7pixel

[redacted]

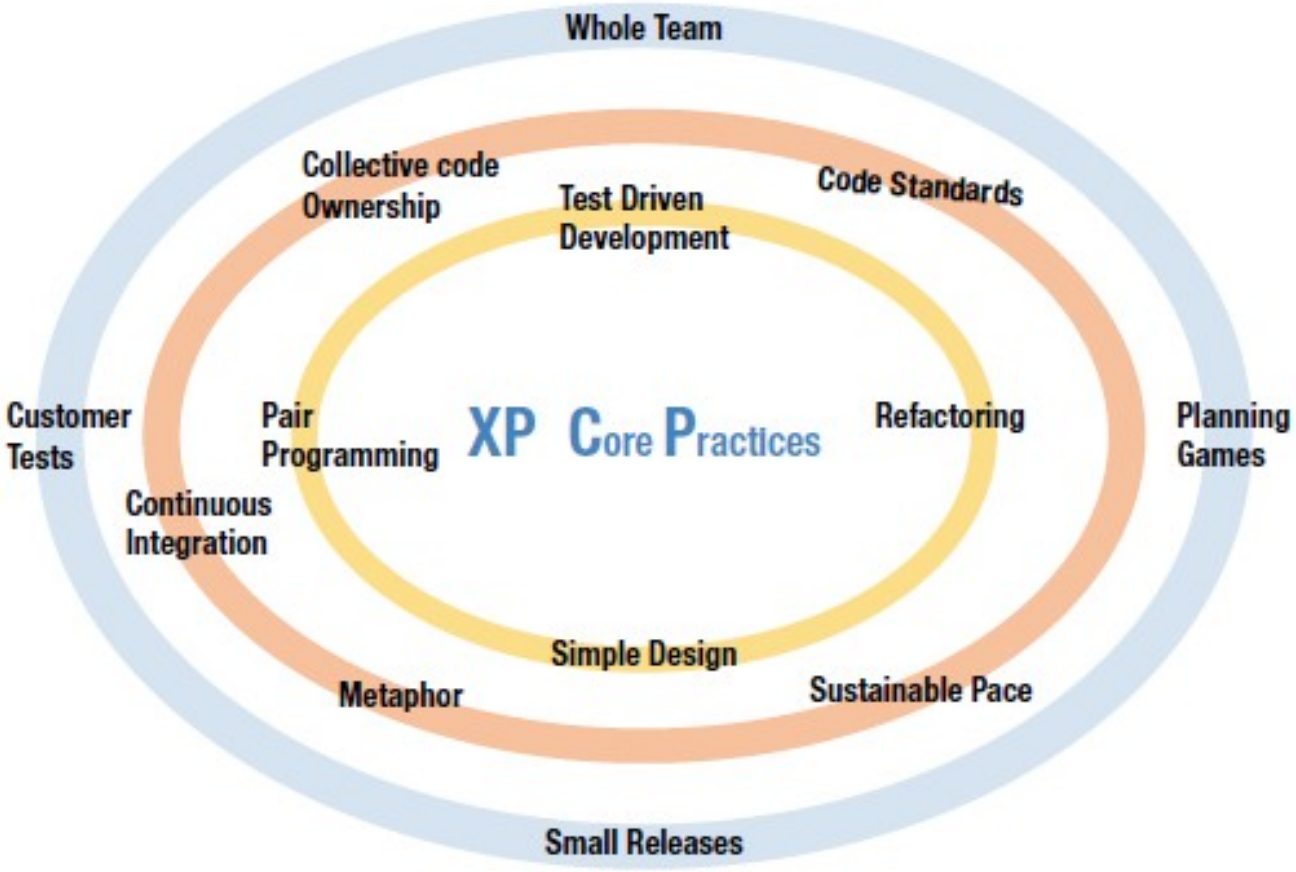
1- [redacted]

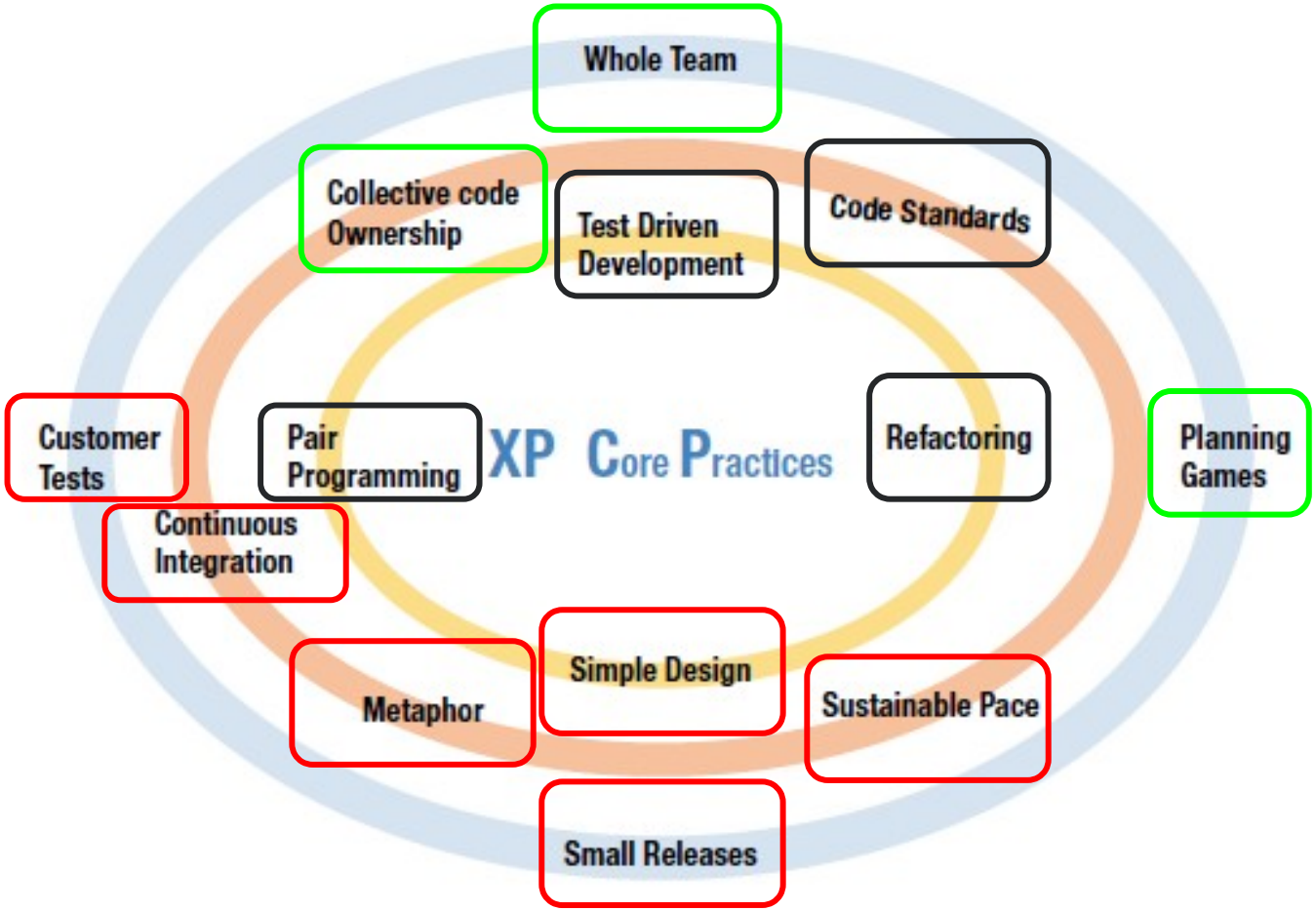
[redacted]. [redacted] [redacted] (7[redacted])

1.

