UNIVERSITÀ
DEGLI STUDI DI BARI
ALDO MORO

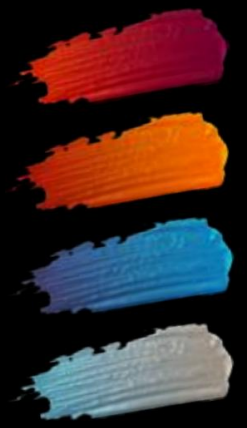
SERLAB
Software Engineering Research

Integrazione e Test di Sistemi Software

Test basati sulle specifiche

Azzurra Ragone

Dipartimento di Informatica - Università degli Studi di Bari
Via Orabona, 4 - 70125 - Bari
Tel: +39.080.5443270 | Fax: +39.080.5442536
serlab.di.uniba.it



Che cosa sono i test basati sulle specifiche?

Test basati sulle specifiche
O
Test funzionali
O
Test della scatola nera

Test basati sulle specifiche

I test basati sulle specifiche sono guidati dai ***requisiti*** *(regole aziendali* che il software implementa e che dobbiamo validare): si **derivano i test dai requisiti**

Funzionalità: cosa il software deve FARE o NON FARE

Requisiti: ad esempio storie utente Agile, casi d'uso UML, semplice testo, ecc.

Di solito questa è la prima tecnica da adottare in ST

Esempio di SubstringsBetween

Metodo di test delle sottostringhe - Between(), ispirato alla libreria Apache Commons Lang (http://mng.bz/nYR5).

Metodo: substringsBetween

Cerca una stringa per sottostringhe delimitate da un tag di inizio e fine, restituendo tutte sottostringhe corrispondenti in un array.

Esempio di SubstringsBetween

Stringa statica pubblica [] substringsBetween(Stringa finale str, Stringa finale aperta, Stringa finale chiusa)

INGRESSO:

str—La stringa contenente le sottostringhe. Null restituisce null; una stringa vuota restituisce un'altra stringa vuota.

aperto: la stringa che identifica l'inizio della sottostringa. Una stringa vuota restituisce nullo.

close—La stringa che identifica la fine della sottostringa. Una stringa vuota restituisce nullo.

PRODUZIONE:

Il programma restituisce un array di stringhe di sottostringhe oppure null se non c'è corrispondenza.

Esempio di SubstringsBetween

Esempio:

str = “axcaycaxc”

aperto = “a”

chiuso = “c”

Qual è il risultato?

Esempio di SubstringsBetween

Esempio:

str = “axcaycaxc”

aperto = “a”

chiuso = “c”

Output: array ["x", "y", "x"].

La sottostringa “a<something>c” appare tre volte nella stringa originale

```
public static String[] substringsBetween(final String str,
 final String open, final String close) {

  if (str == null || isEmpty(open) || isEmpty(close)) {
    return null;
  }

  int strLen = str.length();
  if (strLen == 0) {
    return EMPTY_STRING_ARRAY;
  }

  int closeLen = close.length();
  int openLen = open.length();
  List<String> list = new ArrayList<>();
  int pos = 0;

  while (pos < strLen - closeLen) {
    int start = str.indexOf(open, pos);

    if (start < 0) {
      break;
    }

    start += openLen;
    int end = str.indexOf(close, start);
    if (end < 0) {
      break;
    }

    list.add(str.substring(start, end));
    pos = end + closeLen;

  }

  if (list.isEmpty()) {
    return null;
  }

  return list.toArray(EMPTY_STRING_ARRAY);
}
```

If the pre-conditions do not hold, returns null right away

If the string is empty, returns an empty array immediately

A pointer that indicates the position of the string we are looking at

Looks for the next occurrence of the open tag

Breaks the loop if the open tag does not appear again in the string

Looks for the close tag

Breaks the loop if the close tag does not appear again in the string

Gets the substring between the open and close tags

Moves the pointer to after the close tag we just found

Returns null if we do not find any substrings

metodo substringsBetween

```
public static String[] substringsBetween(final String str,
 final String open, final String close) {

  if (str == null || isEmpty(open) || isEmpty(close)) {
    return null;
  }

  int strLen = str.length();
  if (strLen == 0) {
    return EMPTY_STRING_ARRAY;
  }
```

metodo substringsBetween

```
public static String[] substringsBetween(final String str,
 final String open, final String close) {

  if (str == null || isEmpty(open) || isEmpty(close)) {
    return null;
  }

  int strLen = str.length();
  if (strLen == 0) {
    return EMPTY_STRING_ARRAY;
  }
```

