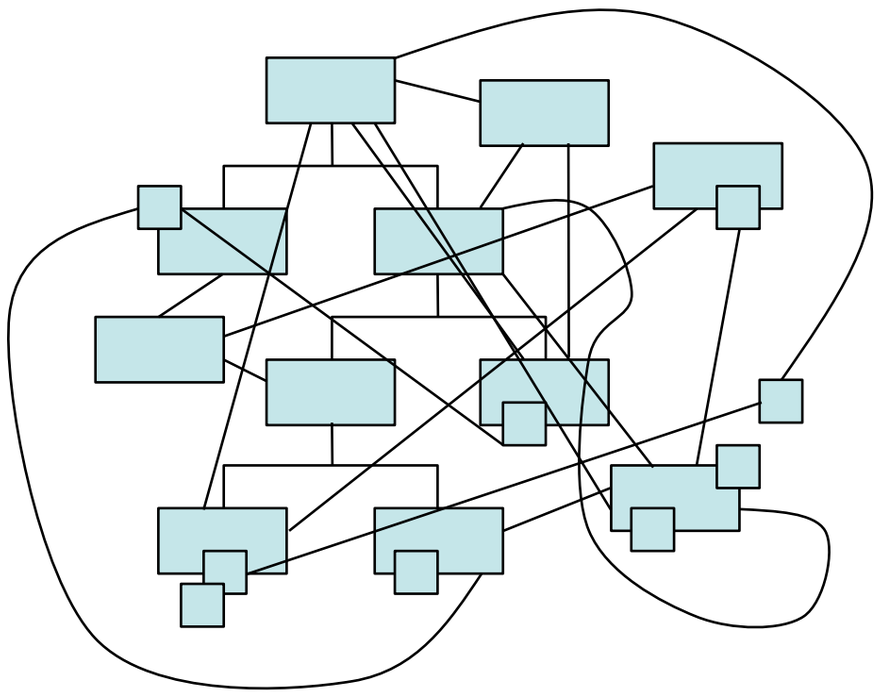
Cost of change: C



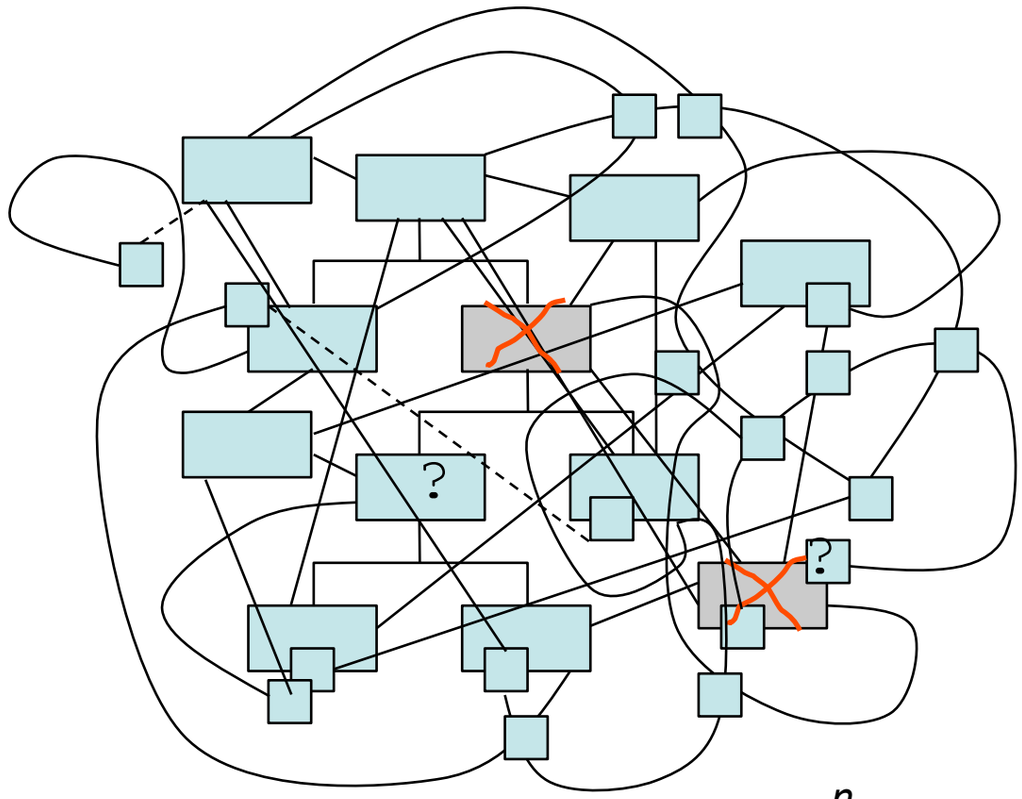
Cost of change: C + *n*



Cost of change: C x *n*

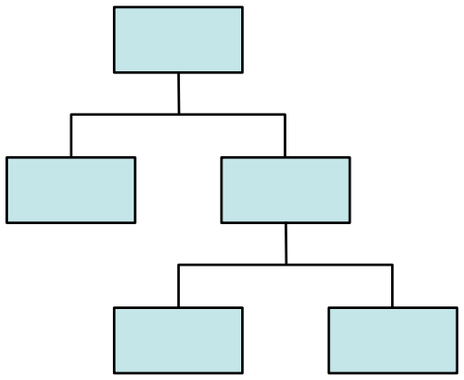


Cost of change: C^{n}



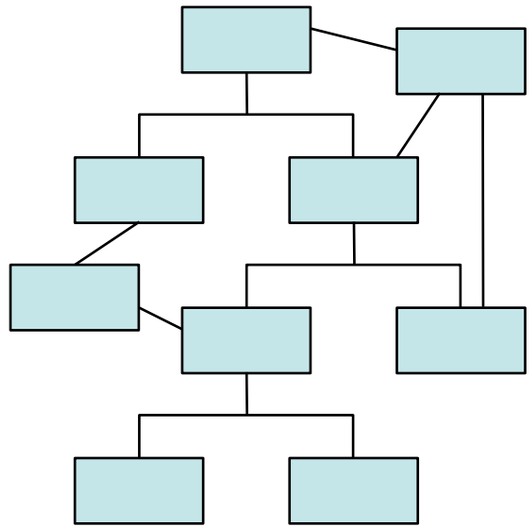
Cost of change: $C^{n^{n}}$

```
public IRequestTarget resolve(final RequestCycle requestCycle,
        final RequestParameters requestParameters)
{
        IRequestCodingStrategy requestCodingStrategy = requestCycle.getProcessor()
                .getRequestCodingStrategy();

        final String path = requestParameters.getPath();
        IRequestTarget target = null;

        // See whether this request points to a bookmarkable page
        if (requestParameters.getBookmarkablePageClass() != null)
        {
                target = resolveBookmarkablePage(requestCycle, requestParameters);
        }
        // See whether this request points to a rendered page
        else if (requestParameters.getComponentPath() != null)
        {
                // marks whether or not we will be processing this request
                int processRequest = 0; // 0 == process, 1 == page expired, 2 == not active page anymore
                synchronized (requestCycle.getSession())
                {
                        // we need to check if this request has been flagged as
                        // process-only-if-path-is-active and if so make sure this