□□□□□

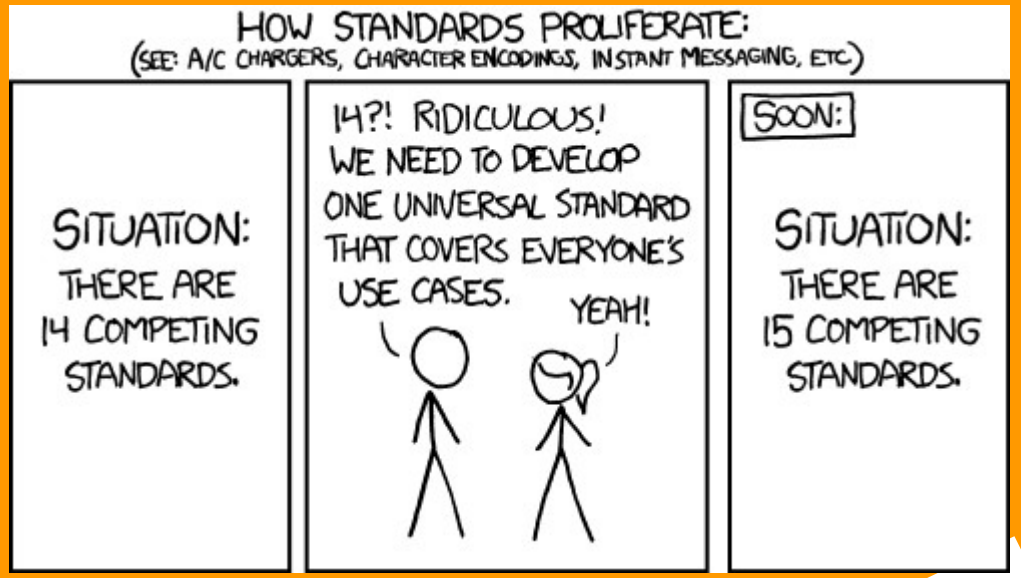
□□ □□□□□□□□□ □□□□□


Whole Team
Collective code Ownership
Test Driven Development
Code Standards
Customer Tests
Pair Programming
XP Core Practices
Refactoring
Planning Games
Continuous Integration
Simple Design
Metaphor
Sustainable Pace
Small Releases


Whole Team
Collective code Ownership
Test Driven Development
Code Standards
Customer Tests
Pair Programming
XP Core Practices
Refactoring
Planning Games
Continuous Integration
Metaphor
Simple Design
Sustainable Pace
Small Releases

2.

[illegible]


HOW STANDARDS PROLIFERATE:
(SEE: A/C CHARGERS, CHARACTER ENCODINGS, INSTANT MESSAGING, ETC)
SITUATION: THERE ARE 14 COMPETING STANDARDS.
14?! RIDICULOUS! WE NEED TO DEVELOP ONE UNIVERSAL STANDARD THAT COVERS EVERYONE'S USE CASES.
YEAH!
SOON:
SITUATION: THERE ARE 15 COMPETING STANDARDS.

[illegible]

- Sono approvati dal Team di Sviluppo, non imposti.
  Supportano anche la pratica CoCoOw.
- Appoggiarsi a standard esistenti (integrazione IDE)
- Esempi:
  - Uso di Javadoc (o analogo)
  - Usare standard affermati: Google, Oracle (Java), PEP (Python), Google, CKAN (Javascript)

□□□□□□ □□□□□□□□□ p□rso□□li

- Ogni team PUO’ darsi regole specifiche ,
- Esempi:
  - Uso del formato di variabili polacco (usato da Microsoft ad esempio): int64Hello, szInput..
  - Nessun metodo deve avere più di 16 righe
  - Non usare mai metodi statici (applicare Singleton)

[illegible] [illegible]?

Silicon Valley

Stagione 3 ep.6

2.

□□□□□□□

□□ □□□ □□ □□□□□?

Sappiamo (almeno in teoria) come funziona il debugging. E preferiamo evitarlo: una pratica lunga, noiosa, e non con risultati certi .

Come possiamo assicurare (o quantomeno migliorare) la qualità del nostro codice , specie in un contesto di "programming in the large"?

Semplice: facendo fare il lavoro sporco al computer.

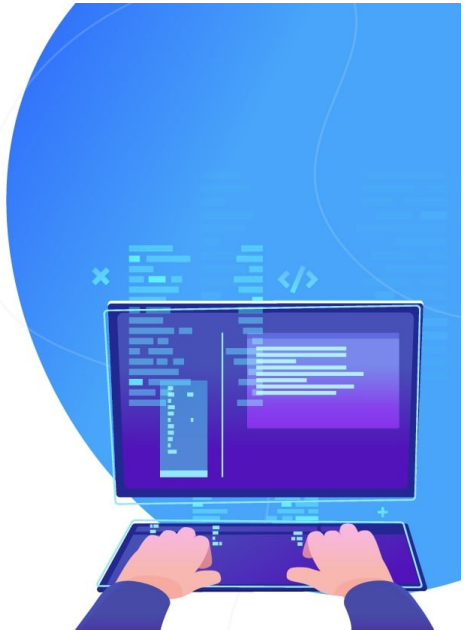
Vi presentiamo i test automatici.

□□□□□□□□□ □□□□□□□□□ □□□□□ □□□□□□...

□□□□□□□

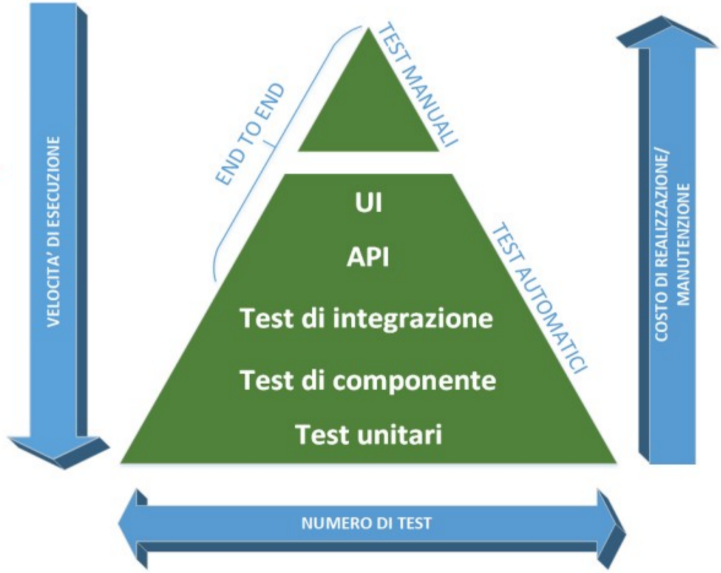
- I test sono l'UNICO modo che il programmatore ha per verifica se il codice scritto funziona oppure no.
- Tipicamente, il programmatore ingenuo lancia il programma 1-2 volte e, se tutto va bene, si considera soddisfatto.
- Questo però non è quasi mai sufficiente: occorre un approccio più metodico.

ALL CODE IS GUILTY UNTIL PROVEN INNOCENT



[illegible]

- Il 70% dei test dovrebbero essere test di unità
- I test manuali dovrebbero essere ridotti all'osso.
- Automatizzare il più possibile permette di risparmiare un sacco di tempo.



Data Analysis: Conclusions

The **data analysis** of customer satisfaction in relation (sales, profits) to product quality. The sales increase in relation to product satisfaction in relation to product quality and to

- the product quality
- the product price
- product sales sales increase in relation to product.

