

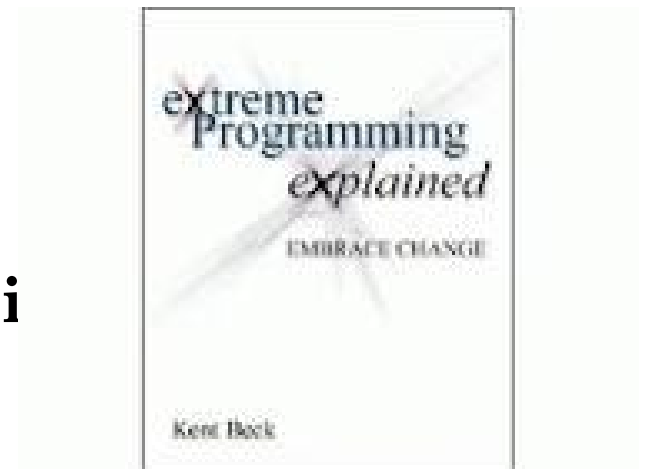


# **METODI AGILI: EXTREME PROGRAMMING**

**Ingegneria del software 2024-2025**

# AGENDA

- Metodi **plan-driven** e metodi **agili**
- “Principi” dei metodi agili
- Cosa **non sono** i metodi agili
- Extreme programming (XP)
- Ingredienti (o pratiche) XP:
  - On-site customer
  - User stories
  - Simple design and simple code
  - Pair programming
  - Test Driven Development (TDD)
  - ...
- Esempio ‘TDD’
  - Current account



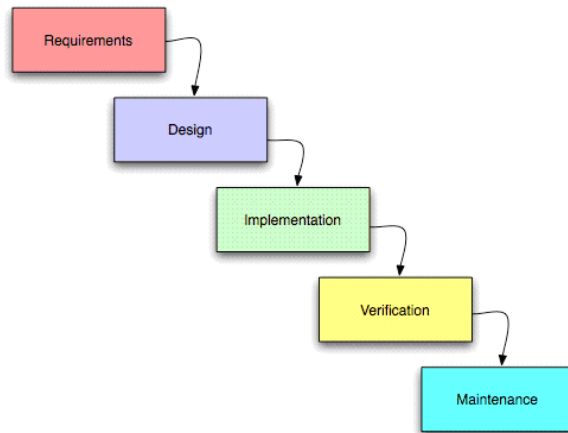
# METODI DI SVILUPPO PLAN-DRIVEN

- **Approccio classico dell'ingegneria del software**  
fondato su processi ben definiti
  - passi standard requisiti/progetto/realizzazione
- Enfasi su:
  - Contratto con il cliente
  - Pianificazione
    - Seguire un piano
  - Processo di sviluppo (e tool)
  - Documentazione (completa)
    - Ogni fase termina con una milestone
      - Produzione artefatti

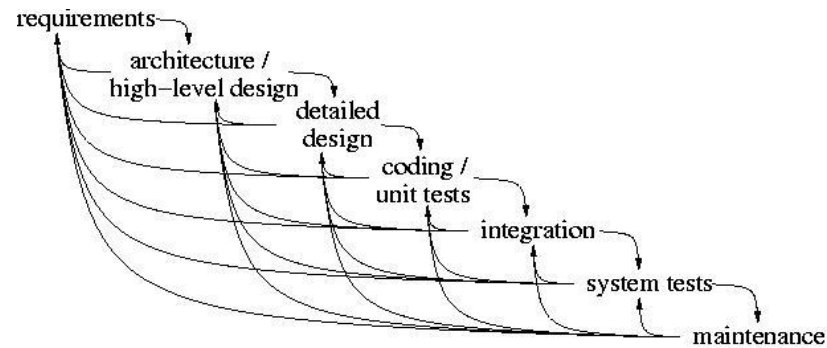


# METODI DI SVILUPPO PLAN-DRIVEN (2)

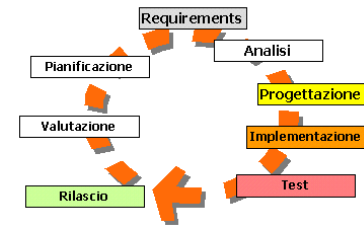
## Cascata



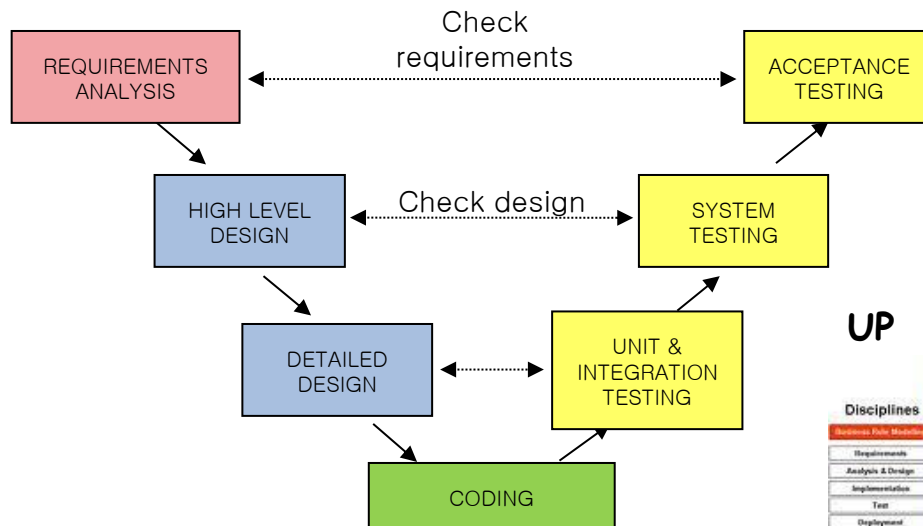
## Cascata con iterazioni e feedback



## Modelli iterativi



## V-model



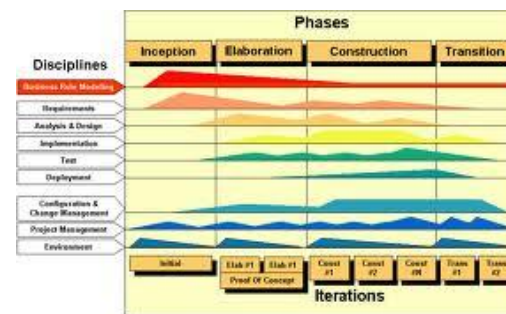
## Modelli incrementali



## Modello a spirale



UP



# CRITICHE AI METODI PLAN-DRIVEN

- Internet economy chiede **flessibilità e velocità**
  - Bisogna **rispondere ai cambiamenti in modo veloce** e lunghi cicli di sviluppo e tanta pianificazione non aiutano ...
- Troppo difficile capire i bisogni del cliente e “formalizzarli” nei requisiti
- Viene messa in crisi: **Qualità del processo => Qualità del prodotto**
  - Che vale nell'industria manifatturiera
  - **Programmare è un arte: non un processo industriale**
  - Software è un ‘prodotto’ particolare!
    - Elemento logico, immateriale, non “fisico”
    - Estremamente “malleabile”
    - Tutti i costi sono nell'ingegnerizzazione non di fabbricazione

**Clienti stupefatti dei continui ritardi**

**Clienti insoddisfatti della bassa qualità del software**

# IL MANIFESTO AGILE (1)

February 11-13, 2001



Salt Lake City rests at the feet of the snow-covered Wasatch Mountain Range.

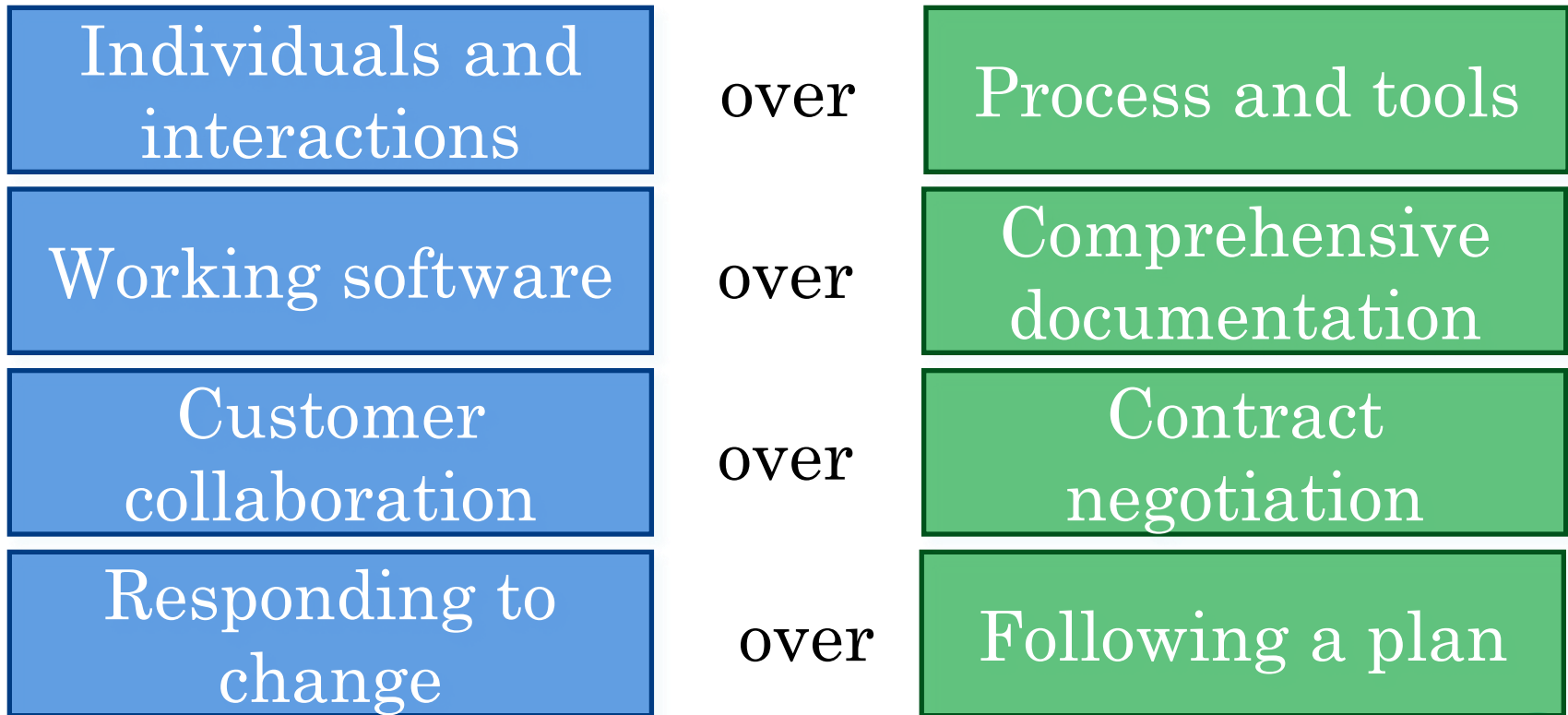
Sottoscritto da: **Kent Beck**, Mike Beedle, Arie van Bennekum, **Alistair Cockburn**, **Ward Cunningham**, **Martin Fowler**, James Grenning, Jim Highsmith, Andrew Hunt, **Ron Jeffries**, Jon Kern, Brian Marick, **Robert C. Martin**, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas.

Source: [www.agilemanifesto.org](http://www.agilemanifesto.org)

# IL MANIFESTO AGILE (2)

Soddisfare il cliente!!!

**I principi** su cui si basa una metodologia agile che segua i punti indicati dall'Agile Manifesto, sono **solo quattro**:



Source: [www.agilemanifesto.org](http://www.agilemanifesto.org)

# COSA NON SONO I METODI AGILI



© Scott Adams, Inc./Dist. by UFS, Inc.

- Non sono “un ritorno alla codifica disordinata e non documentata” ma processi ben definiti che mantengono il controllo sulla qualità del software



# ALCUNE PROPOSTE ...

Ma ne esistono altre ....

Extreme Programming

MSF for Agile Software Development

Agile Modelling

Crystal

Scrum

Lean Development

Feature Driven Development

Adaptive Software Development

Dynamic Systems Development Method

Cosa importante: tutti i metodi condividono gli stessi quattro principi

# COSA È EXTREME PROGRAMMING (XP)?

- XP è un **processo di sviluppo per Software OO** ideato per piccoli gruppi (4-20) che cerca di mantenersi agile e flessibile
  - Non adatto a sistemi grandi e complessi, safety-critical o con forti richieste di affidabilità
- Non usa linguaggi di analisi o design (**UML**), ma dalla raccolta dei requisiti, eseguita con le “**User stories**”, passa **velocemente** alla codifica
- XP è stata ideata nel 1999/2000 da Kent Beck, Ward Cunningham e Ron Jeffries
  - Messo in pratica Durante Progetto **C3**
    - Chrysler Comprehensive Compensation System
    - Doveva rimpiazzare diverse applicazioni per la gestione degli stipendi



# REQUISITI

- I requisiti sono raccolti avendo a disposizione il cliente (**On-site Customer**), per tutta la durata del progetto, tramite **“User stories”**
  - Addirittura è il cliente stesso a scriverli!
- Una **“user story”** è un sintetico “caso d’uso” che viene scritto in linguaggio naturale su una scheda / post-it
- Esempio:

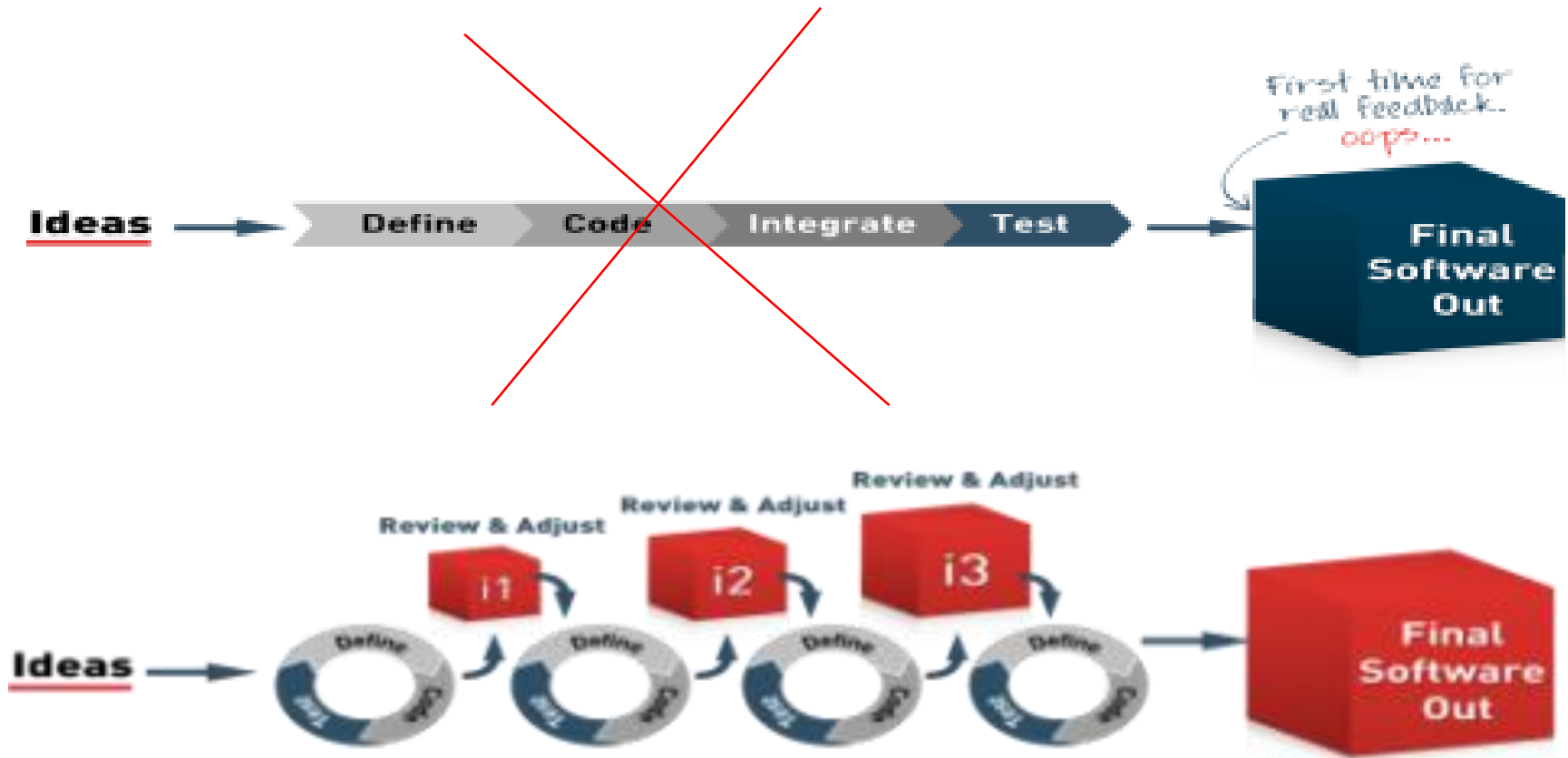
A photograph of a white sticky note with red handwritten text. The text reads: "As a librarian, I want to be able to search for books by publication year." The note is placed on a light-colored surface with horizontal lines.