If the pre-conditions do not hold, returns null right away

metodo substringsBetween

axcaycazc

```
int closeLen = close.length();
int openLen = open.length();
List<String> list = new ArrayList<>();
int pos = 0;

while (pos < strLen - closeLen) {
  int start = str.indexOf(open, pos);

  if (start < 0) {
    break;
  }

  start += openLen;
  int end = str.indexOf(close, start);
  if (end < 0) {
    break;
  }

  list.add(str.substring(start, end));
  pos = end + closeLen;

}

if (list.isEmpty()) {
  return null;
}

return list.toArray(EMPTY_STRING_ARRAY);
}
```

x , y, z

metodo substringsBetween

axcaycazc

```
int closeLen = close.length();
int openLen = open.length();
List<String> list = new ArrayList<>();
int pos = 0;

while (pos < strLen - closeLen) {
  int start = str.indexOf(open, pos);

  if (start < 0) {
    break;
  }

  start += openLen;
  int end = str.indexOf(close, start);
  if (end < 0) {
    break;
  }

  list.add(str.substring(start, end));
  pos = end + closeLen;

}

if (list.isEmpty()) {
  return null;
}

return list.toArray(EMPTY_STRING_ARRAY);
}
```

A pointer that indicates the position of the string we are looking at

Looks for the next occurrence of the open tag

Breaks the loop if the open tag does not appear again in the string

Looks for the close tag

Breaks the loop if the close tag does not appear again in the string

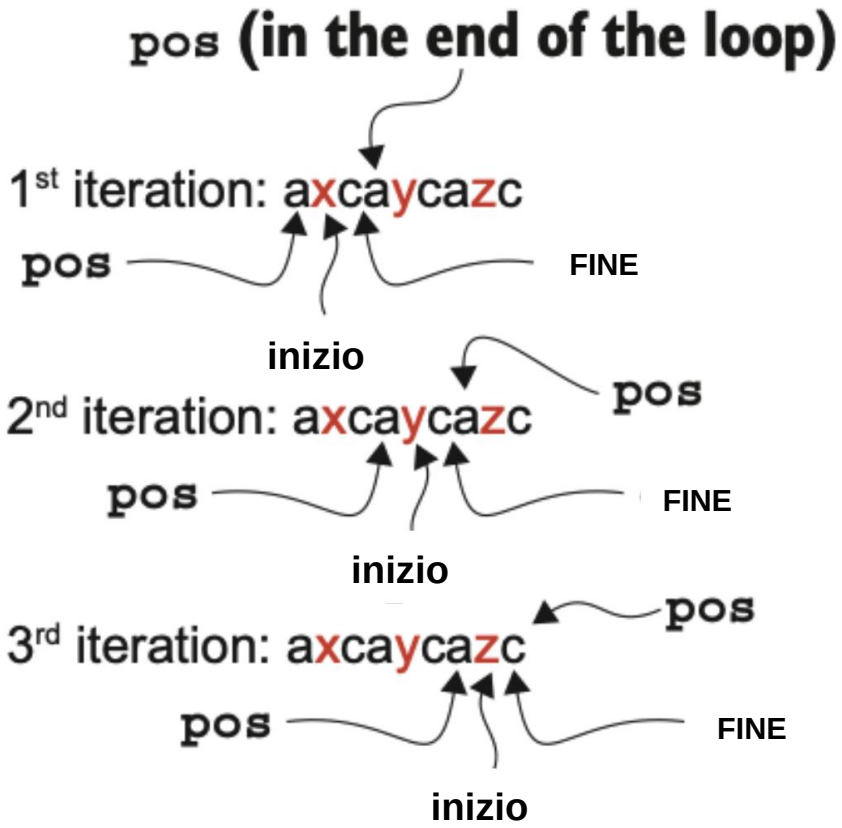
Gets the substring between the open and close tags

Returns null if we do not find any substrings

Moves the pointer to after the close tag we just found

x , y, z

metodo substringsBetween



```
int closeLen = close.length();
int openLen = open.length();
List<String> list = new ArrayList<>();
int pos = 0;

while (pos < strLen - closeLen) {
    int start = str.indexOf( str: open,  fromIndex: pos);

    if (start < 0) {
        break;
    }

    start += openLen;
    int end = str.indexOf( str: close,  fromIndex: start);
    if (end < 0) {
        break;
    }

    list.add(str.substring(start, end));
    pos = end + closeLen;

}

if (list.isEmpty()) {
    return null;
}

return list.toArray( a: EMPTY_STRING_ARRAY);
```