[illegible]: [illegible]

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

2.

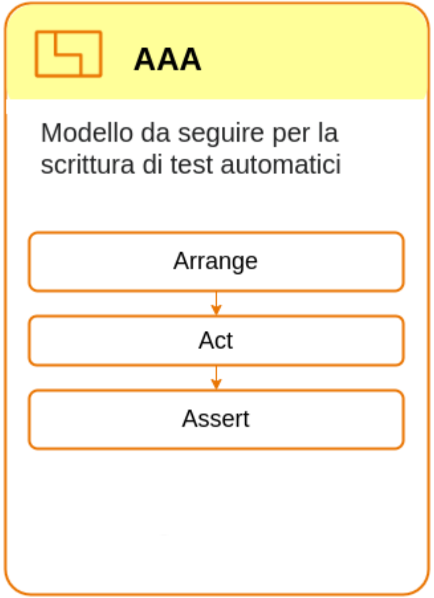
Com scri ere t st

[redacted]

[redacted] {

- [redacted]: [redacted]?
- [redacted] [redacted] [redacted] [redacted] [redacted]
- [redacted] [redacted]
- [redacted] [redacted] [redacted], [redacted]

} [redacted] ([redacted] [redacted] [redacted] [redacted]);


AAA
Modello da seguire per la scrittura di test automatici
Arrange
Act
Assert

[illegible]?

while (non si trovano più bug);

In realtà questa condizione non è facile verificare . In genere si lavora finché si ha tempo e si è "ragionevolmente sicuri" di non trovare nuovi bug. Esiste tuttavia un indicatore "oggettivo", chiamato Code Coverage, che ci dà una misura di quanto siamo /siete stati bravi a testare il codice .

TESTING IS AN INFINITE PROCESS OF COMPARING THE INVISIBLE TO THE AMBIGUOUS IN ORDER TO AVOID THE UNTHINKABLE HAPPENING TO THE ANONYMOUS.

3.

□□□□ □□□□□□□□□□

E2E Tests

Integration Tests

Unit Tests

[illegible]

[illegible], [illegible] - [illegible] - [illegible]..

[illegible]

[illegible] [illegible] [illegible] [

□□□□ + □□□

Tutti gli IDE permettono di utilizzare Junit , spesso senza aggiunte di software particolari.Eclipse, Netbeans, IDEA, ad esempio, sono pre-configurati per lavorare con JUnit quasi “out-of-the-box”.

[illegible]

Situazione è molto simile alla precedente : la libreria [illegible] è di fatto considerata lo standard , ed è largamente utilizzata dai professionisti - anche se esistono librerie alternative . Per gli IDE vale il discorso precedente

□□□□□□□□□□

In Javascript invece non esiste un unico vincitore : ne esistono almeno 5, ciascuno con punti di forza e debolezza. Richiedono tutti un minimo di configurazione, e l ’integrazione con l’IDE non va sempre liscia. Citiamo:

- Jest
- Puppetter
- Mocha
- Jasmine
- QUnit

C/C++

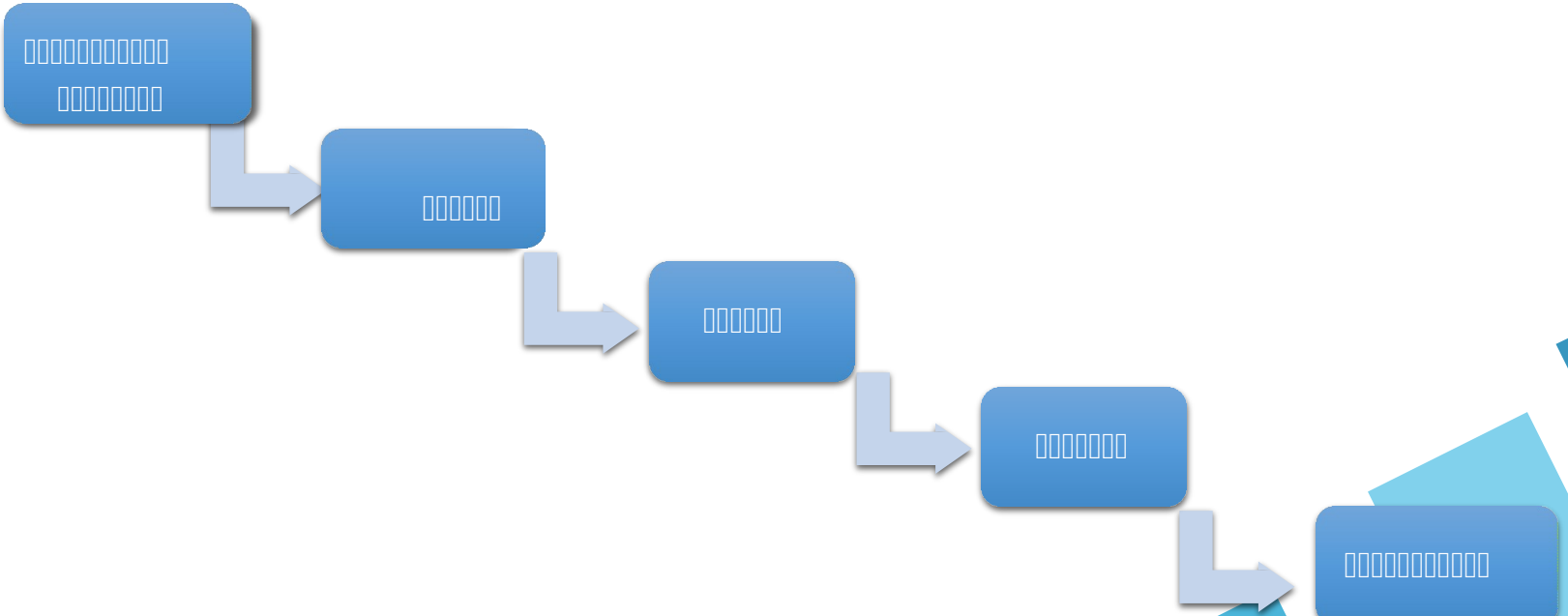
Для языка C/C++ в стандартной библиотеке есть функции, которые позволяют работать с файлами. Для работы с файлами нужно подключить заголовочный файл **stdio.h**. Для открытия файла используется функция **fopen**, для закрытия **fclose**. Для чтения данных из файла используется **fscanf** (работает аналогично **scanf**) и для записи **fprintf** .

4.

□□□ - □□□□ & □□□?

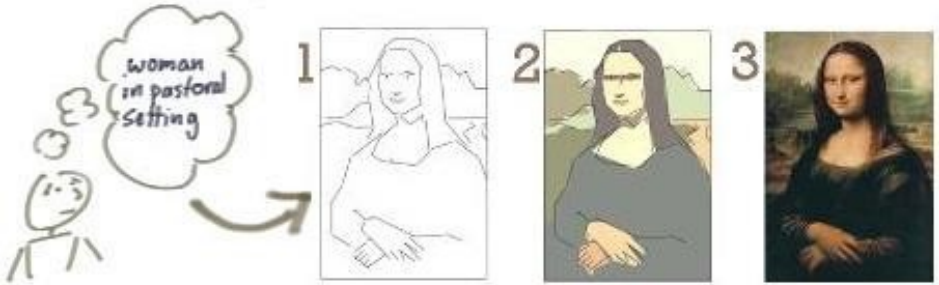
Reversed perspective

□□ □□□□□□□□□□□ – □□□□□□□□□ □□□□□□□□□ □□□□□



Incremental

Iterative



[illegible]:

- [illegible]
- [illegible]?
- [illegible]?