# ESEMPIO COMPLETO DI REQUISITI



# RILASCI FREQUENTI

Le release aiutano il customer a comprendere meglio i reali bisogni!

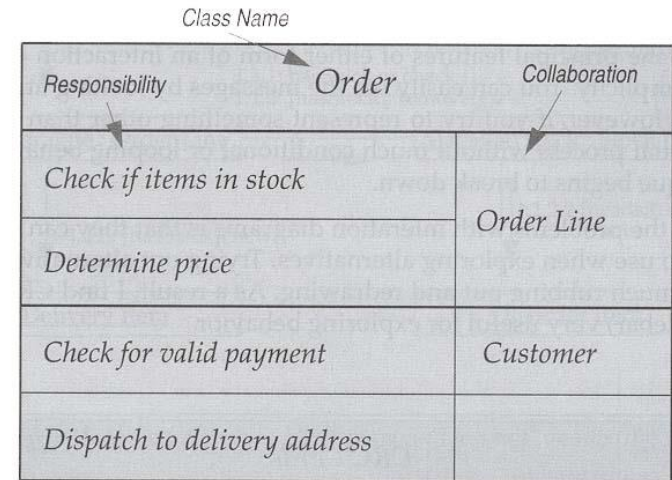


Invece di un rilascio unico XP prevede frequenti rilasci di codice funzionante ogni 1-4 settimane (continuous integration / working software)

# COME SI PRODUCE IL DESIGN?

## ○ Metodo 'CRC'

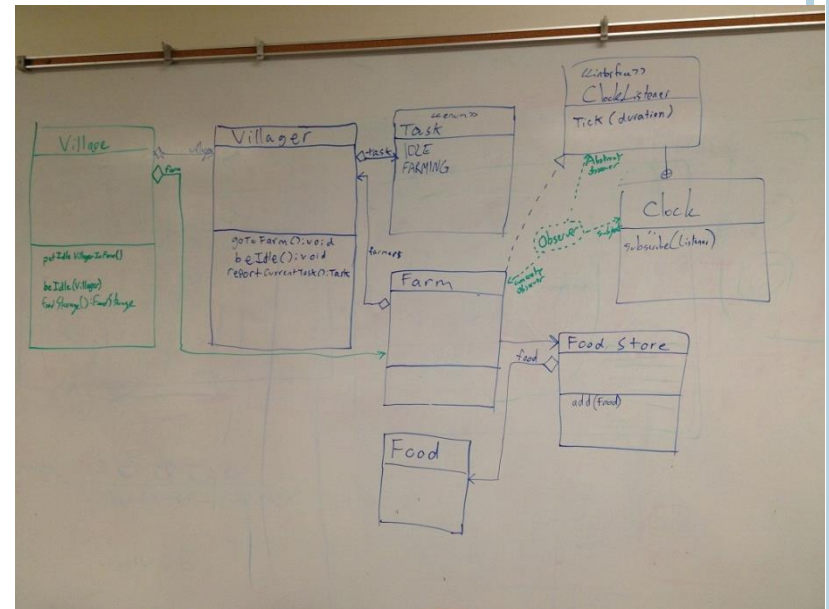
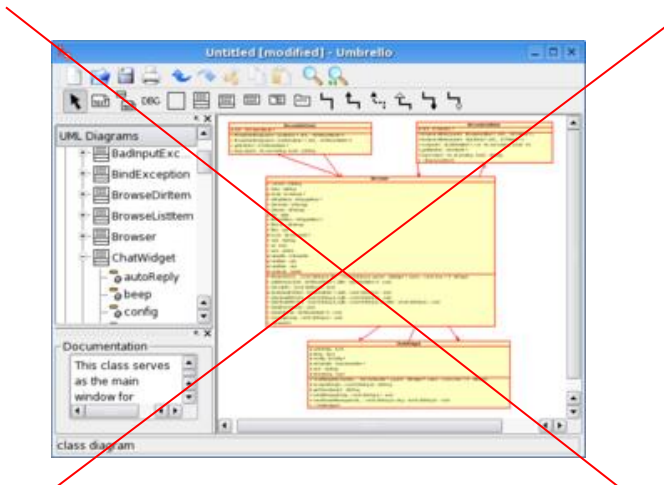
- **Class Responsibility Collaboration**
- Si usano foglietti adesivi su cui vengono scritti il nome della classe, le responsabilità e le collaborazioni con le altre classi



Class-Responsibility-Collaboration (CRC) Card

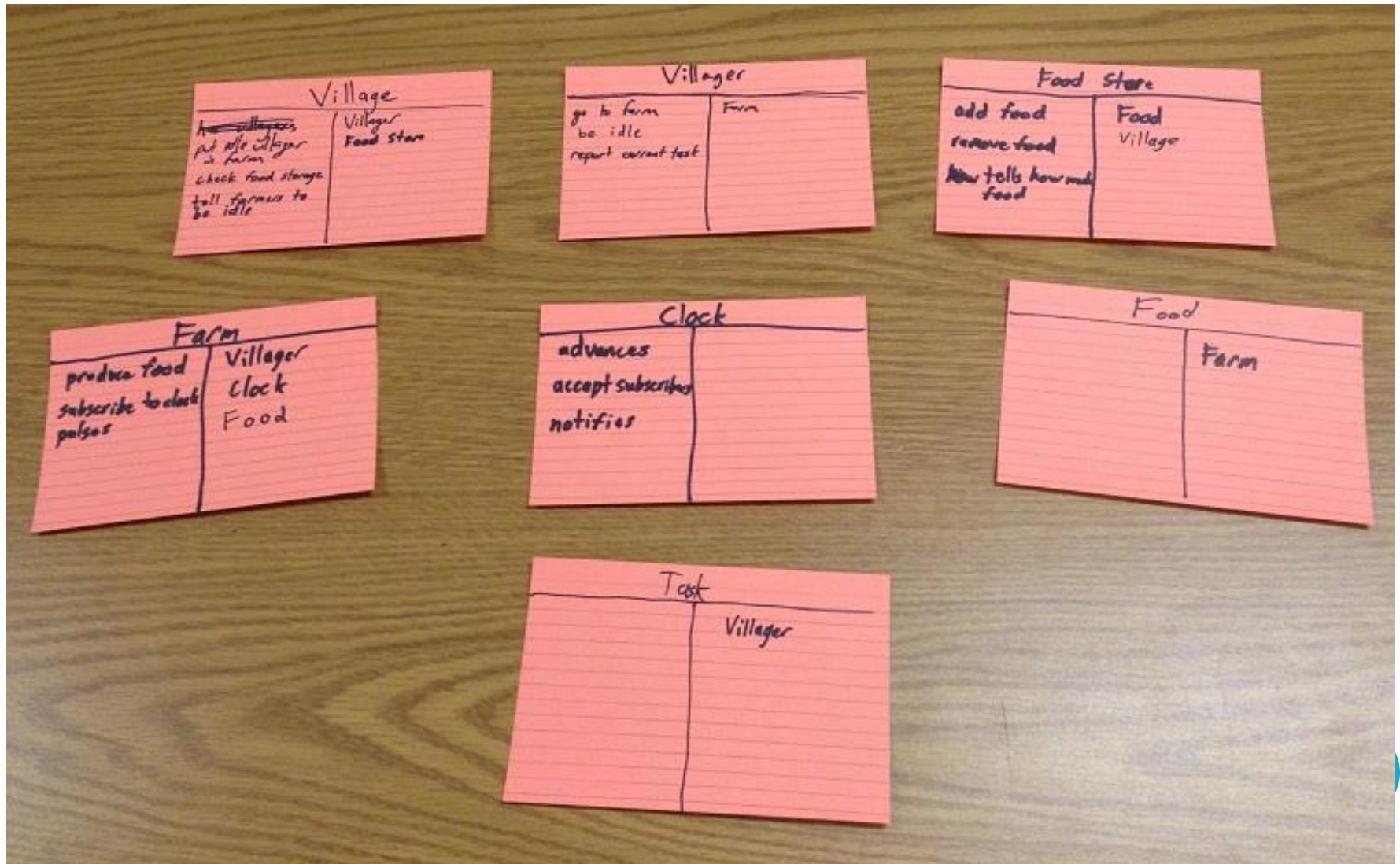
## ○ Non si usano tool per il design e nemmeno linguaggi tipo UML

- Solo in casi particolari: “Quick design session”

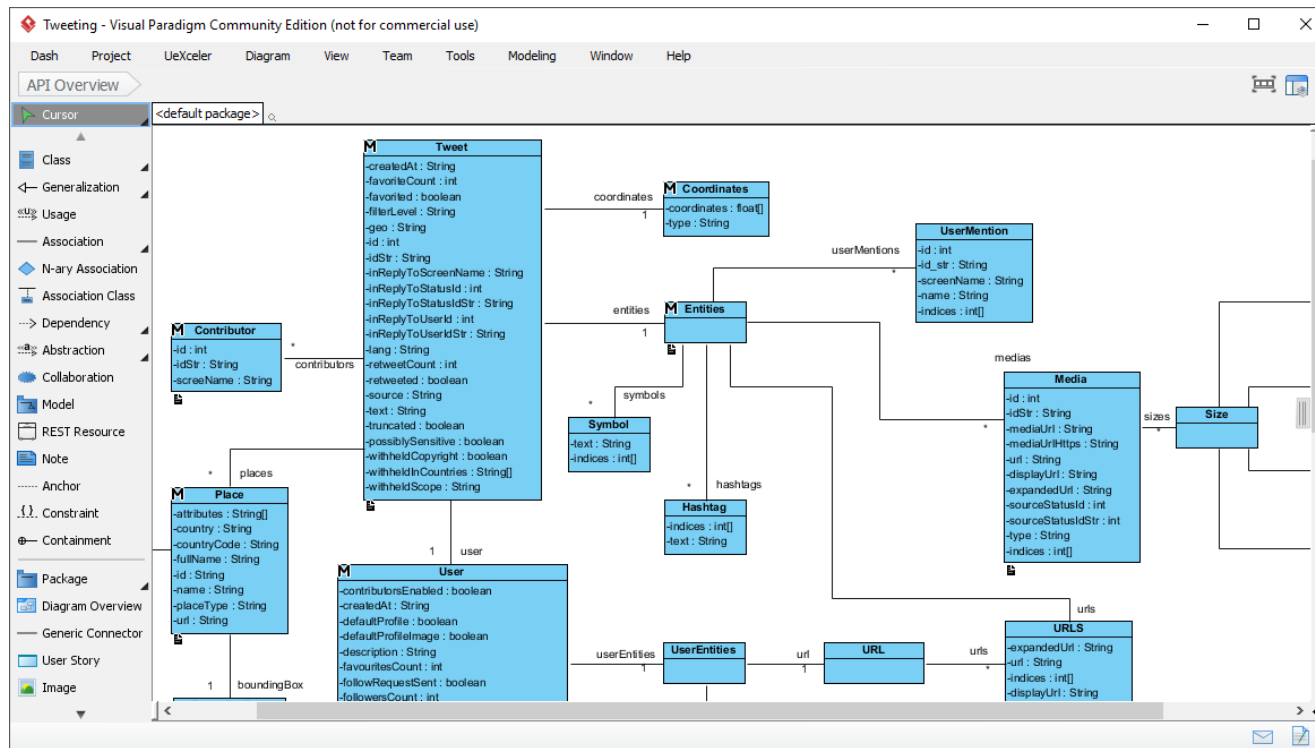




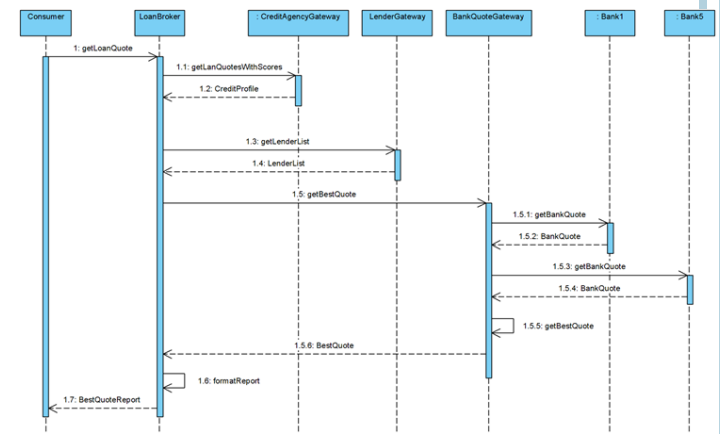
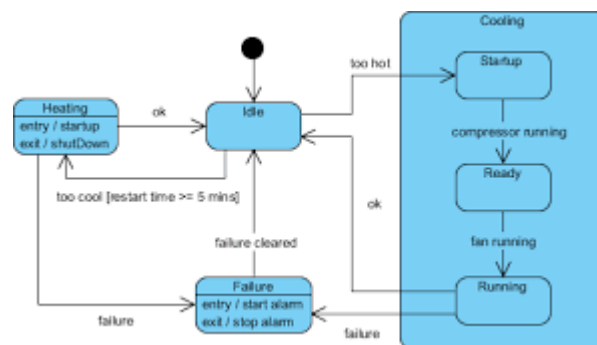
# ESEMPIO DI DESIGN XP (CON CRC CARDS)