1a iterazione:

pos=0, inizio=1, fine=2

2a iterazione:

pos=3, inizio=4, fine=5

3a iterazione=

pos=6, inizio=7, fine=8

Flusso di lavoro di test per test basati sulle specifiche

1. Comprendere i requisiti (cosa deve fare il programma, input e output)
2. Esplora cosa fa il programma per vari input
3. Esplorare input, output e identificare le partizioni
4. Identificare i casi limite (detti anche casi limite)
5. Ideare casi di test (output Test Plan)
6. Automatizzare i casi di test (output: codice Junit)
7. Arricchire la suite di test con creatività ed esperienza

P.S. Non è possibile testare tutte le possibili combinazioni di input (a volte non è nemmeno conveniente o efficace), i test esaustivi dovrebbero essere sostituiti da un approccio pragmatico.

1) Comprendere i requisiti

Scrivi cosa **dovrebbe fare** il programma e come **gli input** vengono convertiti in **output** previsti

1) Comprendere i requisiti

Scrivi cosa **dovrebbe fare** il programma e come **gli input** vengono convertiti in **output** previsti

*1) L' **obiettivo** di questo metodo è raccogliere tutte le sottostringhe in una stringa delimitate da un tag di apertura e da un tag di chiusura (forniti dall'utente)*

*2) Il programma riceve tre parametri come **input:***

a) str, che rappresenta la stringa da cui il programma estrarrà le sottostringhe b) Il tag di apertura, che indica l'inizio di una sottostringa c) Il tag di chiusura, che indica la fine della sottostringa

*3) Il programma restituisce un array composto da tutte le sottostringhe trovate dal programma **(output).***

2) Esplora cosa fa il programma per vari input

Questo passaggio è particolarmente importante se non hai scritto il codice e vuoi avere un modello mentale chiaro di come dovrebbe funzionare il programma

```
@Test
void simpleCase() {
    assertThat(
            actual: StringUtils.substringsBetween( str: "abcd",  open: "a",  close: "d")
    ).isEqualTo( expected: new String[] { "bc" });
}

no usages
@Test
void manySubstrings() {
    assertThat(
            actual: StringUtils.substringsBetween( str: "abcdabcdab",  open: "a",  close: "d")
    ).isEqualTo( expected: new String[] { "bc", "bc" });
}

no usages
@Test
void openAndCloseTagsThatAreLongerThan1Char() {
    assertThat(
            actual: StringUtils.substringsBetween( str: "aabcddaabfddaab",  open: "aa",  close: "dd")
    ).isEqualTo( expected: new String[] { "bc", "bf" });
}
```

3) Esplorare input, output e identificare le partizioni

Il numero di possibili input e output è quasi infinito

Alcuni set di input fanno sì che il programma si comporti sempre nello stesso modo, indipendentemente dal valore preciso dell'input.

Testare un singolo caso che rappresenti l' intera classe di input

Esplorare:

a - **Input individuali** (classi di input)
b - **Combinazioni** di **input**
c - Classi di **output** (attesi)

Input individuali

parametro str :

1 - Stringa nulla
2 - Stringa vuota
3 - Stringa di lunghezza 1
4 - Stringa di lunghezza > 1
(qualsiasi stringa)

parametro aperto :

1 - Stringa nulla
2 - Stringa vuota
3 - Stringa di lunghezza 1
4 - Stringa di lunghezza > 1
(qualsiasi stringa)

parametro di chiusura :

1 - Stringa nulla
2 - Stringa vuota
3 - Stringa di lunghezza 1
4 - Stringa di lunghezza > 1
(qualsiasi stringa)

Combinazioni di input

(str, apri, chiudi) parametri:

1 - ***str*** non contiene né il tag ***di apertura*** né quello ***di chiusura .*** 2 - ***str*** contiene il tag ***di apertura*** ma non quello ***di chiusura .*** 3 - ***str*** contiene il tag ***di chiusura*** ma non quello di apertura. 4 - ***str*** contiene sia il tag ***di apertura*** che quello ***di chiusura .*** 5 - ***str*** contiene sia il tag ***di apertura*** che ***quello di chiusura*** più volte.