[illegible]

- [illegible]
- 90 [illegible]
- [illegible]
  [illegible].

Perché fare TDD (possibili risposte)

- Design evolutivo del codice
- Focus su un unico aspetto utile
- Sicurezza sul codice di produzione
- Documentazione sempre aggiornata
- Esprimere creatività
- Dare un ritmo allo sviluppo

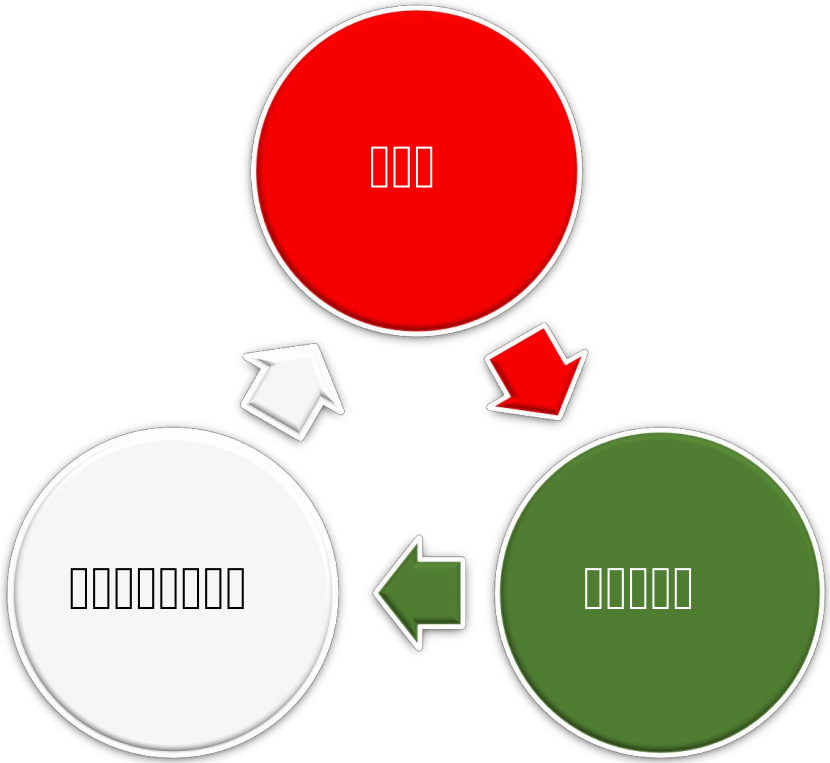
[illegible]

- [illegible]
- [illegible]
- [illegible]
- [illegible]
- [illegible]

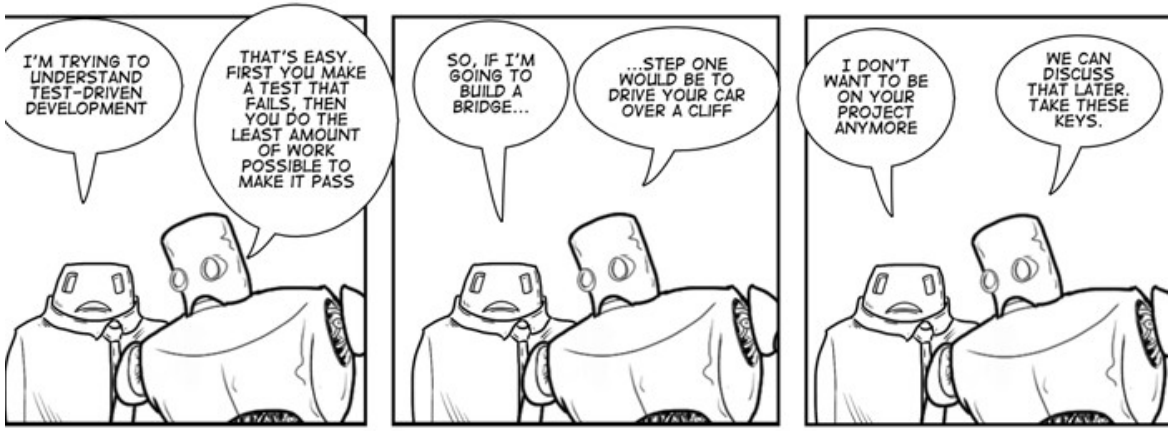
5.

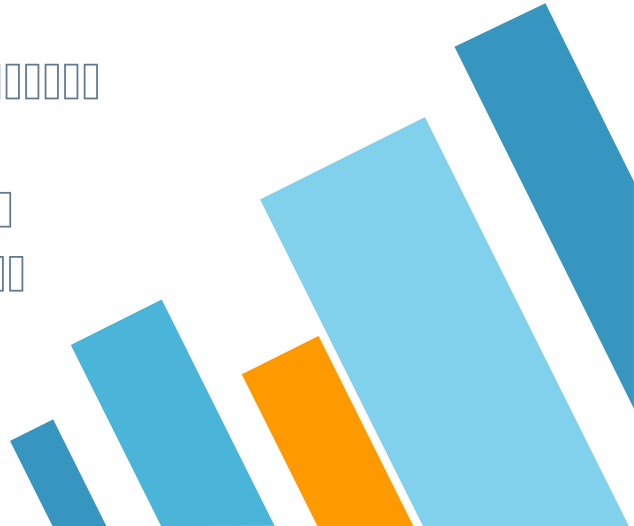
□□□ - □□□?

[illegible]



[illegible]	[illegible]
[illegible]	[illegible]
[illegible]	[illegible]


I'M TRYING TO UNDERSTAND TEST-DRIVEN DEVELOPMENT
THAT'S EASY. FIRST YOU MAKE A TEST THAT FAILS, THEN YOU DO THE LEAST AMOUNT OF WORK POSSIBLE TO MAKE IT PASS
SO, IF I'M GOING TO BUILD A BRIDGE...
...STEP ONE WOULD BE TO DRIVE YOUR CAR OVER A CLIFF
I DON'T WANT TO BE ON YOUR PROJECT ANYMORE
WE CAN DISCUSS THAT LATER. TAKE THESE KEYS.

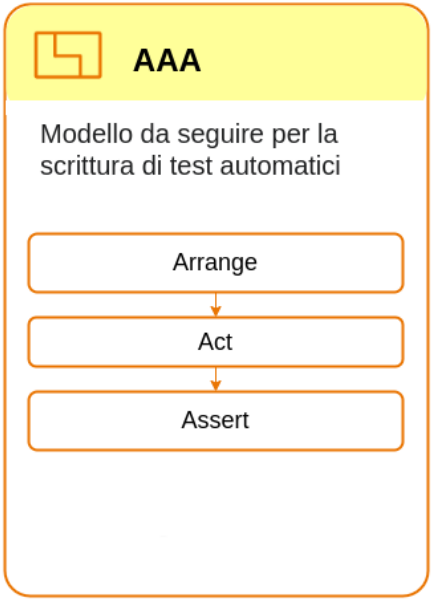


□□ 3 □□□□□□ □□□ □□□

Se voglio fare TDD devo rispettare le seguenti regole

- Non scrivere codice di produzione prima di aver scritto un test rosso
- Non scrivere più di un test che fallisce , un errore di compilazione è come un fallimento
- Non scrivere più codice di produzione di quello necessario per far passare il test correntemente rosso

[illegible]


AAA
Modello da seguire per la scrittura di test automatici
Arrange
Act
Assert

[illegible]

Voglio testare le funzionalità base di una classe conto bancario (Java + JUnit)

```
BankAccount account = new BankAccount().WithBalance(10);
account.addMoney(10);
a[illegible]sertEquals(20, account.getBalanc[illegible]());
```

Arrange →

Act →

Assert →

□on □□□□□ tes□ □art□re?