# MOLTO DIVERSO DA ...



Metodi Plan-driven





# DOCUMENTI DI DESIGN

- **Non si conservano diagrammi di progetto!!!!**

- Questi sono “incorporati” nel software e facilmente desumibili
  - Vale: **Tacit/implicit knowledge**

**Il design è nella testa degli sviluppatori!**

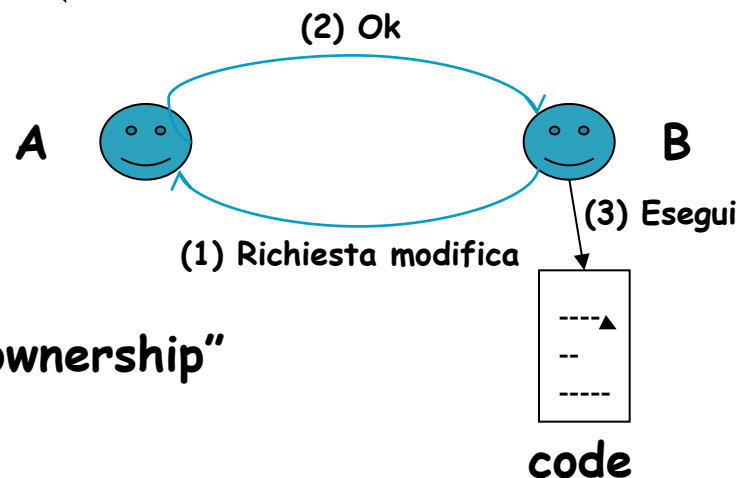


- Per avere sempre ben chiaro il design è importante che il codice sia il più **autodocumentato** e **“chiaro”** possibile (**meaningful code**)

- Questo non si ottiene con i commenti ma tramite il **refactoring** e assegnando alle variabili nomi significativi

# PRINCIPI DI CODIFICA (1)

- **Non si pianifica per il riuso**, ma si sviluppa nel modo più semplice possibile
  - **simple code and simple design (principio KISS)**
- **“Think by example”**. Si inizia con un esempio concreto, si scrive un algoritmo “ad hoc” e poi si generalizza e si cercano i casi particolari
- Il codice è in comune. Chiunque può modificare il codice scritto da altri (**collective code ownership**)



**“Individual code ownership”**

**NO!**

## PRINCIPI DI CODIFICA (2)

- Tutti i programmatori devono attenersi **alle stesse** “code conventions”

The diagram illustrates Java coding conventions with several annotations pointing to specific parts of the code:

- Javadoc conventions:** see <http://java.sun.com/products/jdk/javadoc/writingdoccomments/index.html> (points to the `/** @deprecated` comment).
- One-liners are OK** (points to the single-line `return` statement in `getName()`).
- JavaBeans naming conventions:** see <http://java.sun.com/beans/docs/beans.101.pdf> (points to the `getScore()` method name).
- Braces here and here** (points to the opening and closing braces of the `if` statement in `getScore()`).
- 4-space indent throughout** (points to the indentation of the `return` statement in `getScore()`).

```
/** @deprecated Use getLastname(), not getName() */
public String getName() {return lastname;}

public int getScore() {
    if (height < MIN_SIZE) {
        return MIN_SIZE;
    }
    return height;
}
```

**Utilizzo di tool per il controllo!**  
(es. Checkstyle e PMD)

# PAIR PROGRAMMING

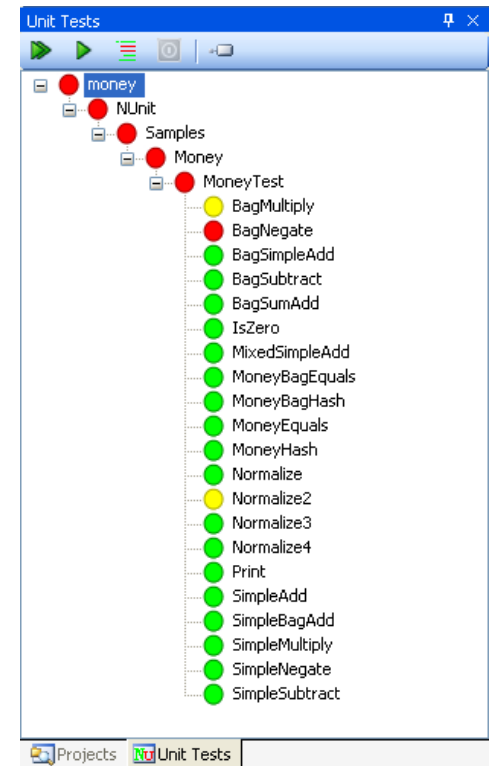


- I programmatori lavorano a coppie:
  - uno digita il codice e l'altro osserva e commenta
  - solitamente il meno esperto scrive il codice
- I componenti della coppia di volta in volta cambiano ....
- Principio controverso di XP
  - È davvero una pratica che migliora le cose?
- Punti di forza:
  - revisione del codice **continua**
  - training
  - ogni modulo è conosciuto in dettaglio da almeno due persone

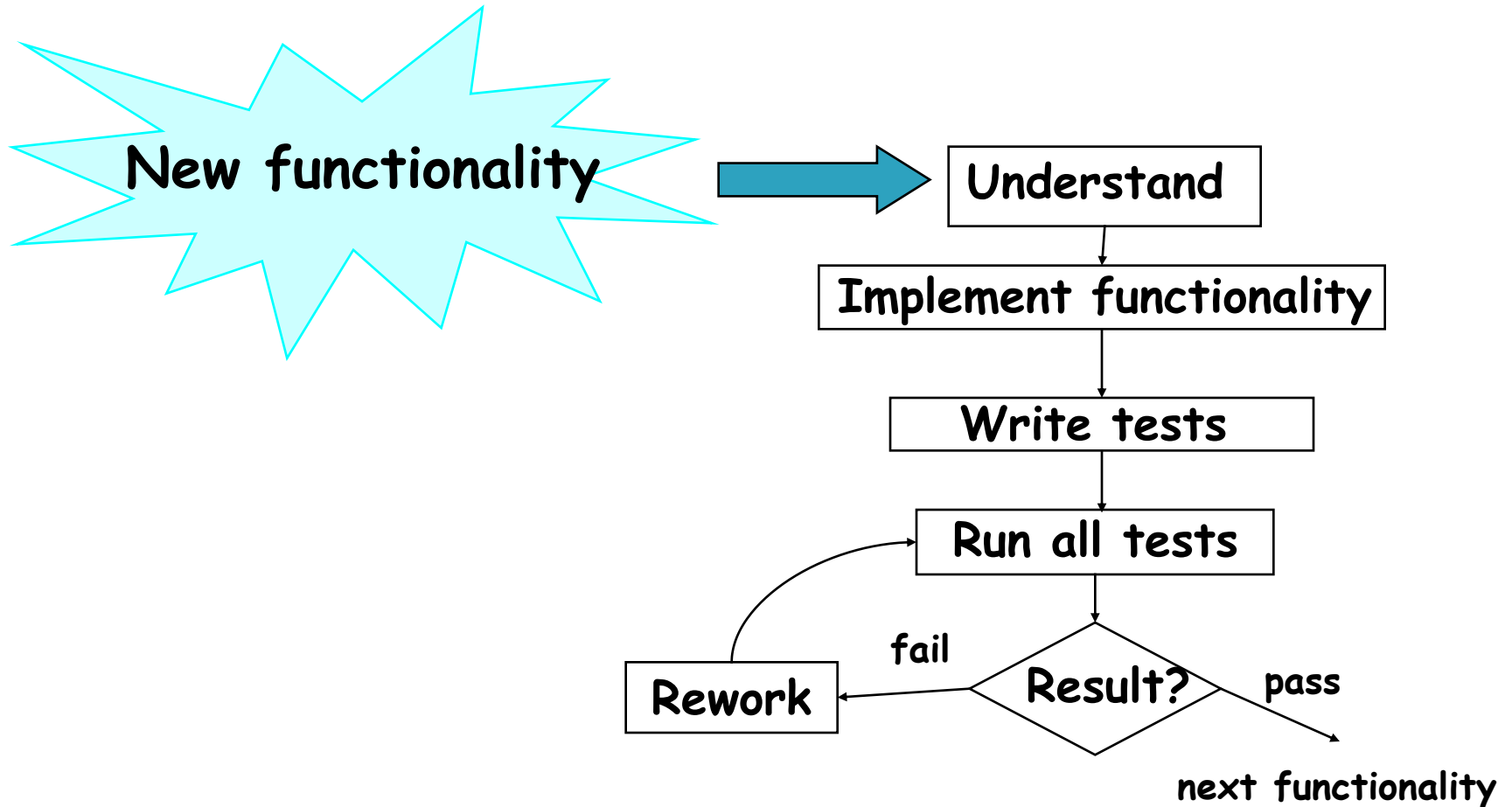
# UNIT TESTING

- **Per ogni classe** sono scritti dei casi di test per verificarne il funzionamento
- Uso di tool/framework specifici
  - **Test Automation**
    - esempio **JUnit**
- Ad ogni modifica della classe tutti i casi di test sono eseguiti e devono essere tutti superati
- I casi di test sono progettati prima del codice stesso perchè aiutano nello sviluppo
  - **Testing guida lo sviluppo!**
  - **Test driven development (TDD) approach**

**Green = passed**  
**Red = failure**

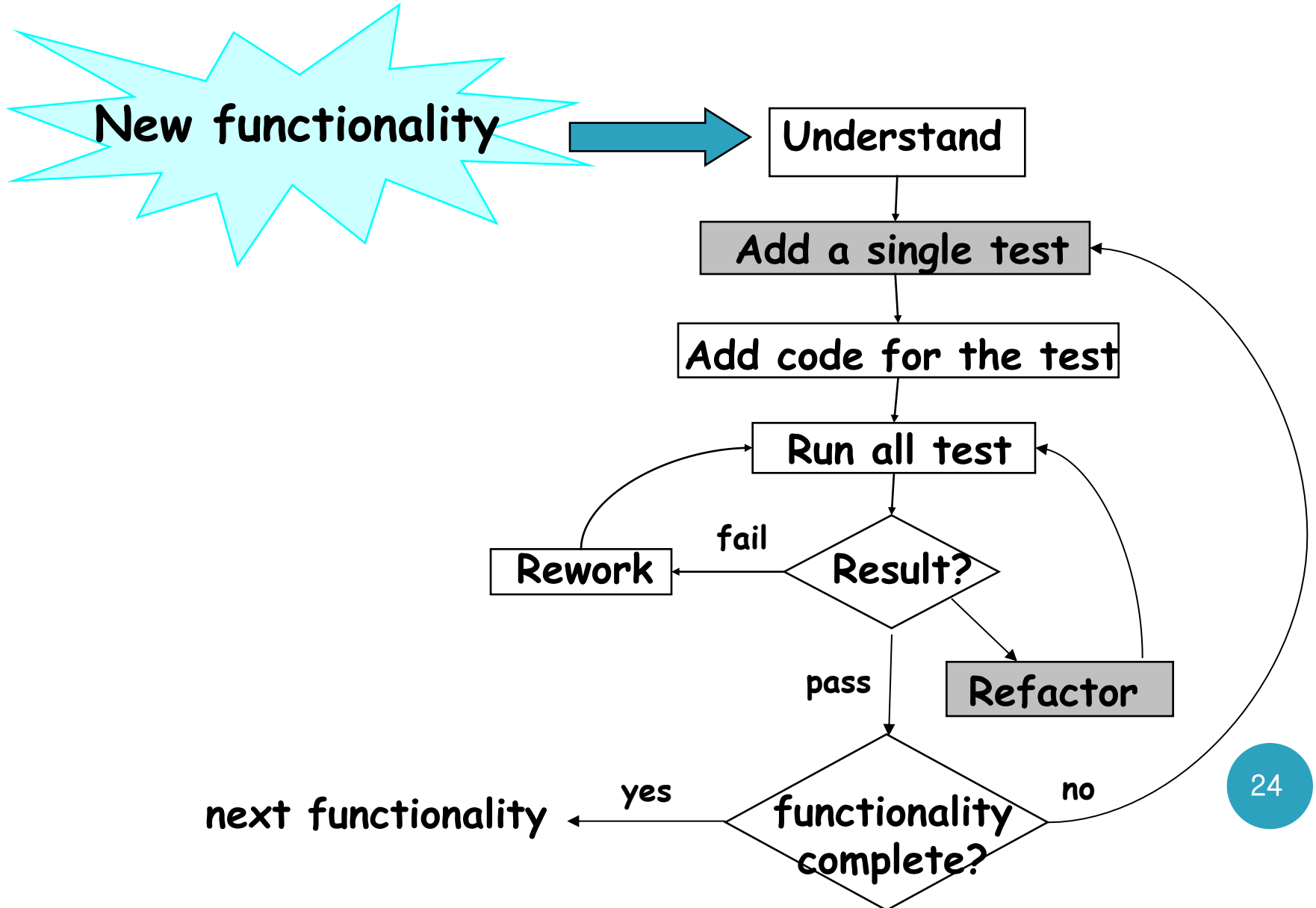


# 'TEST LAST' DEVELOPMENT



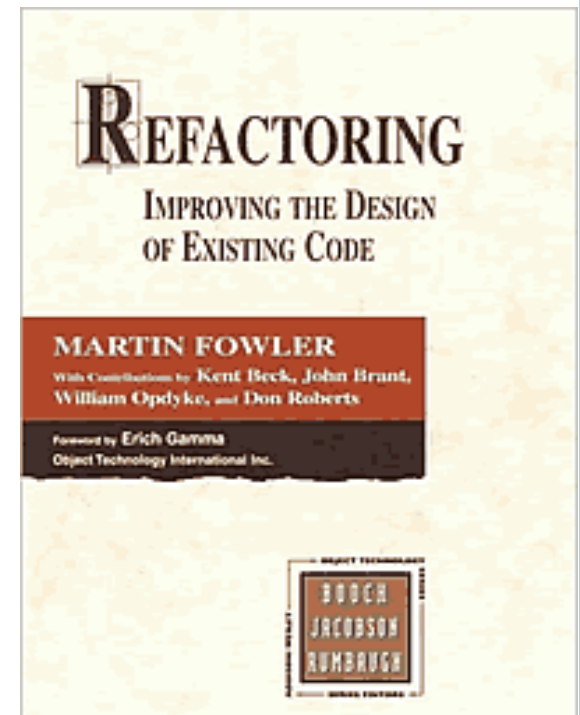
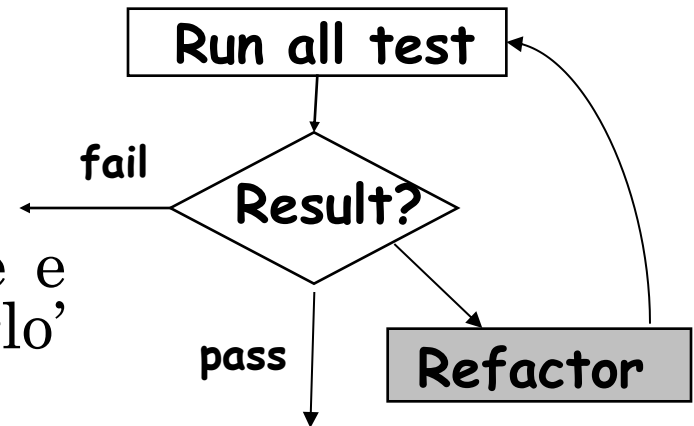
"new functionality" = class with methods

# TEST DRIVEN DEVELOPMENT (TDD)



# REFACTORING

- Per mantenere il codice semplice e meaningful occorre 'ristrutturarlo' spesso
- Appena il codice supera i casi di test di unità si applica il **refactoring**
- Uno dei principi che guida il programmatore XP è: “**Once and Only Once**”. Se due segmenti di codice fanno la stessa cosa (cloni), devono essere unificati in un unico modulo
- La **ristrutturazione continua** è possibile perchè si dispone dei casi di test di unità
  - Posso modificare il software senza paura!