Classi di output (attesi)

Array di stringhe (output):

1 - Array nullo
2 - Array vuoto
3 - Articolo singolo
4 - Articoli multipli

Ogni singola stringa (output):

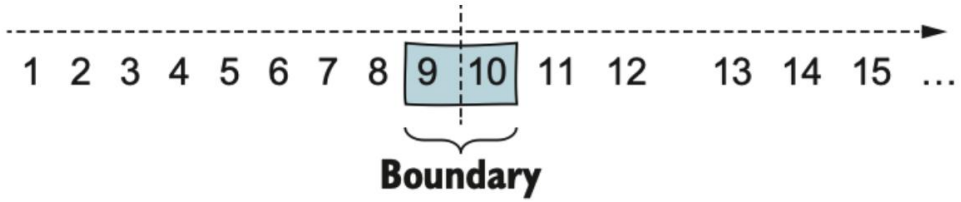
1 - Vuoto
2 - Carattere singolo
3 - Carattere multiplo

4) Identificare i casi limite (detti anche casi limite)

Gli insetti amano i confini!

Test di confine -> il programma dovrebbe funzionare correttamente quando gli input sono vicini ai confini

Es. Se (a>=10)

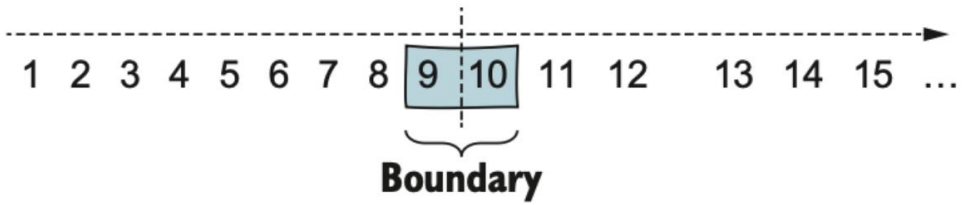


4) Identificare i casi limite (detti anche casi limite)

PUNTI DI ACCENSIONE/SPEGNIMENTO

sul punto: il punto che si trova sul confine;
fuori dal punto: il punto più vicino al confine che appartiene all'altra partizione

Es. Se (a>=10)



4) Identificare i casi limite (detti anche casi limite)

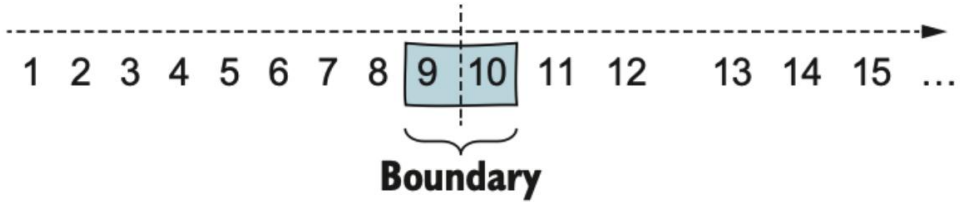
PUNTI DI ACCENSIONE/SPEGNIMENTO

sul punto: il punto che si trova sul confine; fuori dal punto:
il punto più vicino al confine che appartiene all'altra partizione

Es. Se (a>=10)

10 è il punto giusto

9 è il punto di partenza



4) Identificare i casi limite (detti anche casi limite)

PUNTI DI INGRESSO/USCITA

in point: i punti che rendono vera la condizione out point: i punti che rendono falsa la condizione

Es. Se (a=10)

4) Identificare i casi limite (detti anche casi limite)

PUNTI DI INGRESSO/USCITA

Es. Se (a=10)