- Happy Path: il caso più semplice /favorevole che porta maggior valore
- Sad Path: casi con input errato .
- ..
- Boundaries Condition: i casi limite

[illegible] - [illegible]

[illegible] ([illegible] + [illegible])

```
[illegible] [illegible] = [illegible] [illegible]().[illegible](0);
[illegible].[illegible](0);
[illegible]("[illegible] 0", [illegible].[illegible](), [illegible](0));
[illegible].[illegible]([illegible]);
[illegible]([illegible].[illegible]());
```

"[illegible]"

[illegible]: [illegible]

[illegible]

- [illegible] [illegible] [illegible]

[illegible] [illegible] [illegible]

- [illegible] [illegible] [illegible] [illegible] [illegible] [illegible]
- [illegible] [illegible] [illegible] (!!!)
- [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] (e.g. [illegible] [illegible] [illegible] [illegible] a [illegible])
- [illegible] [illegible] (e.g. [illegible] [illegible] [illegible] [illegible] [illegible] [illegible] a [illegible] [illegible]'[illegible])
- [illegible] [illegible] [illegible] [illegible]

[illegible]

- Da dove parto a fare refactoring? Dai Code Smell!

- "Insieme di caratteristiche che il codice sorgente può avere e che sono generalmente riconosciute come probabili indicazioni di un difetto di programmazione"
- Non sono bug! (il software funziona correttamente)
- Sono "debolezze" del codice che ne riducono la qualità

Il sito "definitivo" di riferimento:
https://refactoring.guru/refactoring/smells

[illegible]

- [illegible]
- [illegible] (?, ?)
- [illegible]
- [illegible]

[illegible] [illegible] [illegible] [illegible]?

- [illegible] (и [illegible] [illegible] [illegible] [illegible]! [illegible]: [illegible]?)
- [illegible]
- [illegible] (как [illegible] [illegible], [illegible] и [illegible]!)
- [illegible] (или [illegible] и [illegible])
  - [illegible]
  - [illegible] и [illegible] [illegible] [illegible] [illegible] (или и [illegible] [illegible])!
  - и [illegible] [illegible] [illegible] и [illegible] и [illegible]

□□□□ d□v□ebbe □□□□□□ un □□□□? (2)

- Rifattorizzato (testare le primitive di test)
- Non distruttivo
  - Preserva le condizioni iniziali (No Drop table)
- Self verified
  - E' chiaro perché fallisce ? Se fallisce devo capire al volo come mai
  - Piuttosto metto un messaggio nelle asserzioni
- Timely (eseguito in tempo ragionevole)

Chi testa il test ?

60 secondi per pensarci a coppie....

Il QA?

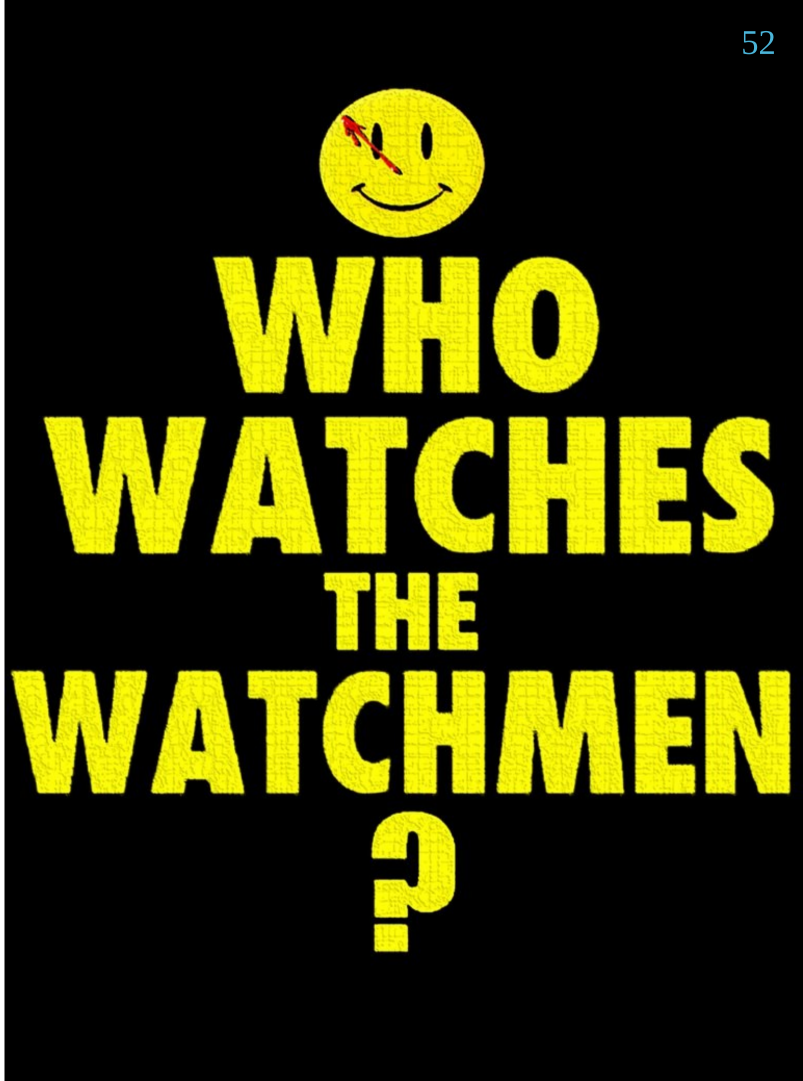
Il PO?

Lo SM?

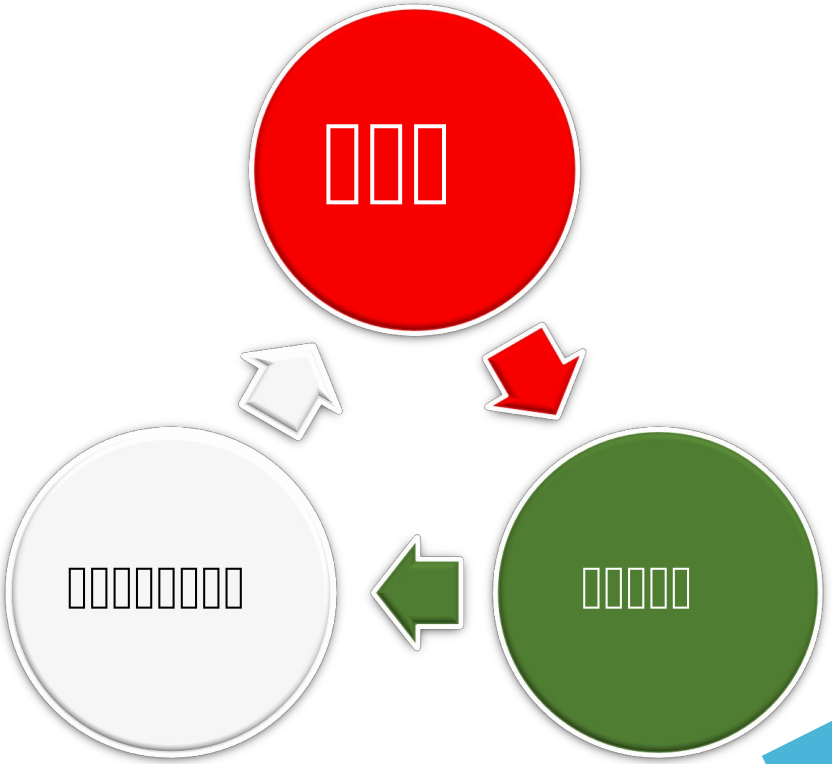
Il mio compagno di pair?