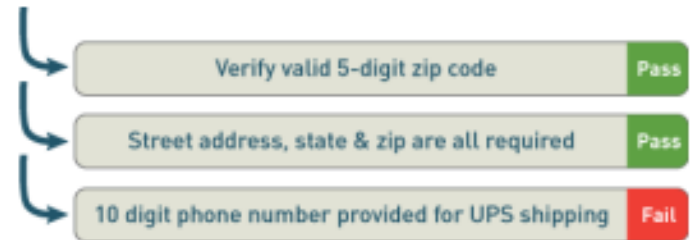




# ACCEPTANCE TESTING

- Sono previsti dei **test funzionali di accettazione**, basati sulle “User stories” e concordati con il cliente
  - Verificano che la user story è stata implementata correttamente
- Durante lo sviluppo i test di accettazione possono anche non essere superati al ‘100%’
  - Vale anche nei rilasci intermedi
- La percentuale di test di accettazione **superati** è un **indice del progresso** nella codifica del sistema

**User Story 23**  
In order to make sure my package arrives at my house as a website customer, I want to validate my shipping information



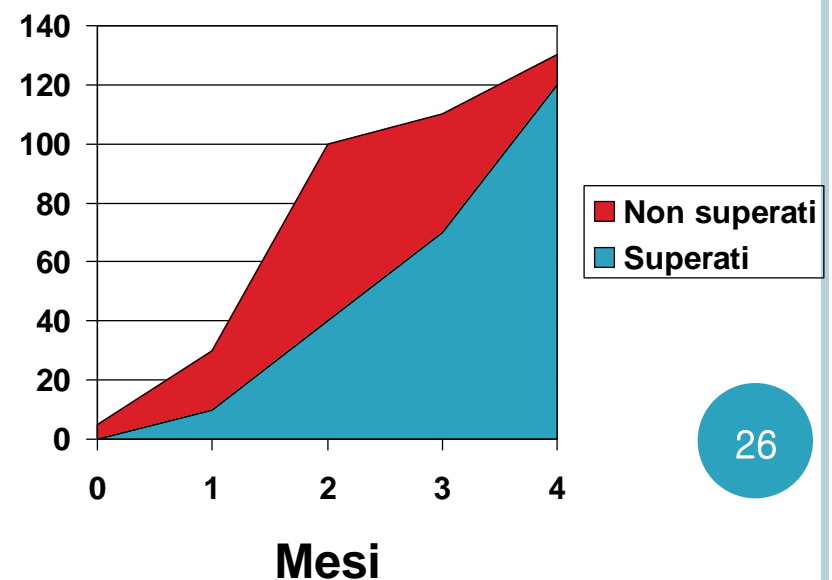
**Address Line1:**   
Street address, P.O. box, company name, c/o

**Address Line2:**   
Apartment, suite, unit, building, floor, etc.

**City:**

**State/Province/Region:**

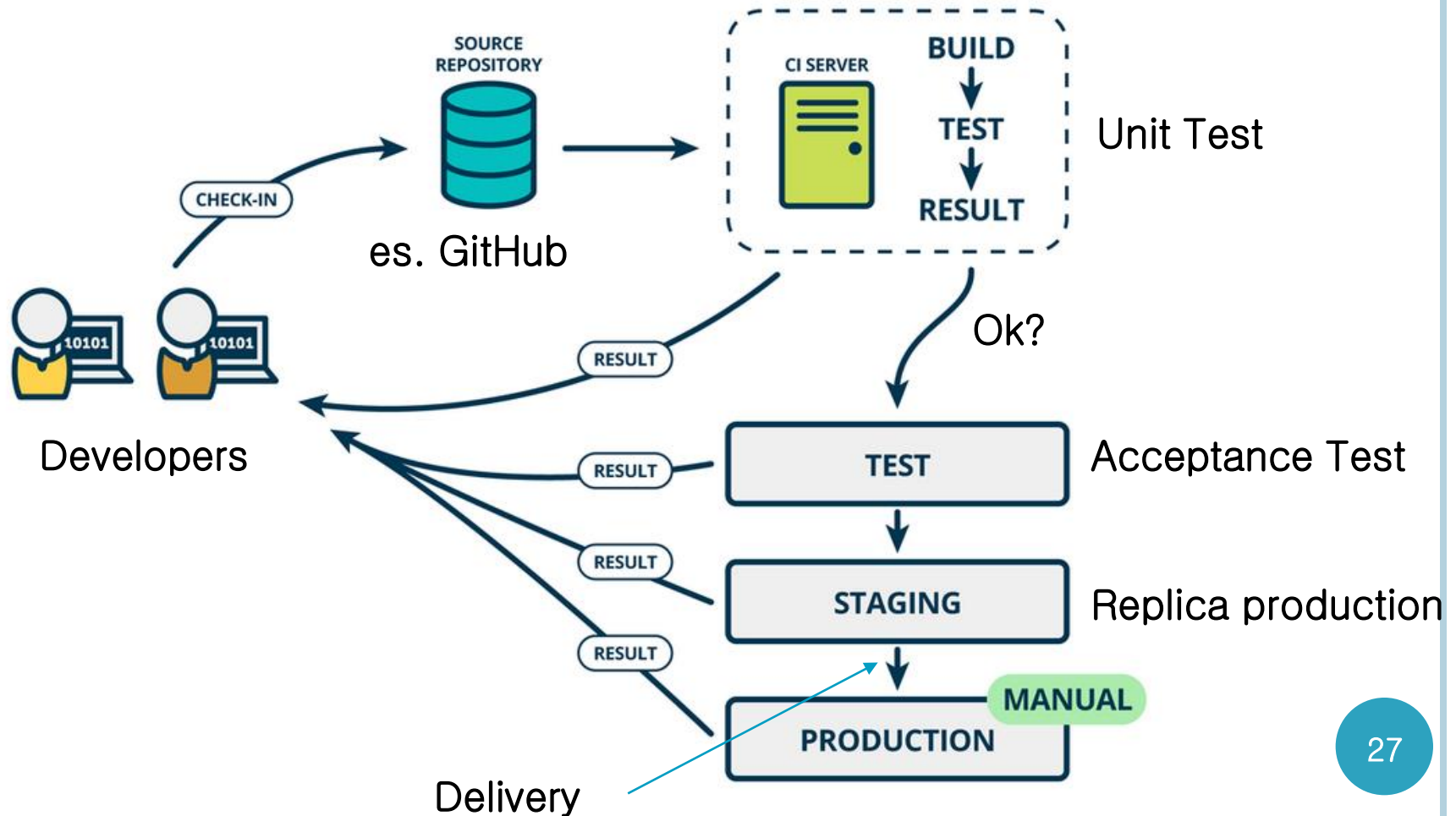
**ZIP/Postal Code:**



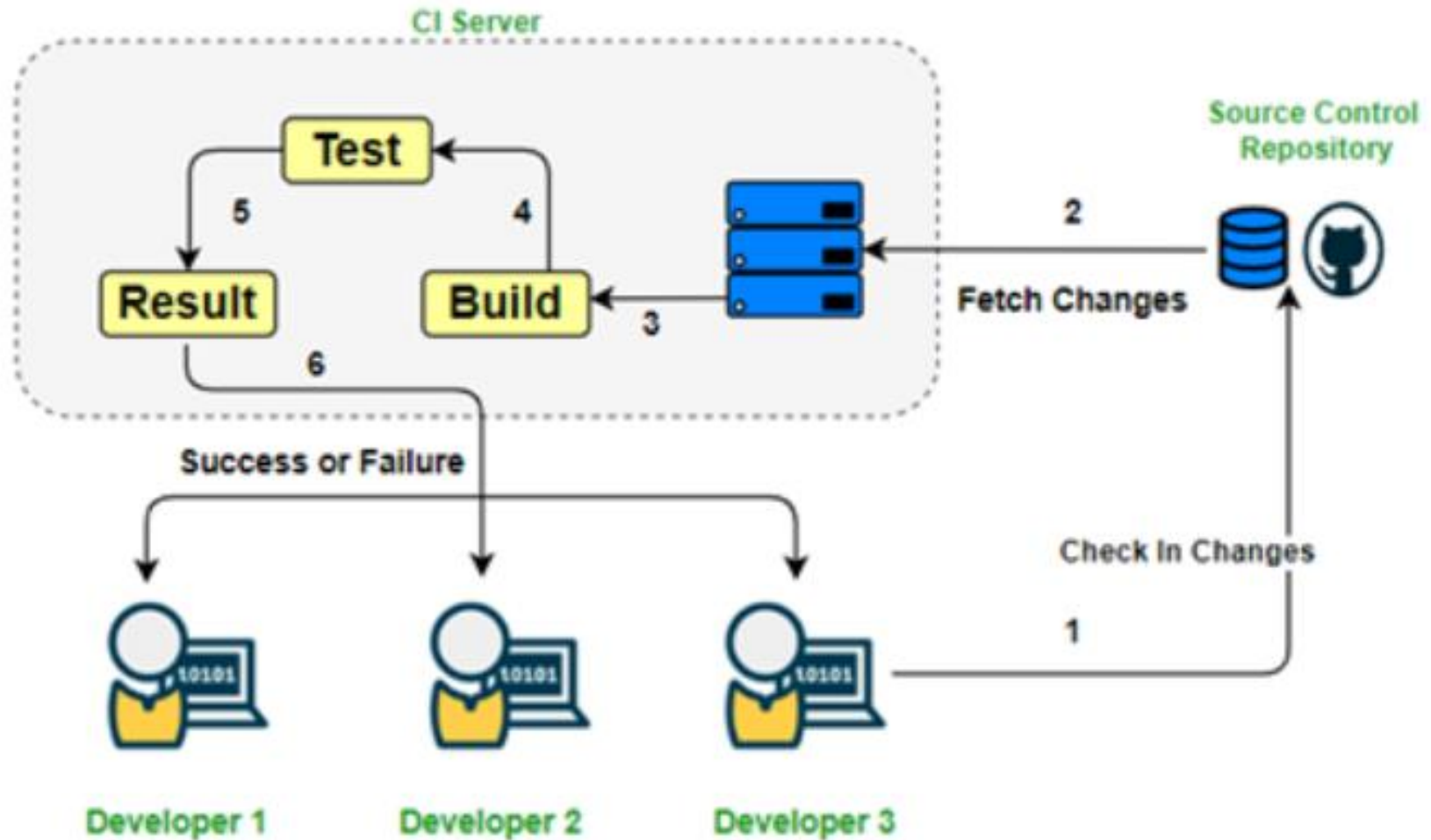
Il numero e nome degli ambienti/server può cambiare

# MODELLO DI SVILUPPO MODERNO (DEVOPS)

(BASATO SU CONTINUOUS INTEGRATION E CONTINUOUS DELIVERY)



# CONTINUOUS INTEGRATION



# PROBLEMI DEI METODI AGILI

## ◦ Gestionali

- La conoscenza è nella testa degli sviluppatori, cosa succede se c'è un alto turnover?
- Si addice solo a sviluppatori molto bravi ....
- Di fatto gli utenti potrebbero non essere sempre disponibili (customer on-site) e allora come si fa?

## ◦ Contrattuali/Legali

- Quando è soddisfatto il contratto? Di fatto non esiste ...

## ◦ Manutenzione

- Documentazione scarsa
- Architettura spesso complessa determinata dalla release attuale ...
  - Spesso evolve in modo casuale ....

# METODI PLAN-DRIVEN VS. METODI AGILI

- Ciascuno degli approcci si trova a proprio agio se “**gioca in casa**”



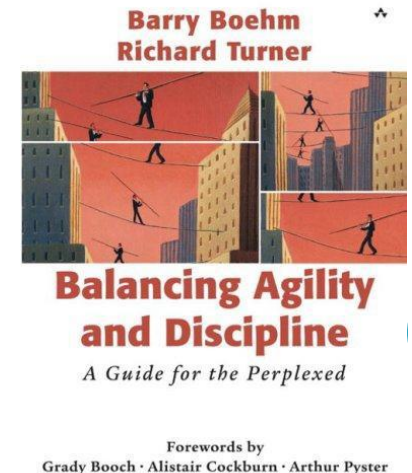
- **Metodi plan-driven:**

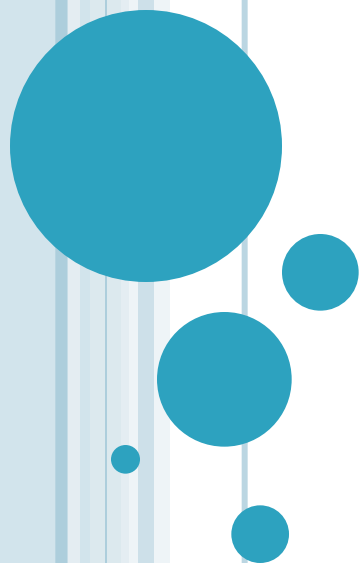
- sistemi grandi e complessi, **safety-critical** o con forti richieste di affidabilità
- requisiti stabili e ambiente predicibile

- **Metodi agili:**

- sistemi e team piccoli, clienti e utenti disponibili, ambiente e requisiti volatili
- team con molta esperienza
- tempi di consegna rapidi

- Ma cosa fare nei casi dubbi?