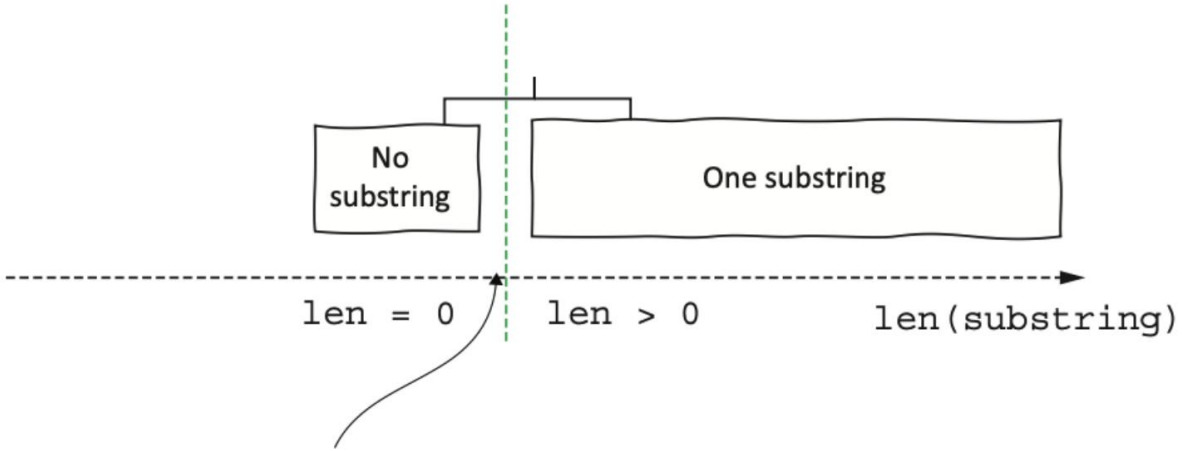
10 è il punto in
6, 8, 12, 52 sono i punti out

4) Identificare i casi limite (detti anche casi limite)

Nel nostro esempio abbiamo due test, uno per ciascun lato del confine:

1 **str contiene sia tag di apertura che di chiusura, *senza* caratteri tra di loro**
2 str contiene sia tag di apertura che di chiusura, con caratteri tra di essi

(Scartamo il test numero 2, poiché altri test coprono già questa situazione)



5) Ideare casi di test

parametro str :

1 - Stringa nulla
2 - Stringa vuota
3 - Stringa di lunghezza 1
4 - Stringa di lunghezza > 1
(qualsiasi stringa)

parametro aperto :

1 - Stringa nulla
2 - Stringa vuota
3 - Stringa di lunghezza 1
4 - Stringa di lunghezza > 1
(qualsiasi stringa)

parametro di chiusura :

1 - Stringa nulla
2 - Stringa vuota
3 - Stringa di lunghezza 1
4 - Stringa di lunghezza > 1
(qualsiasi stringa)

(str, apri, chiudi) parametri:

1 - ***str*** non contiene né il tag ***di apertura*** né quello ***di chiusura .*** 2 - ***str*** contiene il tag ***di apertura*** ma non quello ***di chiusura .*** 3 - ***str*** contiene il tag ***di chiusura*** ma non quello di apertura. 4 - ***str*** contiene sia il tag ***di apertura*** che quello ***di chiusura .*** 5 - ***str*** contiene sia il tag ***di apertura*** che ***quello di chiusura*** più volte.

Se si combinano tutti i possibili input: 4 × 4 × 4 × 5 = 320 test.

È utile sottoporsi a tutti questi test?

5) Ideare casi di test

Decidere pragmaticamente quali partizioni devono essere combinate con altre e quali no

a) Testare i casi eccezionali solo una volta e non combinarli (ad esempio null, empty):

T1: str è nullo.

T2: str è vuoto.

T3: l'apertura è nulla.

T4: aperto è vuoto.

T5: la chiusura è nulla.

T6: close è vuoto.

5) Ideare casi di test

b) Per *una stringa di lunghezza 1* possiamo testare solo questi quattro casi:

T7: Il singolo carattere in str corrisponde al tag aperto.

T8: Il singolo carattere in str corrisponde al tag di chiusura.

T9: Il singolo carattere in str non corrisponde né al tag di apertura né a quello di chiusura.

T10: Il singolo carattere in str corrisponde sia al tag di apertura che a quello di chiusura.

5) Ideare casi di test

c) Una prima combinazione di input:

lunghezza str > 1

lunghezza aperta = 1

lunghezza di chiusura = 1

T11: str non contiene né il tag di apertura né quello di chiusura.

T12: str contiene il tag di apertura ma non contiene il tag di chiusura.

T13: str contiene il tag di chiusura ma non contiene il tag di apertura.

T14: str contiene sia il tag di apertura che quello di chiusura.

T15: str contiene più volte sia il tag di apertura che quello di chiusura.

5) Ideare casi di test

d) Una seconda combinazione di input:

lunghezza str > 1

lunghezza aperta > 1

lunghezza ravvicinata > 1

T16: str non contiene né il tag di apertura né quello di chiusura.