Un tester del test ?

Un test automatico del test?



□□□□ □□□ □□□□:
□□□ □□□□□□□□ □□
□□□□!



6.

□□□□ □□□□□□□□□□□

□□□□□ & □□□□

□□ □□□□□ □□□□□ □□□□□□□□

- Come possiamo introdurre il concetto di ritmo nella giornata dello sviluppatore?
- Possiamo usare uno strumento che ci permetta anche di valutare il tempo speso?
- Quale unità di misura scegliere ?

□□□□□□□□

{

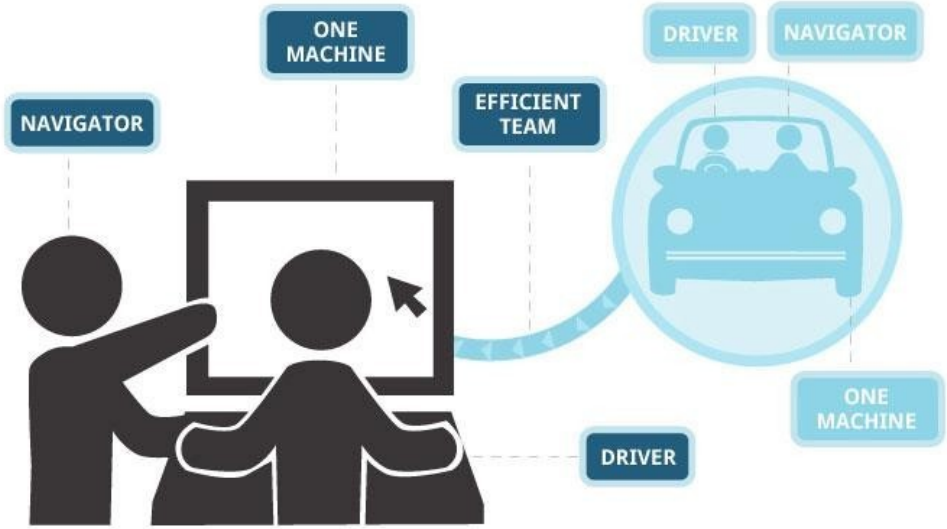
“name”: “tecnica del pomdoro”,

“y□□tube”: htt□□://□□w.y□□tube.com/watc□?v=CT70iCa□0Gs,

“website”: “http://pomodorotechnique.com/”

}

[illegible]


ONE MACHINE
DRIVER
NAVIGATOR
NAVIGATOR
EFFICIENT TEAM
ONE MACHINE
DRIVER

[illegible]

I programmatori lavorano sempre in coppia, alternandosi tra i ruoli di Driver e Navigator.

DRIVER

- Ha mouse e tastiera
- Scrive i test ed il codice
- Ha una visione di breve periodo

NAVIGATOR

- Ha una todo-list
- Appunta i test mancanti
- Appunta refactoring possibili
- Ha una visione di lungo periodo

□□□□□ □ □□□□□□□□

1. □lass□ca: Driver scrive codice, Navigator commenta controlla ad alto livello
2. □ard: Navigator detta le idee, Driver esegue - *“□□□ □□ □□□□ □□ □□ □□□□ □□□□ □□□□ □□□□ □□□ □□□□□□□□ □□ □□□□ □□ □□□□□□□ □□□□□□□ □□□□□□ □□□□□”*
3. Tour : Driver fa tutte le scelte raccontando quel che fa , Navigator segue il ragionamento e controlla
4. □ob : un intero gruppo partecipa come Navigator, un solo driver

Pair Programming: annedottica

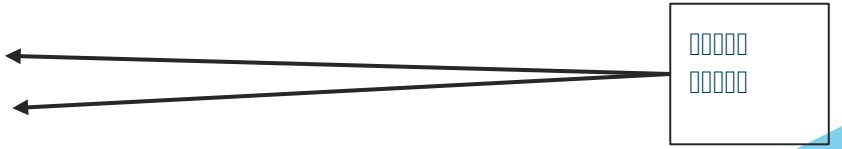
	Progetto 1: solisti	Progetto 2: coppie
Dimensione (KLOC)	20	520
Team	4	12
Sforzo (mesi persona)	4	72
Produttività (KLOC/mp)	5	7.2
Produttività (KLOC/mpair)	n.d.	14.4
Difetti test unità	107 (5.34 difetti /KLOC)	183 (0.4 difetti /KLOC)
Difetti test integrazione	46 (2.3 difetti /KLOC)	82 (0.2 difetti /KLOC)

[redacted] [redacted] - [redacted]

[redacted]: [redacted] [redacted] .

[redacted] [redacted] -[redacted] "[redacted] [redacted] [redacted].
>
>
[redacted]-[redacted]-[redacted]: [redacted] <[redacted].[redacted]>
[redacted]-[redacted]-[redacted]: [redacted].[redacted] <
[redacted].[redacted].[redacted]>"

[redacted] [redacted] [redacted] [redacted], [redacted] [redacted]
[redacted] [redacted]



[illegible] [illegible] [illegible]

- [illegible]
- [illegible] («[illegible]?»)
- [illegible] [illegible] [illegible]
- [illegible]

The dark side of pair programming.

Pair Programming - Note

A) Non è detto che le coppie siano fisse. Anzi, in molti contesti si obbliga una turnazione stretta in modo da amalgamare gli stili programmazione ed eliminare la "Silo culture".

B) Anche il miglior pilota ha bisogno di riposo e anche il miglior navigatore vuole adrenalina. Morale: cambiare ogni 2-3 pomi, seguire user story diverse, affrontare le cose che si conoscono meno

7.

TDD □al i s□heni□s

□□□□□

- Linguaggio di programmazione: Java
- Un sistema di versionamento: Git
- Un ambiente di sviluppo: come IDE di riferimento utilizzeremo IntelliJ IDEA , senza supporto di sistemi di building come Maven, Gradle e altro.
- Un Test Framework: Ci concentreremo su Junit 5 (Jupiter). Questa versione permette anche di eseguire test scritti nelle versioni precedenti (JUnit Vintage).

□□□□

Vogliamo costruire una calcolatrice in grado di sommare i numeri presenti una stringa di testo

- La stringa può avere più numeri separati dalla virgola
- Se non ho numeri voglio ottenere 0
- ….ho dimenticato qualcosa?