# **ESEMPIO TDD**

**Current Account**

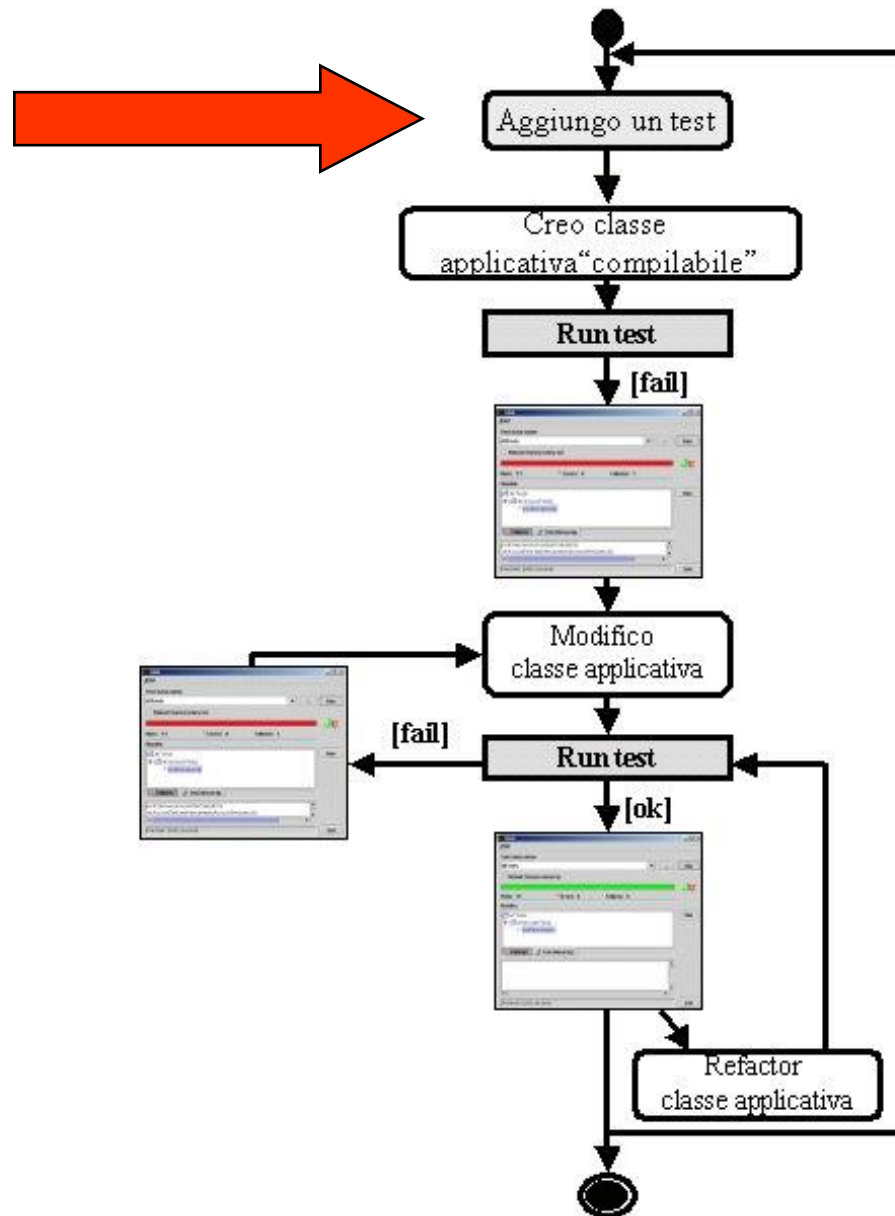
# TDD IN PRATICA CON JUNIT

- **User story**: “Creare una classe **conto corrente** che permette il prelievo, il deposito ed il saldo”
- Current account class (o bank account) con tre operazioni:
  - deposit (*deposito*)
  - withdraw (*prelievo*)
  - settlement (*saldo*)

## Esempio:

```
Account cc = new Account();  
cc.deposit(12);  
cc.draw(-8);  
cc.deposit(10);  
cc.settlement()
```

expected value      **14 euro!**





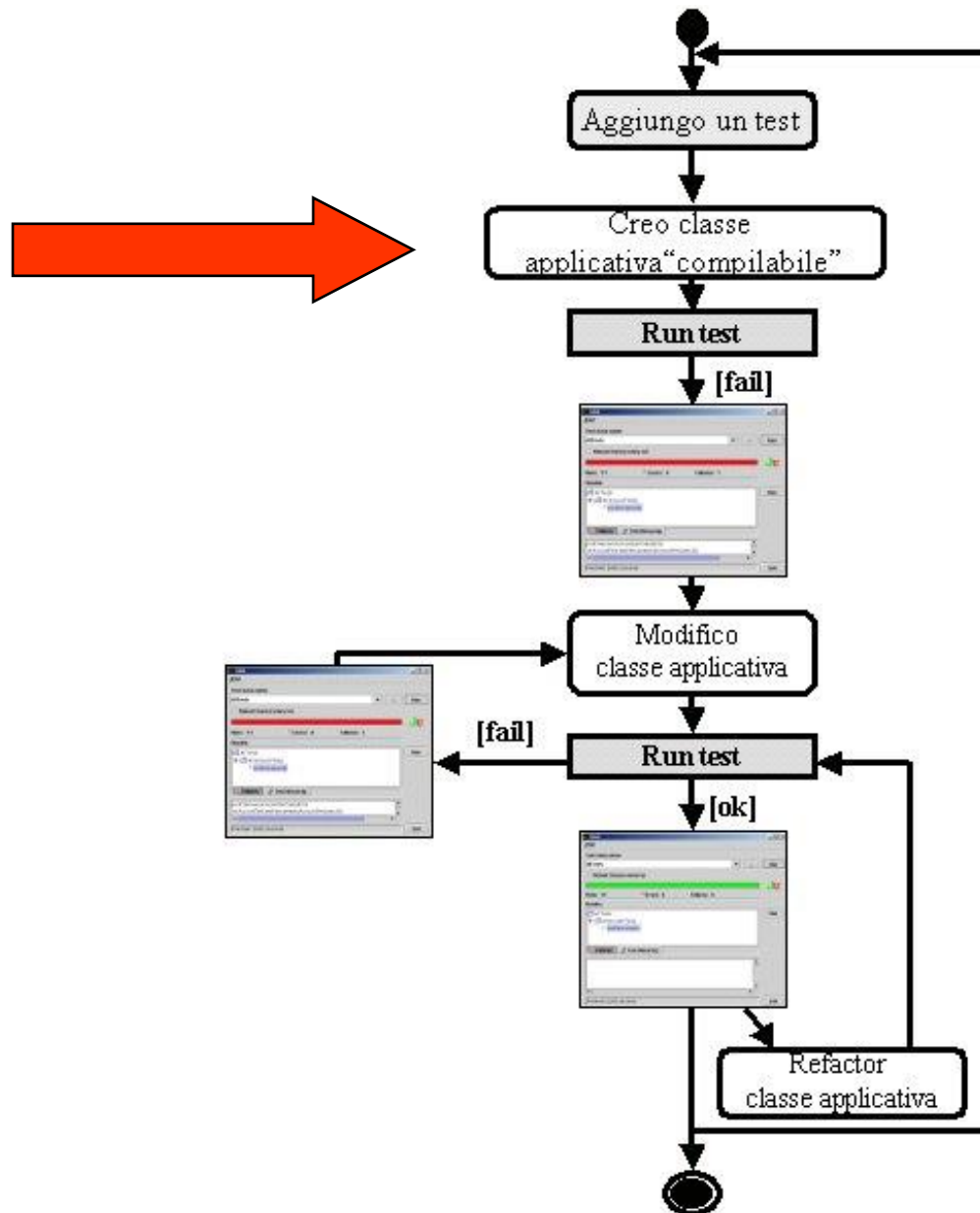
# AGGIUNGO IL PRIMO TEST

Si inizia con un caso concreto: Think by example!

Specifica al  
testing  
framework JUnit  
che questo è un  
test method

Assertazione  
JUnit: se falsa  
durante  
l'esecuzione  
determina il  
fallimento del  
test (failure) e  
'barra rossa'

```
public class Test_account {  
    @Test  
    public void testSettlement() {  
        cAccount c = new Account();  
        c.deposit(12);  
        c.draw(-8);  
        c.deposit(10);  
        assertTrue(c.settlement() == 14);  
    }  
}
```



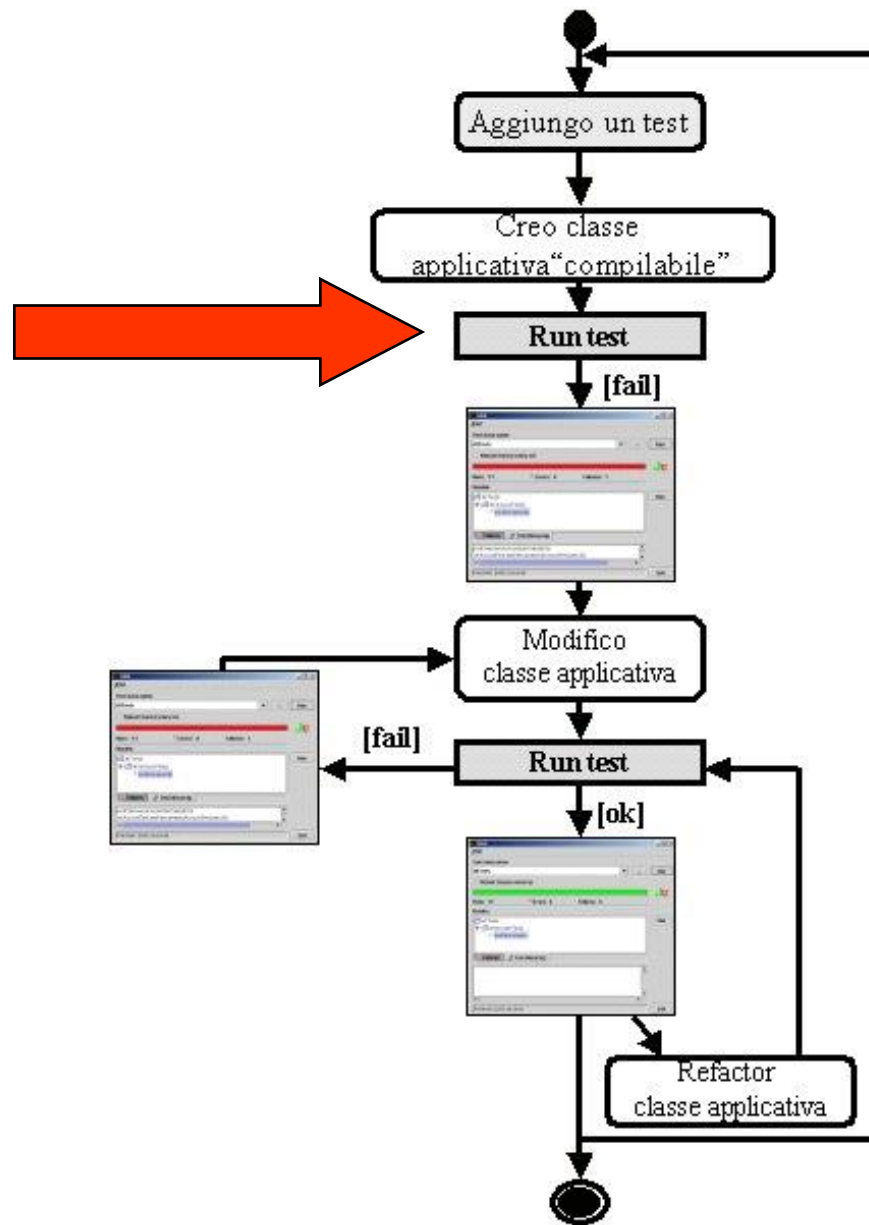
# CREO LO SCHELETRO DELLA CLASSE

```
public class Account {  
  
    public void deposit(int i) {  
        // Auto-generated method stub  
    }  
  
    public void draw(int i) {  
        // Auto-generated method stub  
    }  
  
    public int settlement() {  
        // Auto-generated method stub  
        return 0;  
    }  
}
```

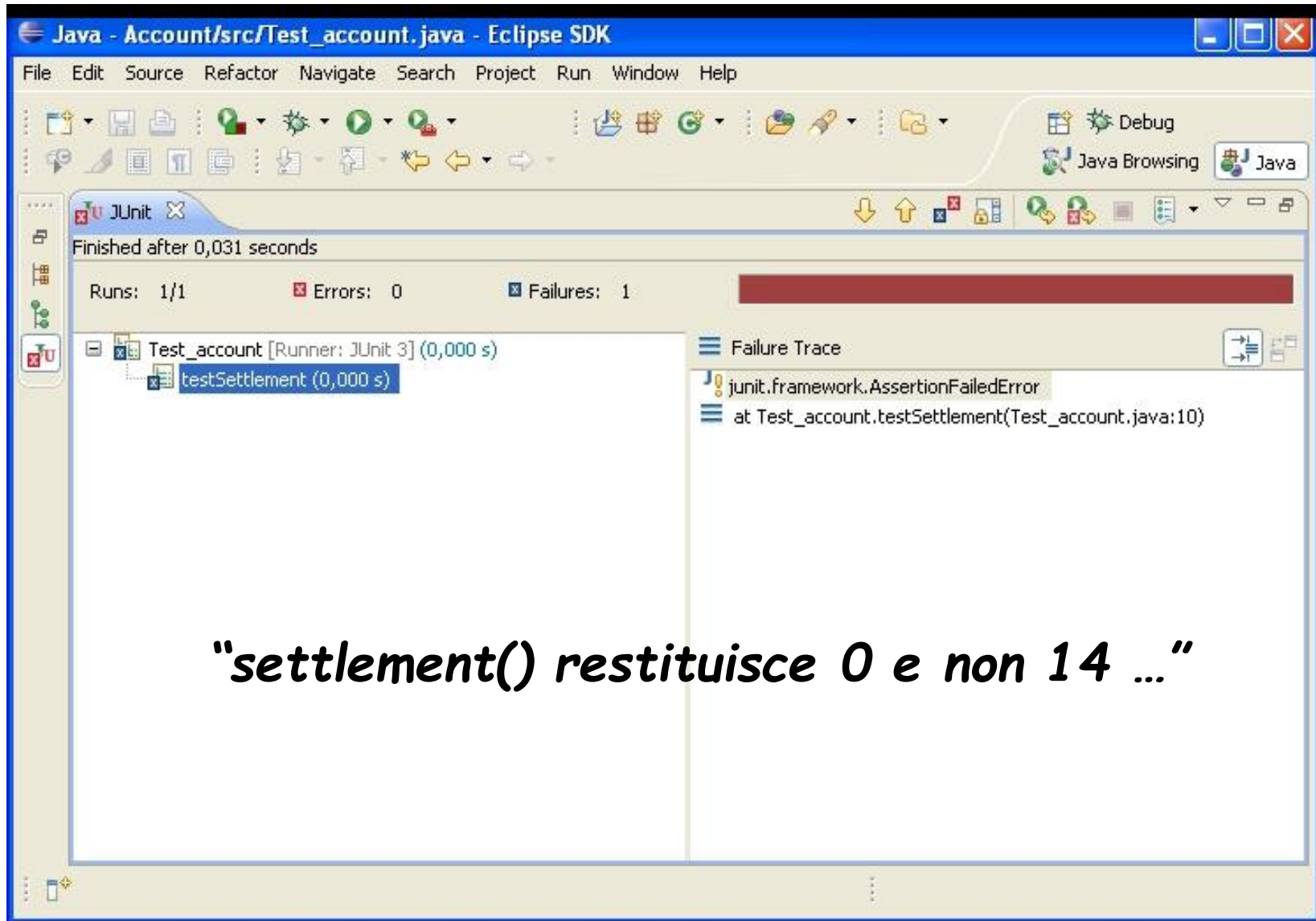
```
public class Test_account {  
  
    @Test  
    public void testSettlement() {  
        Account c = new Account();  
        c.deposit(12);  
        c.draw(-8);  
        c.deposit(10);  
        assertTrue(c.settlement() == 14);  
    }  
}
```