                        // condition is met
                        if (requestParameters.isOnlyProcessIfPathActive())
                        {
                                // this request has indeed been flagged as
                                // process-only-if-path-is-active

                                Session session = Session.get();
                                IPageMap pageMap = session.pageMapForName(requestParameters.getPageMapName(), false);
                                if (pageMap == null)
                                {
                                        // requested pagemap no longer exists - ignore this
                                        // request
                                        processRequest = 1;
                                }
                                else if (pageMap instanceof AccessStackPageMap)
                                {
                                        AccessStackPageMap accessStackPageMap = (AccessStackPageMap)pageMap;
                                        if (accessStackPageMap.getAccessStack().size() > 0)
                                        {
```



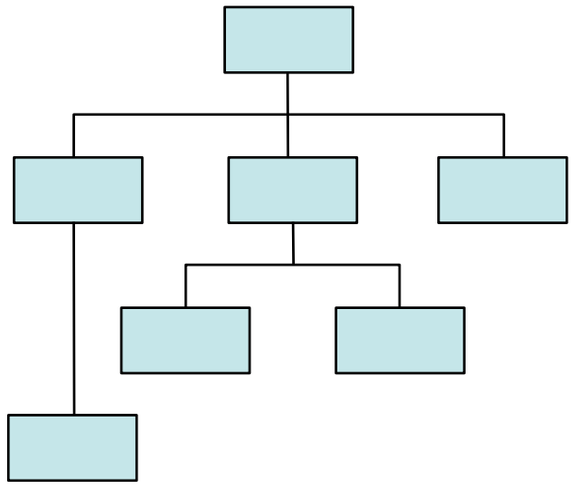
Starting code base

Cost of change: C



Changes implemented

Cost of change: C + n



Code cleaned up

Cost of change: C

Test-Driven Development

Obiettivo:

clean code that works

I Valori del TDD

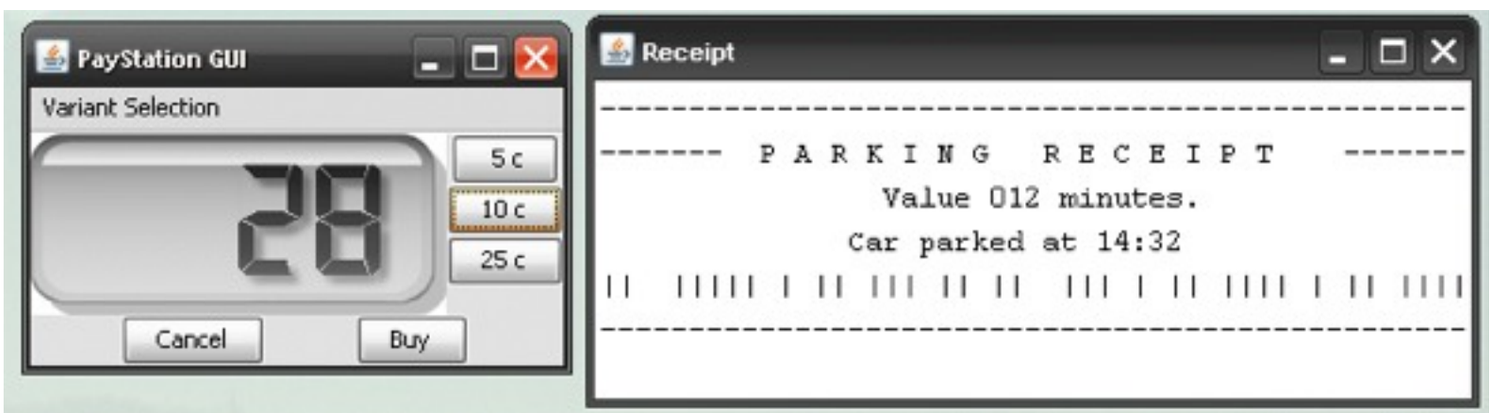
- Mantieni il *focus*
  - *Fai una sola cosa per volta*
- Passi piccoli!
- Semplicità

You are all employed today

AARHUS UNIVERSITET

Welcome to *PayStation Ltd.*

We will develop the main software to run pay stations.



Case: Pay Station

Welcome to *PayStation Ltd.*

Customer: AlphaTown

Requirements

- accept coins for payment
  - 5, 10, 25 cents
- show time bought on display
- print parking time receipts
- US: 2 minutes cost 5 cent
- handle buy and cancel

Stories

AARHUS UNIVERSITET

Story 1: Buy a parking ticket. A car driver walks to the pay station to buy parking time. He enters several valid coins (5, 10, and 25 cents) as payment. For each payment of 5 cents he receives 2 minutes parking time. On the pay station's display he can see how much parking time he has bought so far. Once he is satisfied with the amount of time, he presses the button marked "Buy". He receives a printed receipt, stating the number of minutes parking time he has bought. The display is cleared to prepare for another transaction.

Story 2: Cancel a transaction. A driver has entered several coins but realize that the accumulated parking time shown in the display exceeds what she needs. She presses the button marked "Cancel" and her coins are returned. The display is cleared to prepare for another transaction.

Story 3: Reject illegal coin. A driver has entered 50 cents total and the display reads "20". By mistake, he enters a 1 euro coin which is not a recognized coin. The pay station rejects the coin and the display is not updated.

Minimum Terminology

- Production code
- Test code
- Failure
- Defect
- Test case:
  - Input: -37;
  - Execute: Math.abs(x);
  - expected output: +37

Principi del TDD

- Principle: *test*
- How do I test my code? Write an automated test

Principi del TDD

- Principle: *test first*
  - When should you write your tests? Before you write the code that's to be tested

Principi del TDD

- Principle: *test list*
  - What should you test?
    - Before you begin, write a list of all the tests you know you will have to write.

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

The Rhythm: Red-Green-Refactor

AARHUS UNIVERSITET

The Rhythm

Improve code quality

Clean part

Implement delta-feature that does not break any existing code

Works part

Introduce test of delta-feature

Quanto dura un ciclo?

- Da 1 a 15 minuti -- di più significa che non stiamo facendolo bene

Exercise: Test List

Generate the Test List for these stories

Story 1: Buy a parking ticket. A car driver walks to the pay station to buy parking time. He enters several valid coins (5, 10, and 25 cents) as payment. For each payment of 5 cents he receives 2 minutes parking time. On the pay station's display he can see how much parking time he has bought so far. Once he is satisfied with the amount of time, he presses the button marked "Buy". He receives a printed receipt, stating the number of minutes parking time he has bought. The display is cleared to prepare for another transaction.

Story 2: Cancel a transaction. A driver has entered several coins but realize that the accumulated parking time shown in the display exceeds what she needs. She presses the button marked "Cancel" and her coins are returned. The display is cleared to prepare for another transaction.