LINK: https://osherove.com/tdd-kata-1

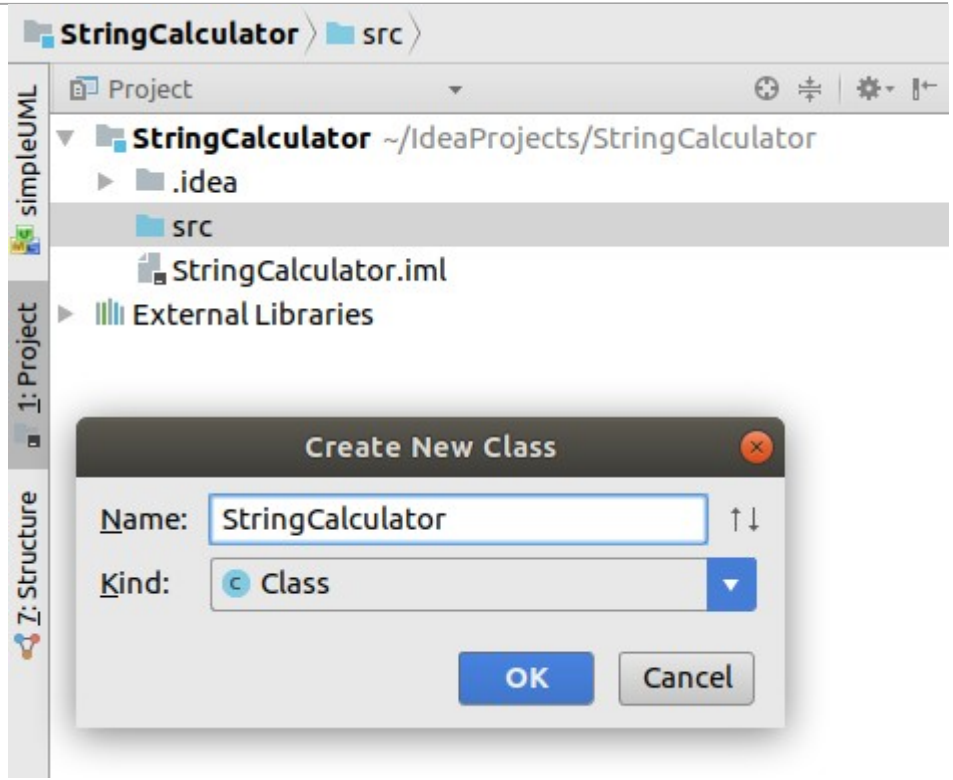
[illegible] & [illegible]

[illegible] & [illegible]

[illegible] [illegible] & [illegible]

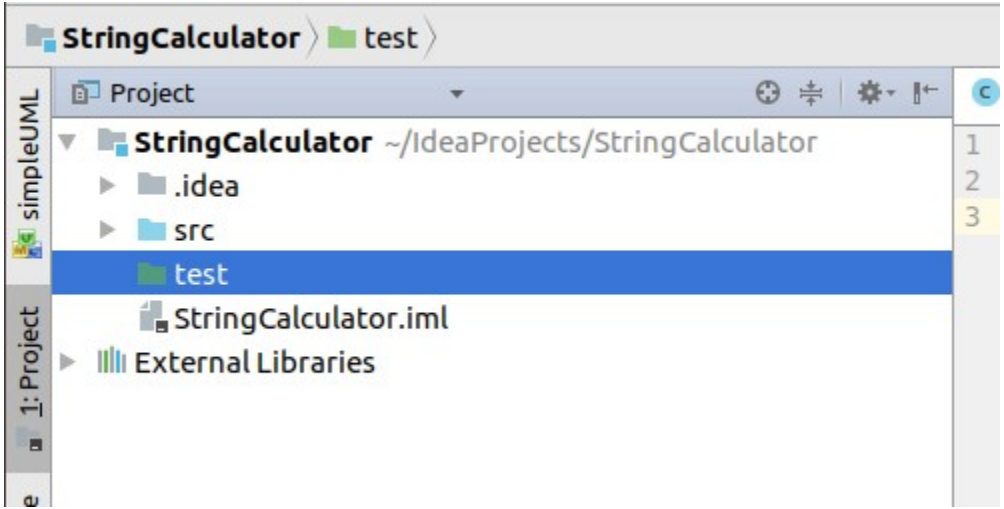
□□□□□□□□

- □□□□□□ □□ □□□□□□□ □□□□□□□□ □□□□.
- □□□□□□ □□ □□□□□□ □□□□□□□□□□□□□□□□□


StringCalculator
src
Project
StringCalculator ~/IdeaProjects/StringCalculator
.idea
src
StringCalculator.iml
External Libraries
simpleUML
1: Project
7: Structure
Create New Class
Name:
StringCalculator
Kind:
Class
OK
Cancel

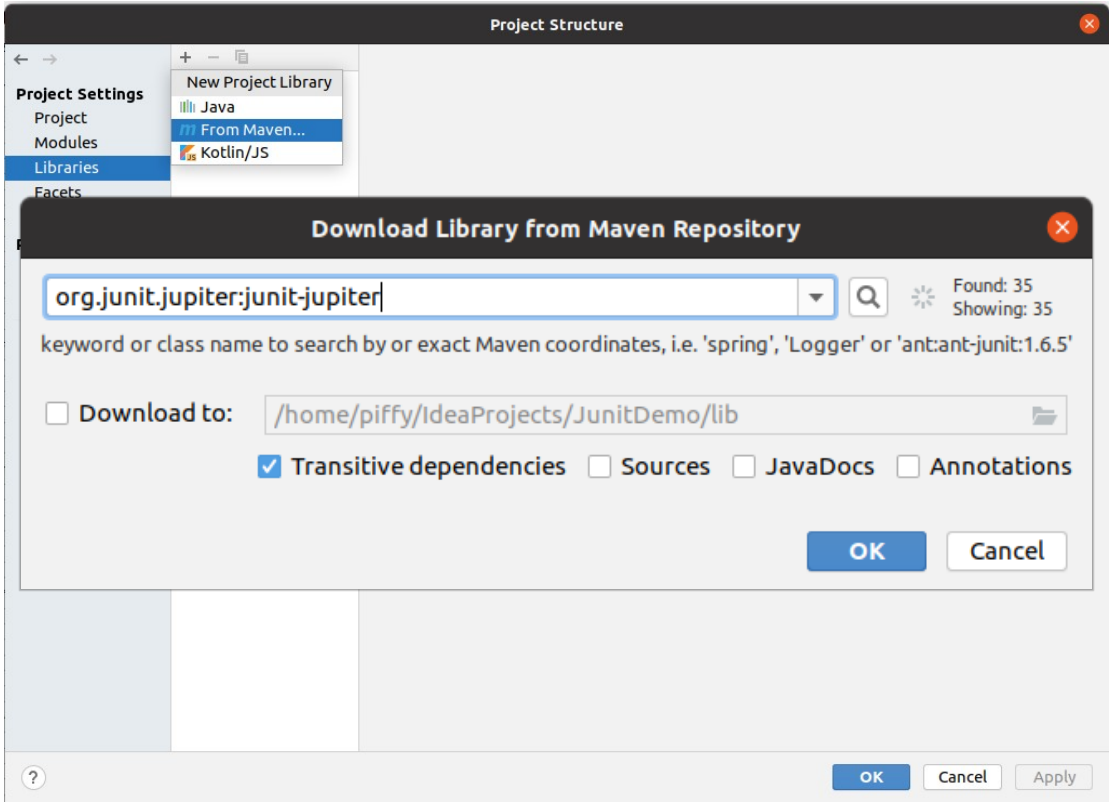
□□□□□□□□

- □□□□□□ □□□ □□□□□□□□ □□□□□□□□ □□□□
- □□□□□ □□□□□□, □□□□ □□□□□□□□□ □□ > □□□□ □□□□□□ □□□□
- □□□□□□□□ □□□□□□□□□ □□□□□□□


StringCalculator
test
Project
StringCalculator ~/IdeaProjects/StringCalculator
.idea
src
test
StringCalculator.iml
External Libraries
simpleUML
1: Project

□□□□□□□□

- □□□□ □□□-□□□□□ □□□ □□□□ □□□□□ □□□□□□ □ □□□□□□□□□ “□□□□□□ □□□□”
- □□□□□□□□ □□□.□□□□□.□□□□□□□:□□□□□-□□□□□□□ □ □□□□□□□ □□□□□□□. □□□□□□□□□□□ □□□□□□ □□□ □□□□□□□□ □□□ □□□□-□□□□ □□□□ (□□□□ 5.8.1)


Project Structure
Project Settings
Project
Modules
Libraries
Facets
New Project Library
Java
From Maven...
Kotlin/JS
Download Library from Maven Repository
org.junit.jupiter:junit-jupiter
Found: 35
Showing: 35
keyword or class name to search by or exact Maven coordinates, i.e. 'spring', 'Logger' or 'ant:ant-junit:1.6.5'
Download to:
/home/piffy/IdeaProjects/JunitDemo/lib
Transitive dependencies
Sources
JavaDocs
Annotations
OK
Cancel
OK
Cancel
Apply

□□□□□□□□

□□□□□-□□□□□ □□□ □□□□ □ □□□□ “□□□ □□□□□□□□□□□□□□□□□□□□” (□□□□□□ □□□□□□ □□□ □□□□□□□□□□□□□□□).