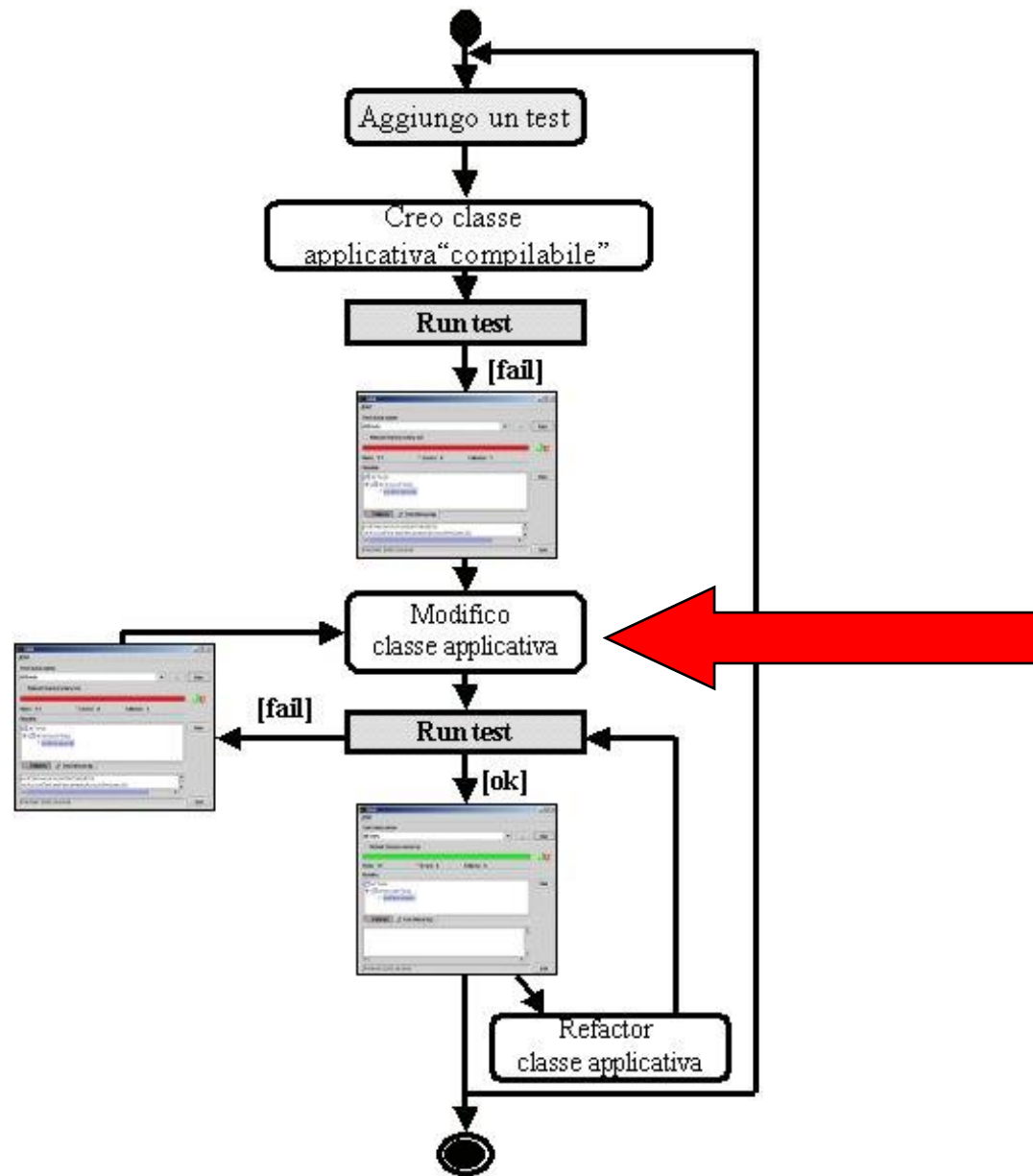
***"Auto-completion Eclipse"***



# RUN JUNIT (1)



*"settlement() restituisce 0 e non 14 ..."*

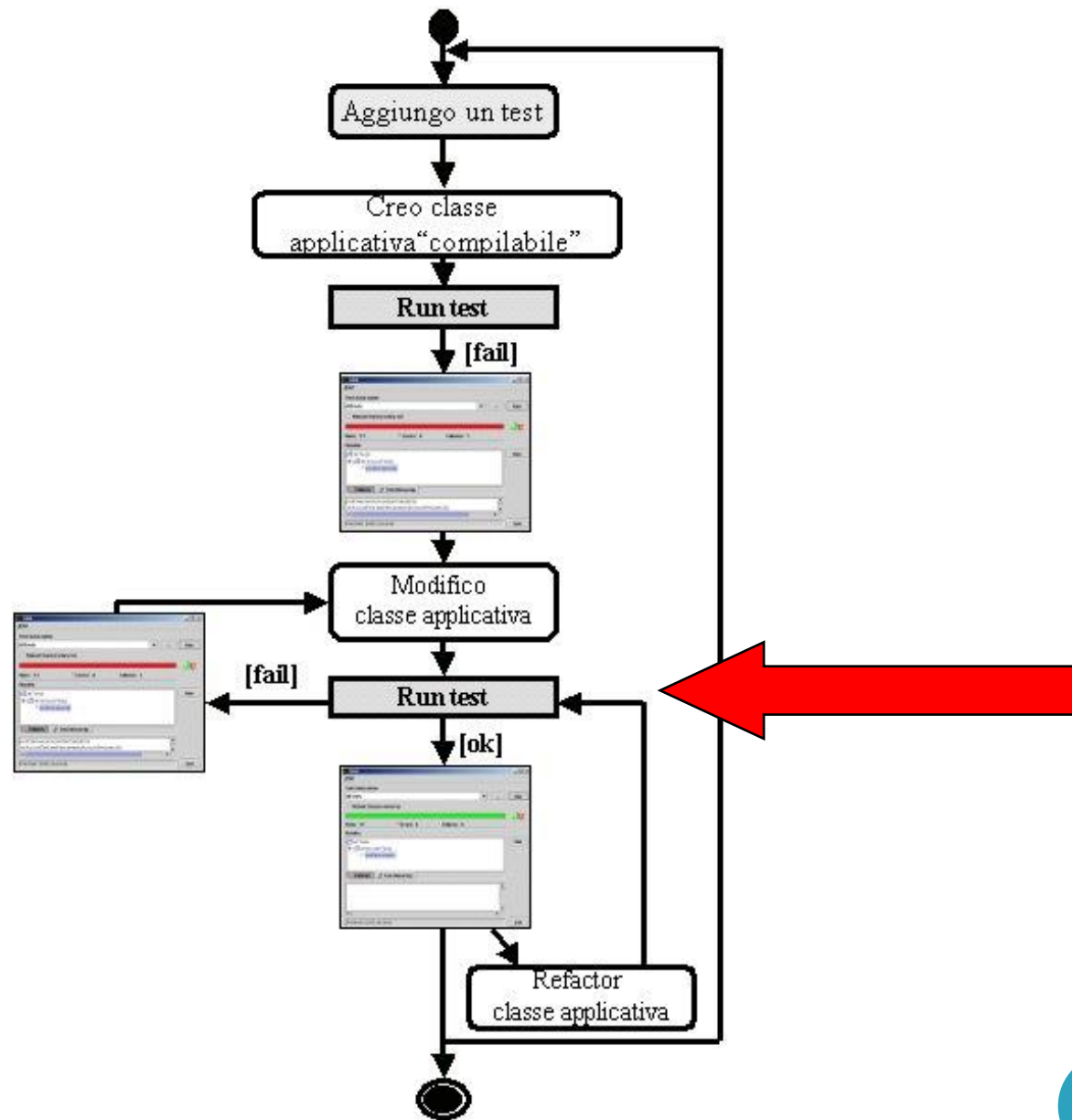


*'Primo posto libero'*

```
public class Account {  
    int account[]; int lastMove;  
  
    public Account() {  
        lastMove=0; account=new int[10];  
    }  
  
    public void deposit(int value) {  
        account[lastMove]=value;  
        lastMove++;  
    }  
  
    public void draw(int value) {  
        account[lastMove]=value;  
        lastMove++;  
    }  
  
    public int settlement() {  
        int result = 0;  
        for (int i=0; i<account.length; i++) {  
            result = result + account[i];  
        }  
        return result;  
    }  
}
```

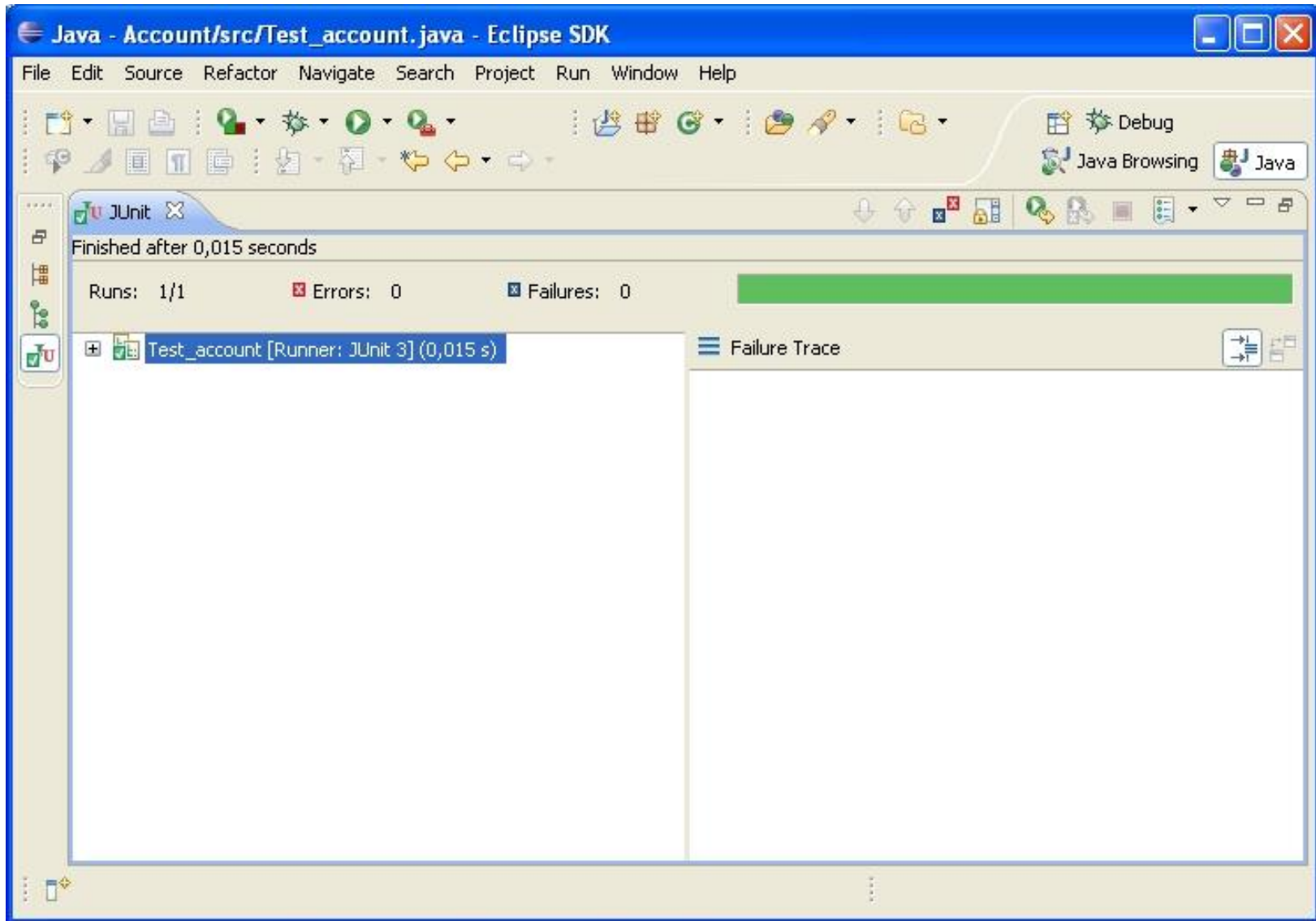
## AGGIUNGO IL CODICE

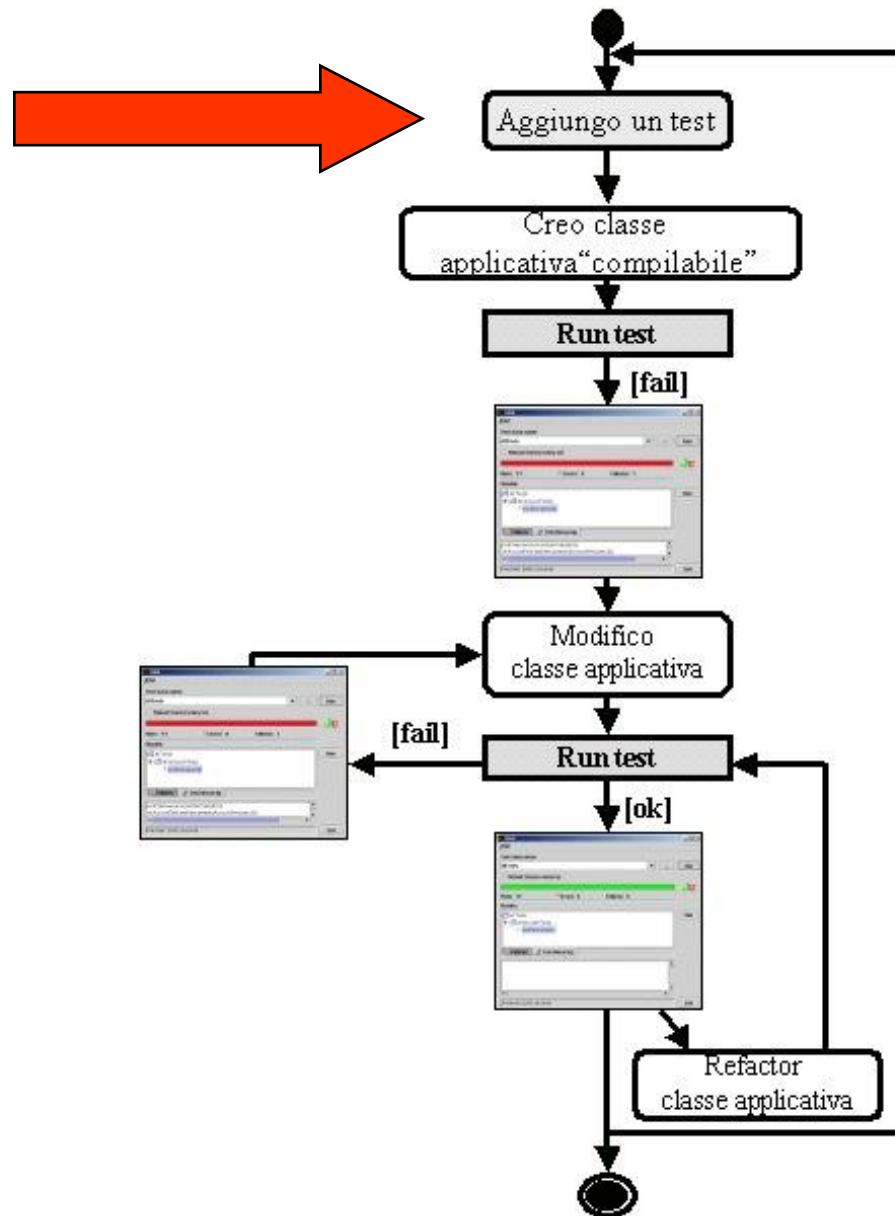
```
public class Test_account {  
  
    @Test  
    public void testSettlement() {  
        cAccount c = new cAccount();  
        c.deposit(12);  
        c.draw(-8);  
        c.deposit(10);  
        assertTrue(c.settlement() == 14);  
    }  
}
```