Story 3: Reject illegal coin. A driver has entered 50 cents total and the display reads "20". By mistake, he enters a 1 euro coin which is not a recognized coin. The pay station rejects the coin and the display is not updated.

?

Una lista di test

- Accetta una moneta corretta
- 5 cent comprano 2 minuti
- Rifiuta moneta scorretta
- Il display mostra i minuti comprati
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione

Preliminari

- Apriamo Eclipse, prendiamo confidenza con JUnit

Da dove iniziamo?

- Principio TDD: inizia da un test che:
- ti insegna qualcosa e che
- pensi di poter implementare

Una lista di test

- Accetta una moneta corretta
- 5 cent comprano 2 minuti
- Rifiuta moneta scorretta
- Il display mostra i minuti
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione

Scegliamo...

- Accetta una moneta corretta
- 5 cent comprano 2 minuti
- Rifiuta moneta scorretta
- Il display mostra i minuti
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione

TDD Principle: **Fake It ('Til You Make It)**

What is your first implementation once you have a broken test? Return a constant.
Once you have your tests running, gradually transform it.

Una lista di test

- ~~Accetta una moneta corretta~~
- ~~5 cent comprano 2 minuti~~
- Rifiuta moneta scorretta
- ~~Il display mostra i minuti~~
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione

TDD Principle: **Triangulation**

How do you most conservatively drive abstraction with tests? Abstract only when you have two or more examples.

Triangoliamo

- ~~Accetta una moneta corretta~~
- ~~5 cent comprano 2 minuti~~
- Rifiuta moneta scorretta
- ~~Il display mostra i minuti~~
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione
- *25 cent comprano 10 minuti*

Scegliamo il secondo test

- ~~Accetta una moneta corretta~~
- ~~5 cent comprano 2 minuti~~
- Rifiuta moneta scorretta
- ~~Il display mostra i minuti~~
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione
- *25 cent comprano 10 minuti*

Il secondo test passa!

- ~~Accetta una moneta corretta~~
- ~~5 cent comprano 2 minuti~~
- Rifiuta moneta scorretta
- ~~Il display mostra i minuti~~
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione
- ~~25 cent comprano 10 minuti~~

Questa implementazione è sbagliata!!!

```
public class PayStationImpl implements PayStation {
  private int insertedSoFar;
  public void addPayment( int coinValue )
    throws IllegalCoinException {
    insertedSoFar = coinValue;
  }
  public int readDisplay() {
    return insertedSoFar / 5 * 2;
  }
  public Receipt buy() {
    return null;
  }
  public void cancel() {
  }
}
```

Che cosa dobbiamo fare ora?

Aggiungiamo un test!

- ~~Accetta una moneta corretta~~
- ~~5 cent comprano 2 minuti~~
- Rifiuta moneta scorretta
- ~~Il display mostra i minuti~~
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione
- ~~25 cent comprano 10 minuti~~
- *2 monete da 5 cent comprano 4 minuti*

Scegliamo il terzo test

- ~~Accetta una moneta corretta~~
- ~~5 cent comprano 2 minuti~~
- Rifiuta moneta scorretta
- ~~Il display mostra i minuti~~
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione
- ~~25 cent comprano 10 minuti~~
- 2 monete da 5 cent comprano 4 minuti

Scegliamo il quarto test

- ~~Accetta una moneta corretta~~
- ~~5 cent comprano 2 minuti~~
- ~~Rifiuta moneta scorretta~~
- ~~Il display mostra i minuti~~
- Premendo “buy” ottengo una ricevuta
- Premendo “cancel” si resetta la stazione
- ~~25 cent comprano 10 minuti~~
- 2 monete da 5 cent comprano 4 minuti

Step 3. Make a little change

- *Ups? Little change??? We need two changes*
  - An implementation of Receipt
  - Implementing the buy method
- *Small steps? What are my options?*
  - The old way: Do both in one go!
  - Fix receipt first, buy next...
  - Fix buy first, implement receipt later...

Che cosa facciamo?

A) Do both in one go!

B) Fix receipt first, buy next...

C) Fix buy first, implement receipt later...

Votiamo :-)

Analisi

- Small steps: B or C
  - Fix receipt first, buy next...
    - Perdo il focus!!!
- Complete buy first – do receipt next
  - Ma come faccio senza avere una Receipt?

Suspence ...

Fake it!

Restituiamo un oggetto “finto”

Quinto test: *Receipt*

Sesto test: *Buy* (vero)

Problema: 100 cent comprano 40 minuti

- But how to enter 100 cent?
  - add 5, add 5, add 5, add 10, add ...
  - for (int i = 0; i <= 20; i++) { add 5; }
  - private method add2Quarters()