Str□□tura base di □□ □□□□ □□□□ □□NIT

```
@Test
    public void nomeDelTest() {

        -> SETUP □ELLE VARIABILI (A...)
      -> IN□OCAZ□ONE D□L METODO/FUN□IONE (A...)
        -> UNA O P□U' ISTRUZIONI □□ CONTROLLO (A...)

    }
```

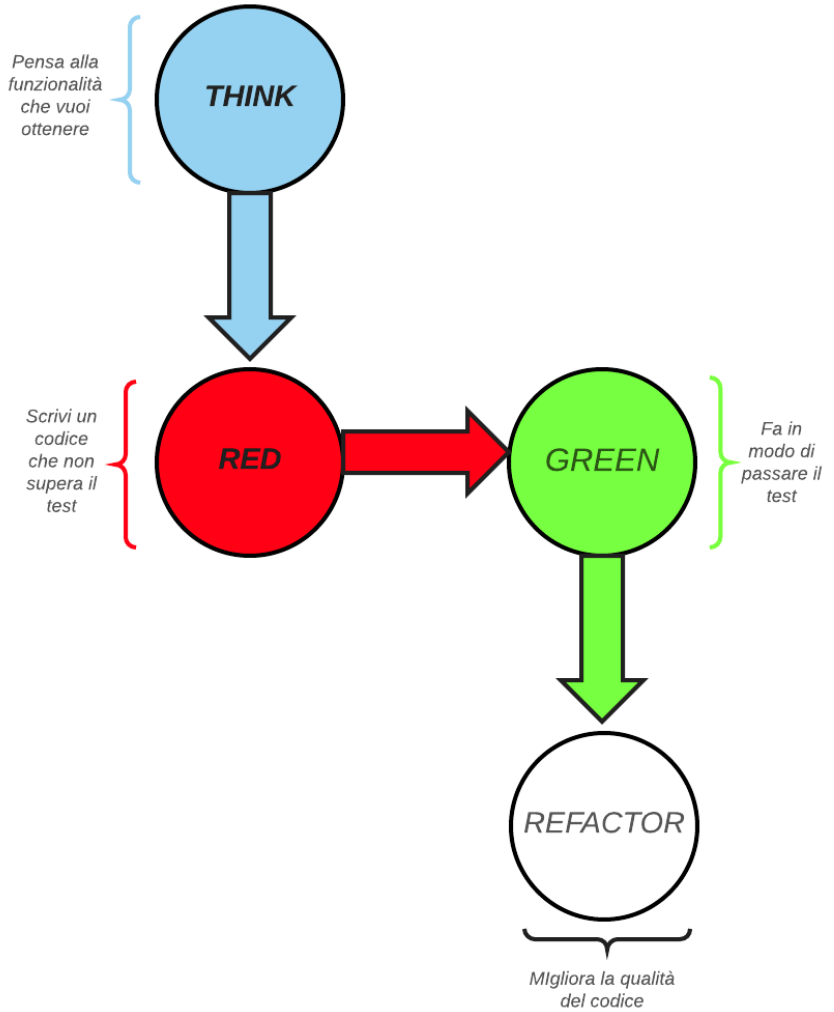
```
@Test
        @DisplayName("Should be false") //OPTIONAL
    public void testToString() {  //Cambiare con il nome dellc classe che avete creato
        String tester = new MiaClasse().toString();

        // assert statements
        assertEquals(true, tester.equals("Oggetto"), "Deve restituire la parola Oggetto");
    }
```

[illegible] [illegible] [illegible] [illegible] a [illegible] [illegible] di [illegible] ([illegible], [illegible], [illegible] [illegible]) [illegible] [illegible] "[illegible]" di [illegible] - [illegible] [illegible] [illegible] a di [illegible] [illegible].

```
@Test
public void shouldBeFalse() {
        assertThat(false, is(false));
        }
```

□□□□□ □□□ □□□□□□
(□□□□□□□□)


Pensa alla funzionalità che vuoi ottenere
THINK
Scrivi un codice che non supera il test
RED
GREEN
Fa in modo di passare il test
REFACTOR
MIgliora la qualità del codice

□□□□□ □□□□ (□□□)

- Scrivere prima il metodo che avete intenzione di sviluppare
- Scrivere il test in modo che richiami il metodo in modo sintatticamente corretto corretto (es: int calcola(String s))
- Deve essere un test semplice ed elementare (Happy Path). Esempio: fornendo una stringa vuota, bisogna ottenere 0
- Scrivetelo e mostratemi il codice e che genera un Rosso

[illegible]

[illegible]

[illegible]

[illegible]

SCAMBIATE I RUOLI!

□□□□□□□□ □□□□ (□□□)

Il metodo deve essere modificato.

Scrivere un test che ESTENDA le funzionalità del metodo ;
le funzionalità precedenti devono essere mantenute.

Show me the (RED) code.

□□condo test (□□□□□)

Presumibilmente, test di due o più valori , stringa separata da un carattere di qualche tipo. Oppure semplicissimo: fornendo “1” il metodo deve restituire 1.

Sviluppate il codice

Deve essere verde

[illegible]

Migliorate il codice ...

Generalizzate (n numeri?)

Togliere rigidità (hardcoded values?)

Commenti (Javadoc?)

I test DEVONO RIMANERE VERDI

SCAMBIATE I RUOLI!

□□□□□ □□□□ (□□□)

Fornite robustezza al vostro codice! Gestite errori del programmatore e/o dell'utente

null

Empty string

...

Show me the code(s)

□□□□□□□□□□□

- Martin Fowler et al. (1999), Refactoring: Improving the Design of Existing Code, Addison-Wesley
- Robert C. Martin (2008), Clean Code: A Handbook of Agile Software Craftsmanship, Prentice Hall
- Kent Beck (2000), Extreme Programming Explained: Embrace Change, 2nd Edition, Addison-Wesley

[redacted] [redacted] ([redacted]-[redacted])

```
[redacted]
[redacted] [redacted] [redacted]() {
[redacted] [redacted] = [redacted] [redacted]();
[redacted] [redacted] = [redacted].[redacted]("1");
[redacted](1);}
[redacted]
[redacted] [redacted] [redacted]() {
[redacted] [redacted] = [redacted] [redacted]();
[redacted] [redacted] = [redacted].[redacted]("1,2");
[redacted](3);}
```

□□□□□□!

[redacted]

[redacted] :

- [redacted] [redacted] [redacted]
- [redacted] [redacted]
- *[redacted] &*
  [redacted]

- *[redacted] [redacted] [redacted] [redacted]*
  [redacted] 3.0