# RUN JUNIT (2)





```

public class Account {
int account[]; int lastMove;

public Account() {
    lastMove=0; account=new int[10];}

public void deposit(int value) {
    account[lastMove]=value;
    lastMove++;
}

public void draw(int value) {
    account[lastMove]=value;
    lastMove++;
}

public int settlement() {
    int result = 0;
    for (int i=0; i<account.length; i++) {
        result = result + account[i];
    }
    return result;
}
}

```

## AGGIUNGO UN ALTRO TEST (+ COMPLESSO)

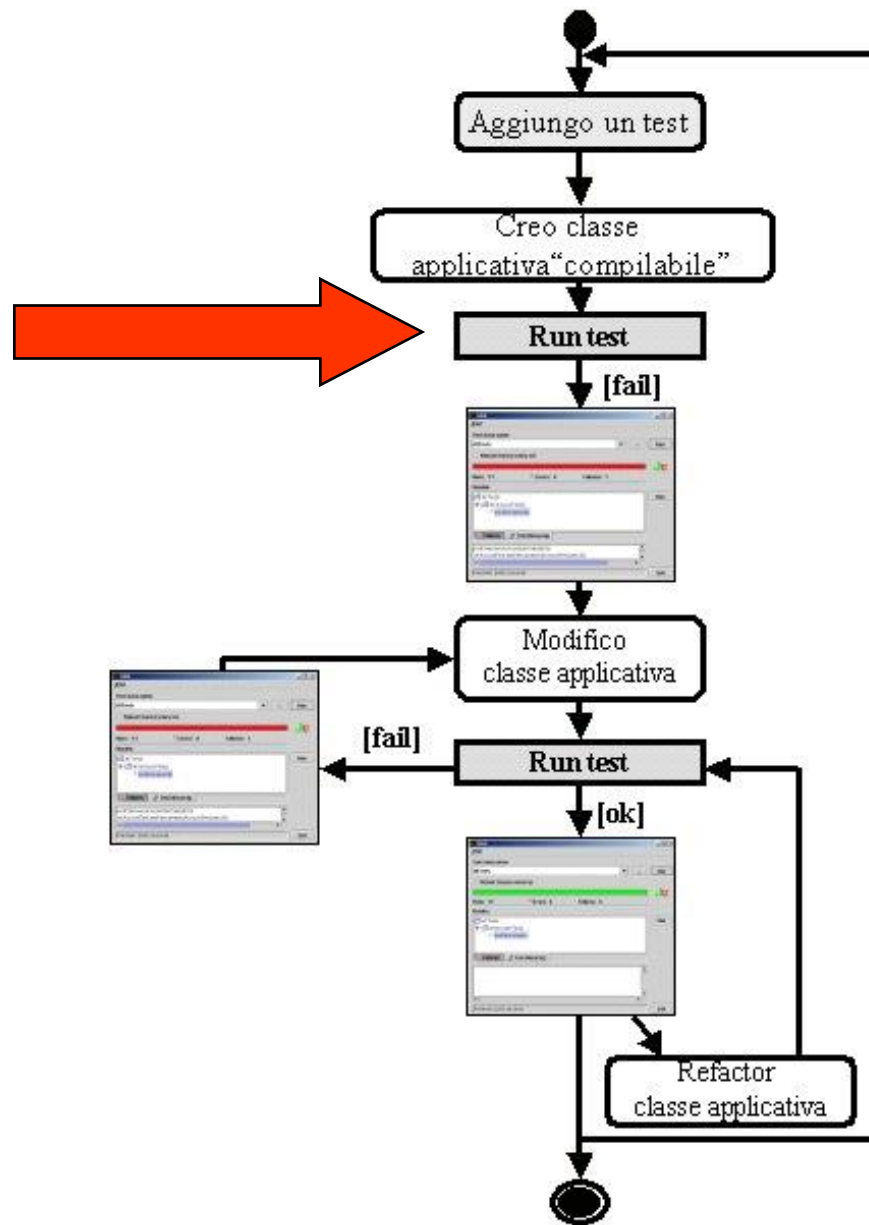
```

public class Test_account {

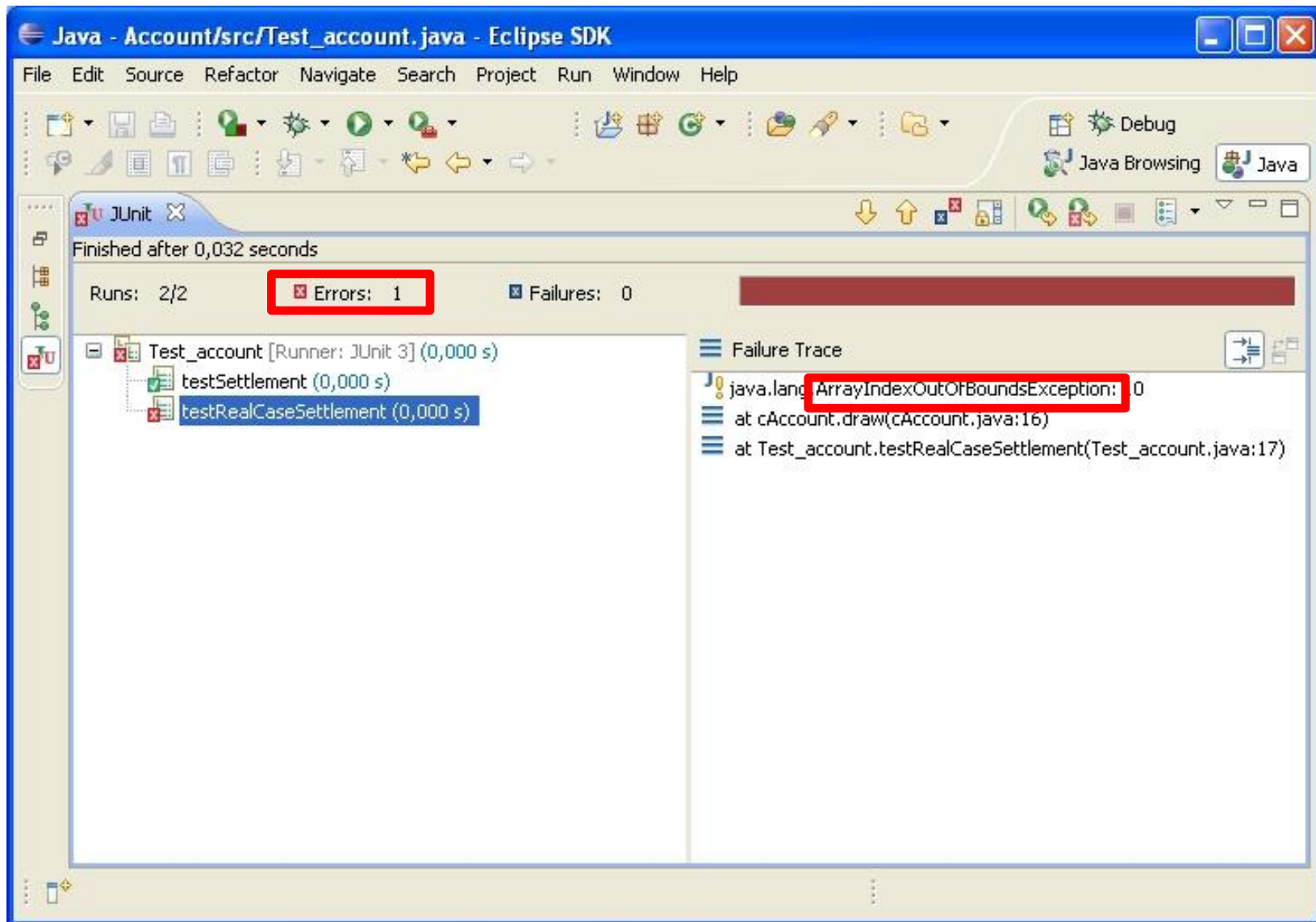
    @Test
    public void testSettlement() {
        ....
    }

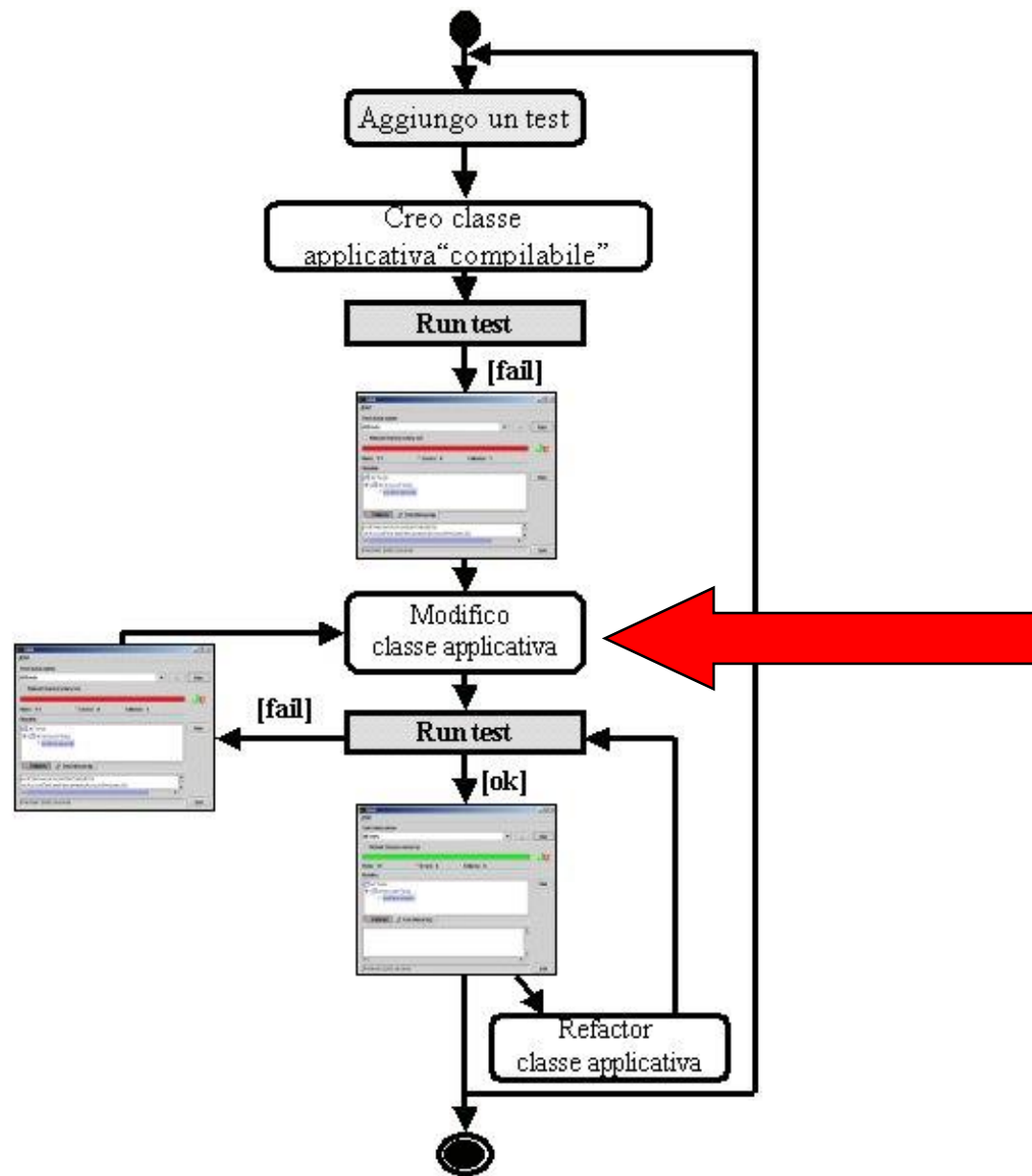
    @Test
    public void testRealCaseSettlement() {
        Account c = new Account();
        for (int i=0; i<10 ; i++)
            c.deposit(1);
        c.draw(-10);
        assertTrue(c.settlement() == 0);
    }
}