T17: str contiene il tag di apertura ma non contiene il tag di chiusura.

T18: str contiene il tag di chiusura ma non contiene il tag di apertura.

T19: str contiene sia il tag di apertura che quello di chiusura.

T20: str contiene più volte sia il tag di apertura che quello di chiusura.

5) Ideare casi di test

e) Test di confine

T21: str contiene sia il tag di apertura che quello di chiusura senza caratteri tra di essi.

Nota finale:

ci ritroviamo con ***21*** *test anziché* ***320!***

6) Automatizzare i casi di test

T1: str è nullo.

T2: str è vuoto.

```
@Test void strIsNullOrEmpty() {
    assertThat( actual: substringsBetween( str: null,  open: "a",  close: "b")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "",  open: "a",  close: "b")).isEqualTo( expected: new String[]{});
}
```

6) Automatizzare i casi di test

T3: l'apertura è nulla.

T4: aperto è vuoto.

T5: la chiusura è nulla.

T6: close è vuoto.

```
@Test
void openIsNullOrEmpty() {
    assertThat(substringsBetween( str: "abc",  open: null,  close: "b")).isEqualTo( expected: null);
    assertThat(substringsBetween( str: "abc",  open: "",  close: "b")).isEqualTo( expected: null);
}

no usages
@Test
void closeIsNullOrEmpty() {
    assertThat(substringsBetween( str: "abc",  open: "a",  close: null)).isEqualTo( expected: null);
    assertThat(substringsBetween( str: "abc",  open: "a",  close: "")).isEqualTo( expected: null);
}
```

6) Automatizzare i casi di test

T7: Il singolo carattere in str corrisponde al tag aperto.

T8: Il singolo carattere in str corrisponde al tag di chiusura.

T9: Il singolo carattere in str non corrisponde né al tag di apertura né a quello di chiusura.

T10: Il singolo carattere in str corrisponde sia al tag di apertura che a quello di chiusura.

```
@Test
void strOfLength1() {
    assertThat( actual: substringsBetween( str: "a",  open: "a",  close: "b")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "a",  open: "b",  close: "a")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "a",  open: "b",  close: "b")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "a",  open: "a",  close: "a")).isEqualTo( expected: null);
}
```

6) Automatizzare i casi di test

lunghezza str > 1, lunghezza aperta = 1, lunghezza chiusa = 1

T11: str non contiene né il tag di apertura né quello di chiusura.
T12: str contiene il tag di apertura ma non contiene il tag di chiusura.
T13: str contiene il tag di chiusura ma non contiene il tag di apertura.
T14: str contiene sia il tag di apertura che quello di chiusura.
T15: str contiene più volte sia il tag di apertura che quello di chiusura.

```
@Test
void openAndCloseOfLength1() {
    assertThat( actual: substringsBetween( str: "abc",  open: "x",  close: "y")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "abc",  open: "a",  close: "y")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "abc",  open: "x",  close: "c")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "abc",  open: "a",  close: "c")).isEqualTo( expected: new String[] {"b"});
    assertThat( actual: substringsBetween( str: "abcabc",  open: "a",  close: "c")).isEqualTo( expected: new String[] {"b", "b"});
```


6) Automatizzare i casi di test

lunghezza str > 1, lunghezza aperta > 1, lunghezza chiusa > 1

T16: str non contiene né il tag di apertura né quello di chiusura.

T17: str contiene il tag di apertura ma non contiene il tag di chiusura.

T18: str contiene il tag di chiusura ma non contiene il tag di apertura.

T19: str contiene sia il tag di apertura che quello di chiusura.

T20: str contiene più volte sia il tag di apertura che quello di chiusura.

```
@Test
void openAndCloseTagsOfDifferentSizes() {
    assertThat( actual: substringsBetween( str: "aabcc",  open: "xx",  close: "yy")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "aabcc",  open: "aa",  close: "yy")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "aabcc",  open: "xx",  close: "cc")).isEqualTo( expected: null);
    assertThat( actual: substringsBetween( str: "aabbcc",  open: "aa",  close: "cc")).isEqualTo( expected: new String[] {"bb"});
    assertThat( actual: substringsBetween( str: "aabbccaaeecc",  open: "aa",  close: "cc")).isEqualTo( expected: new String[] {"bb", "ee"});
```


6) Automatizzare i casi di test

T21: str contiene sia il tag di apertura che quello di chiusura senza caratteri tra di essi.

```
@Test
void noSubstringBetweenOpenAndCloseTags() {
    assertThat( actual: substringsBetween( str: "aabb",  open: "aa",  close: "bb")).isEqualTo( expected: new String[] {""});
}
```

Nota finale: decidere come raggruppare i test nel metodo di test (fare riferimento alle partizioni)

7) Arricchire la suite di test con creatività ed esperienza

Nei test dovresti sempre considerare le varianti: testare i caratteri speciali è sempre una buona idea. E gli spazi vuoti?

T22: "abcabyt byrc " -> Spazio nella stringa, apri=a, chiudi=c

T23: "a abb ddc ca abbcc" -> Spazio nei tag di apertura e chiusura open='a a' , chiudi= 'c c'

```
T22: assertThat(substringsBetween("abcabyt byrc" , "a", "c"))
                                    .isEqualTo(new String[] {"b", "byt byr"});
```

```
T23: assertThat(substringsBetween("a abb ddc ca abbcc", "a a", "c c")).
                                    isEqualTo(new String[] {"bb dd"});
```

Test basati sulle specifiche: riepilogo

7 passaggi per creare la suite di test:

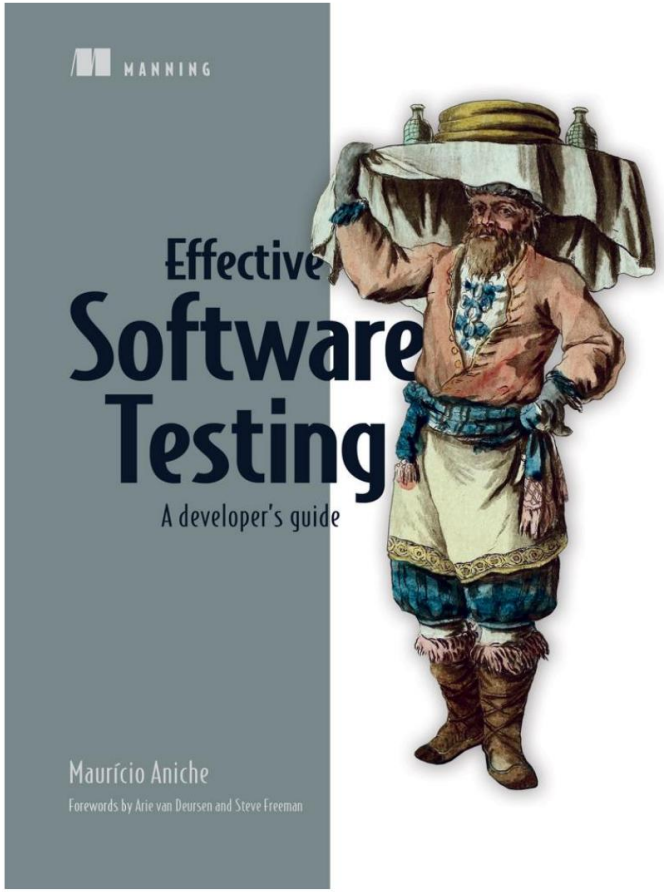
1. Comprensione dei requisiti (cosa deve fare il programma, input e output)

2. Esplorare cosa fa il programma per vari input 3. Esplorare input, output e identificare le partizioni 4. Identificare i casi limite (ovvero i casi limite)
5. Ideare casi di test 6. Automatizzare i casi di test 7. Arricchire la suite di test con creatività ed esperienza

Libro di riferimento:

Test software efficaci. Guida per sviluppatori. Mauricio Aniche. Ed. Manning. **(Capitolo** 2)

Utilizza il codice sconto "au35ani" per uno sconto del 35% sul prezzo.



Riferimenti:

- AssertJ - libreria Java per asserzioni fluide: https://assertj.github.io/doc/
- Javadoc di base di Assertj: https://www.javadoc.io/doc/org.assertj/assertj-core/latest/index.html
- Assertj core javadoc: Asserzioni: https://www.javadoc.io/doc/org.assertj/assertj-core/latest/org/assertj/core/api/Assertions.html
- Introduzione ad AssertJ: https://www.baeldung.com/introduction-to-assertj

Azzurra Ragone

Dipartimento di Informatica - Piano VI - Stanza 616 Email: azzurra.ragone@uniba.it