```



# RUN JUNIT (3)





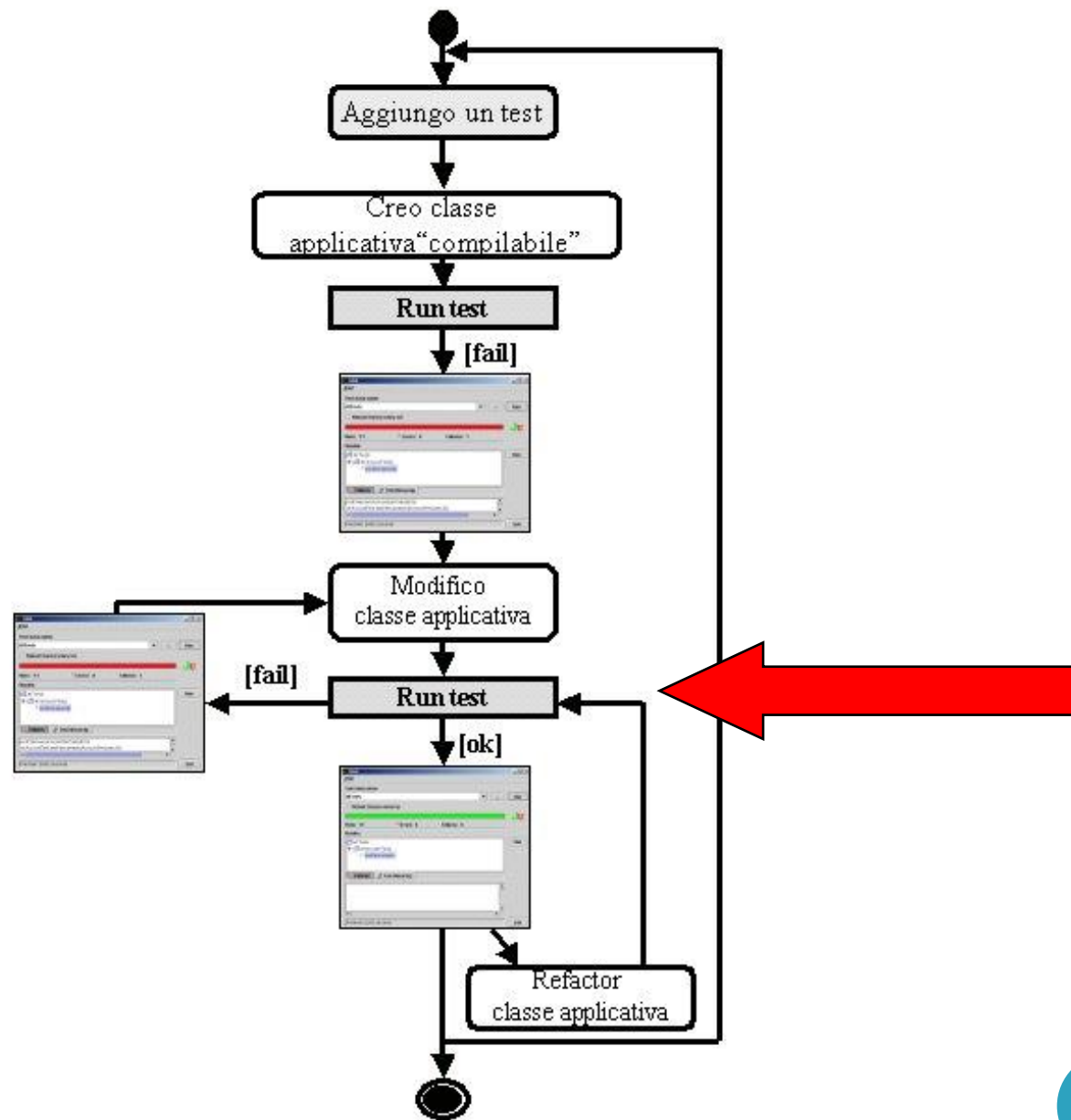
## CAMBIO ARRAY CON LISTA

```
public class Account {  
    List account = new LinkedList();  
  
    public void deposit(int value){  
        account.add(new Integer(value));  
    }  
  
    public void draw(int value){  
        account.add(new Integer(value));  
    }  
  
    public int settlement() {  
        int result = 0;  
        Iterator it = account.iterator();  
        while (it.hasNext()) {  
            Integer valueI = (Integer)it.next();  
            int val = valueI.intValue();  
            result = result + val;  
        }  
        return result;  
    }  
}
```

*Boxing*

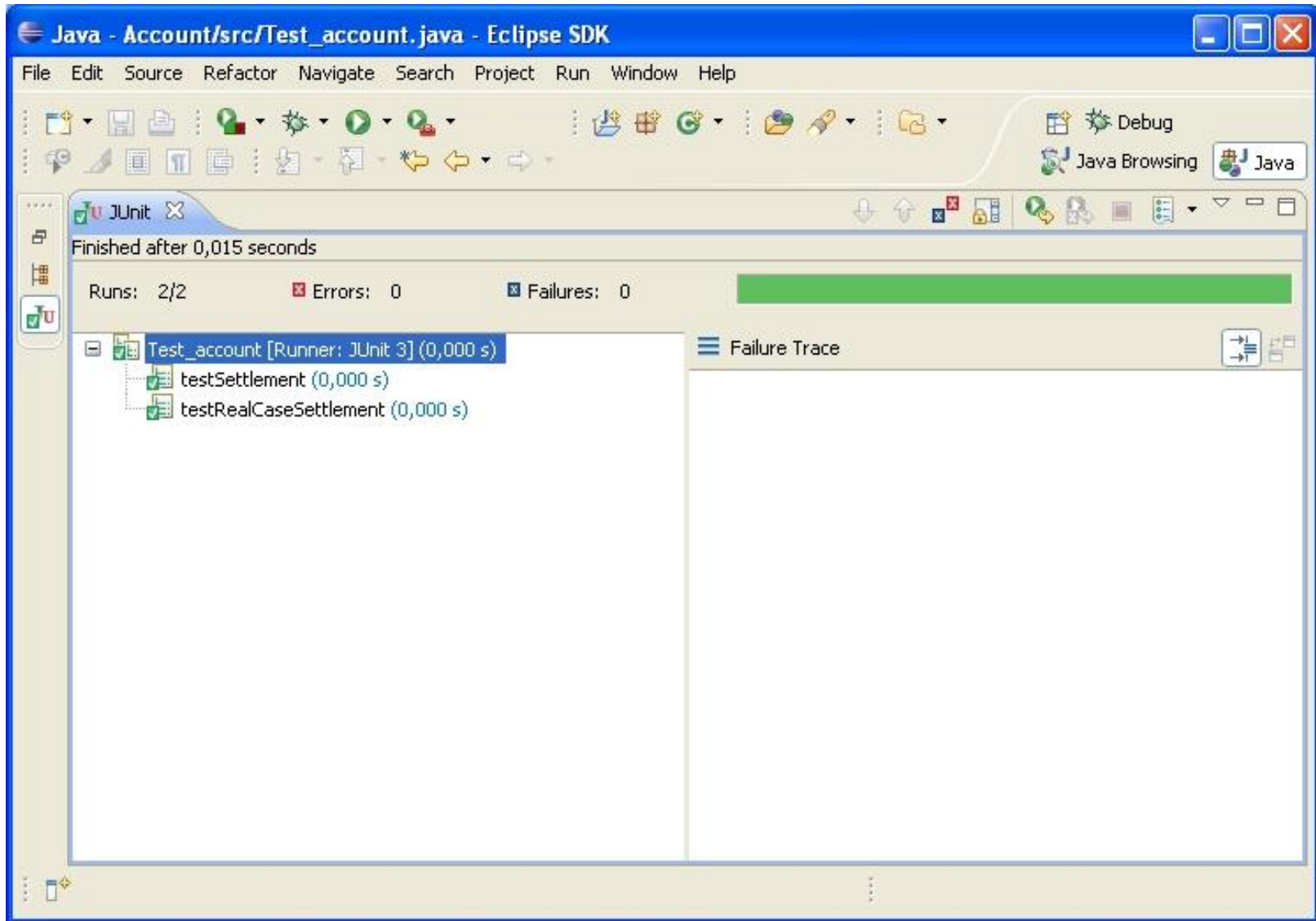
*Unboxing*

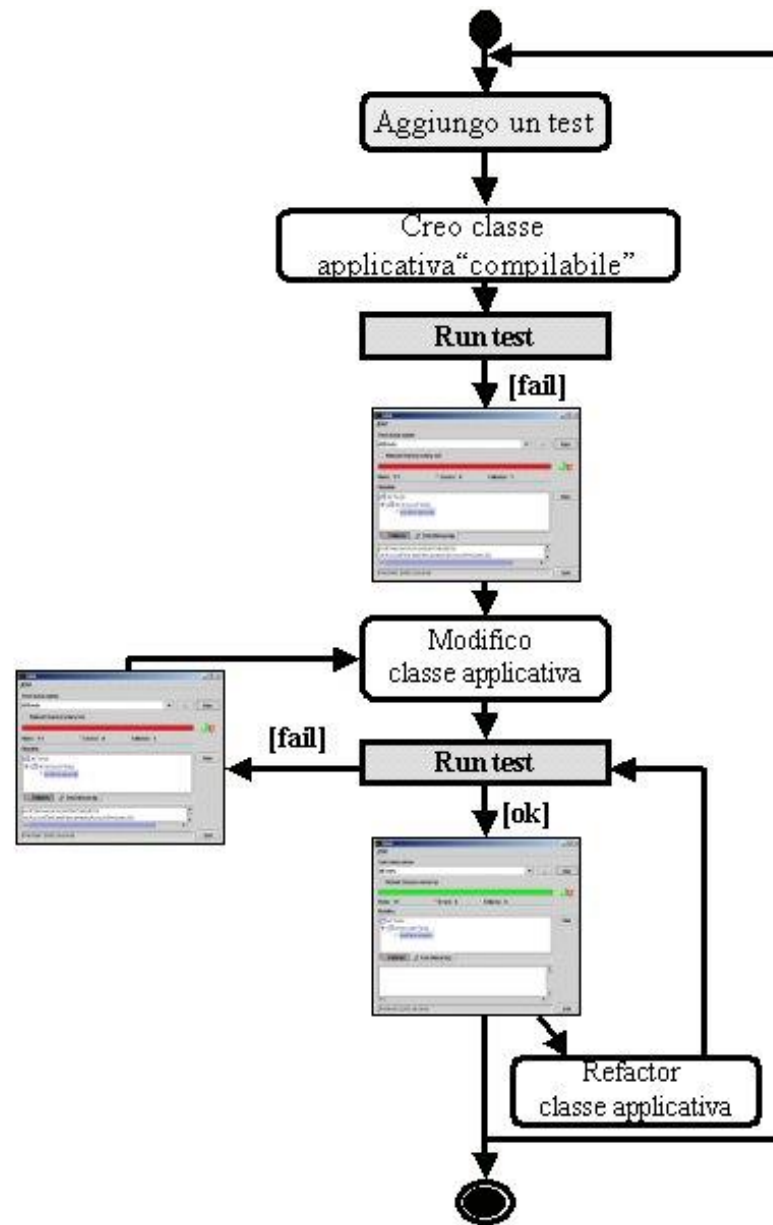
```
public class Test_account {  
    @Test  
    public void testSettlement() {  
        ....  
    }  
  
    @Test  
    public void testRealCaseSettlement() {  
        cAccount c = new cAccount();  
        for (int i=0; i < 10 ; i++)  
            c.deposit(1);  
        c.draw(-10);  
        assertTrue(c.settlement() == 0);  
    }  
}
```





# RUN JUNIT (4)






*"I can refactoring the program without anxiety. I have the testcases ..."*

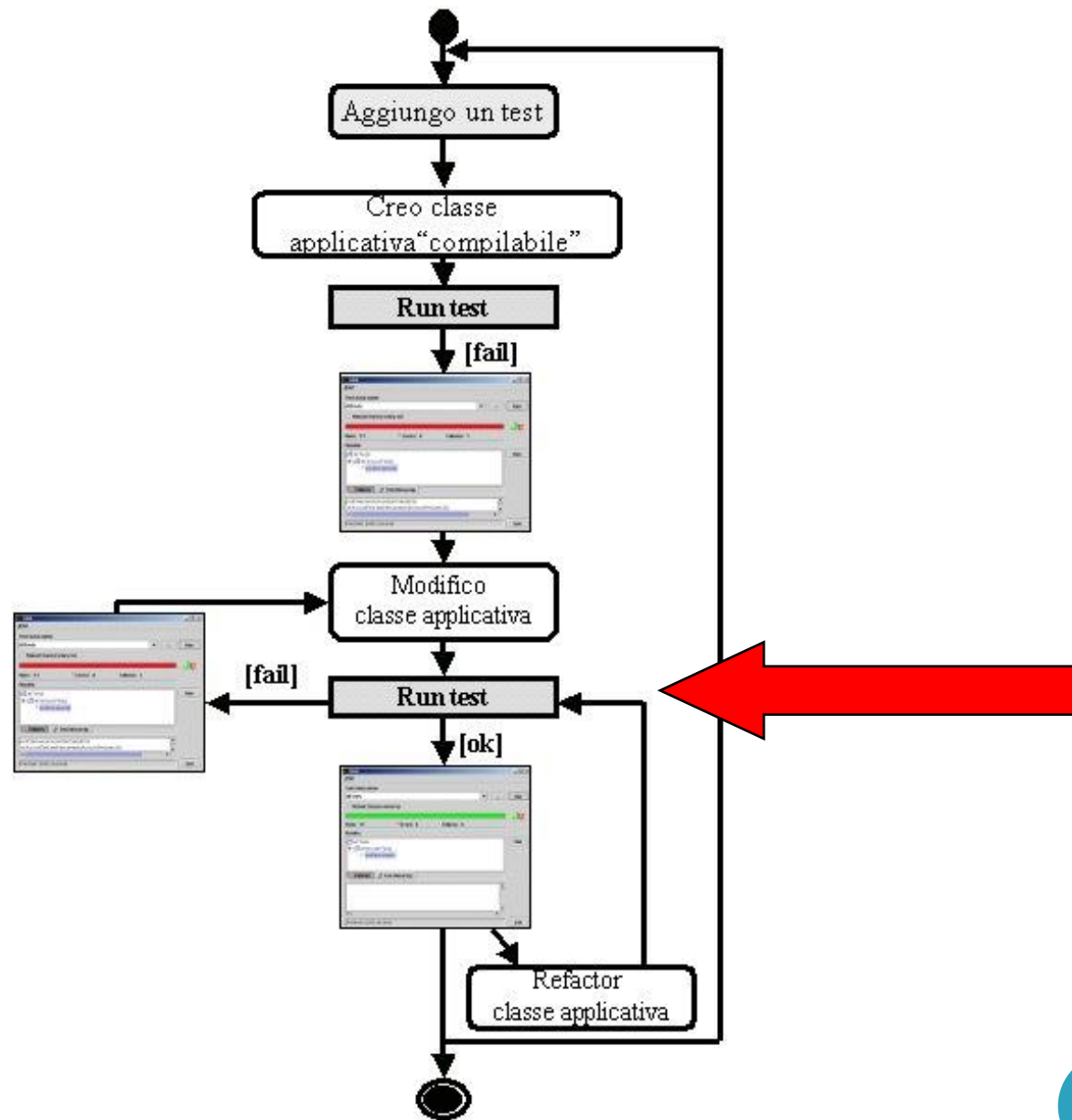
## REFACTOR: TIPI GENERICI

```
public class Account {  
    List<Integer> account= new  
        LinkedList<Integer>();  
  
    public void deposit(int value){  
        account.add(value);  
    }  
  
    public void draw(int value){  
        account.add(value);  
    }  
  
    public int settlement() {  
        int result = 0;  
        for(int val: account) {  
            result = result + val;  
        }  
        return result;  
    }  
}
```

*Costrutto  
For each*



```
public class Test_account {  
    @Test  
    public void testSettlement() {  
        .....  
    }  
  
    @Test  
    public void testRealCaseSettlement() {  
        cAccount c = new cAccount();  
        for (int i=0; i <10 ; i++)  
            c.deposit(1);  
        c.draw(-10);  
        assertTrue(c.settlement() == 0);  
    }  
}
```



# RUN JUNIT (5)

