# Appunti di
# Sicurezza e Affidabilita'

**A cura di:**
Francesco Refolli
Matricola 865955

**Anno Accademico 2022-2023**

# Part I

# Affidabilita

# Chapter 1

# Introduzione ai Test e all'Analisi del Software

## 1.1 Testing Adequacy

Il metodo naive per testare il software e' chiaramente il **Testing Esaustivo**. Ovvero si mette il software in azione provando e controllando tutti i possibili input e tutte le possibili condizioni. Per adottare questo approccio naturalmente e' necessario che venga stesa una specifica per assicurarsi che il programma funzioni come ci si aspetta.

Mentre in teoria e' l'unico vero metodo per testare (quasi) completamente il codice, resta in pratica un metodo infallibile. Questo spesso a causa del fatto che o lo spazio di input e' infinito o troppo elevanto, oppure la totalita' degli stati condizionali e' troppo varia.

La correttezza di un sistema quindi e' un problema indecidibile (vedi Halting Problem). Puo' essere ottenuta solo una ottimistica approssimazione di essa.

## 1.2 Bugs

**Bug** e' una parola utilizzata spesso a sproposito, in quanto e' nei fatti una parola ombrello per diversi tipi di problemi che agiscono su livelli diversi.

| nome | descrizione |
|---|---|
| failure (fallimento) | crash o comportamento scorretto del software |
| fault (difetto) | problema nel codice, puo' causare una **failure** |
| error (errore) | ragione o spiegazione del **fault** |

Il **testing** e' l'attivita' che permette di rivelare i **faults** mostrando le **failures**. Quindi il **testing** fornisce un stima (letteralmente **confidence**) della correttezza.

La percentuale di copertura del codice (**coverage**) pero' non fornisce di per se' motivo di sicurezza. Infatti e' possibile (e nella pratica e' cosi') che il sottospazio degli input che manifestano **failure** sia molto piccolo rispetto allo spazio degli input.

Non potendo fare infiniti test si e' costretti a selezionare quelli giusti e piu' (anche solo potenzialmente) utili tra quelli possibili.

# Chapter 2

# Test Adequacy

## 2.1 Terminologia

**test obligation** Chiamato anche test objective. Specifica di cosa deve essere testato.

**test case** E' un set di input e condizioni appaiati con un risultato o un comportamento che descrivono il fallimento o il successo del test.

**test suite** E' una colezione di **test cases**.

**test catalogue** *da capire*

## 2.2 Test Obligations

**Test Funzionale** Si testa un comportamento della specifica.

**Test Strutturale** Si testa il "comportamento" del codice, ovvero letteralmente i loop e i branch condizionali (= coverage).

**Test Basato su Modello** *da capire*

**Test Basato su Difetto** Verificano la gestione (e quindi la presenza) di un difetto specifico.

**Boundary Testing** Selezionare i **test objectives** che si riferiscono a comportamenti importanti della specifica distinguendo con riguardo tra comportamento normale, eccezioni e casi limite.

## 2.3 Adequacy Criterion

E' un predicato che si dice soddisfatto o meno per una coppia (programma, **test suite**). Quindi e' un insieme di **test obligations**. Una test suite si dice che soddisfa un **adequacy criterion** se per ogni **test obligation** esiste almeno un **test case** nella suite che lo soddisfa e se tutti i **test cases** nella suite passano.

L'adequatezza di un test e' un problema indecidibile, ma ne e' misurabile invece l'inadequatezza, in base alla quale si ricava la lista dei test da modificare o sostituire.

## 2.4 Complementarieta'

Test funzionali e strutturali sono tra loro complementari, infatti alcune situazioni o alcune **failures** possono rilevate solo tramire una combinazione di essi.

## 2.5 Testing Sistematico e Randomico

**Random** Si usando input randomici. Questo permette di evitare il problema del bias del tester: il tester portrebbe subire le stesse sviste del programmatore. Ma non e' detto che manifesti i problemi: la probabilita' (essendo quasi uniforme) e' la stessa che selezioni i valori che non danno problemi, o addirittura molto ridotta nel caso in cui la probabilita' di manifestazione di un difetto sia non uniforme (il caso reale).

**Systematic** Si scelgono gli input che sono considerati come di maggior valore. Permette di controllare dei casi esplicitamente. Il **testing funzionale** e' sistematico.

Tendenzialmente le **failures** sono sparse nello spazio degli input ma probabilmente dense in alcuni punti. Si cerca di trovare questi punti sfruttando la conoscenza pregressa del software.

Successive analisi sui test permettono di stabilire l'inadeguatezza dei test, quindi si ottengono in teoria delle partizioni sempre migliori.

# Chapter 3

# Combinatorial Testing

## 3.1 Dalla specifica ai test case

**Decomporre la specifica** La specifica e' gonfia e confusionaria, occorre spezzettarla in caratteristiche individualmente testabili Ci si concentra separatamente su come la specifica descrive il comportamento del sistema rispetto a input, parametri e risultati. Quindi si applica **boundary testing** per ognuno di questi.

**Aggregazione per Combinazione** Si formano delle combinazioni che permettono di testare contemporaneamente diverse di queste condizioni. Quindi si implementano queste combinazioni nei **text case**.

## 3.2 Concetti Chiave

- Category-partition Testing (Decompose)
- Pairwise Testing (Aggregate)
- Catalog-based Testing (Decompose)

**Category-partition Testing** Identificazione manuale dei rappresentanti per i test objectives delle categorie che caratterizzano l'input space. Quindi si generano automaticamnete tutte le combinazioni rilevanti ai fini del testing.

**Pairwise Testing** Testing sistematico delle interazioni tra gli attributi dell'input space. Questo puo' essere fatto attraverso software specifici che permettono di generare quelle combinazioni sufficienti a testare tutte le interazioni a coppie. Chiaramente e' possibile che le interazioni siano piu' complesse di semplici coppie. Si fare la stessa cosa con combinazioni di n-attributi contemporaneamente. Il concetto rimane lo stesso.

**Catalog-based Testing** Aggregazione o sintesi dell'esperienza dei test designers in un particolare dominio per identificare i valori degli attributi.

## 3.3 Altri Concetti Chiave

- inputs
- parametri
- risultati
- caratteristiche
- controllabilita'

**inputs** Gli input sono esplicitamente indicati dalla specifica.

**parametri** Altri fattori che possono essere variabili e che alterano il comportamento del software.

**risultati** Sono le condizioni misurabili che il programma resistituisce, sia come valori sia come comportamento. Si vuole testare tutti i distiti tipi di risultati osservabili durante i test.

**caratteristiche** Sono caratteristiche intriseche di input, parametri e risultati.

**controllabilita'** Si definisce controllabile un qualcosa che puo' essere ottenuto durante l'esecuzione. Non si e' interessati a testare valori che si sa con certezza che non possano essere ottenuti a runtime.

**Vincoli** Alcuni attributi possono porre vincoli ad altri, come la lunghezza di un array ai valori contenuti nell'array. Questi vincoli sono utili per intuire quali combinazioni e' inutile / impossibile testare. I vincoli possono introdurre categorie che rendono superfluo testare piu' di un rappresentante per categoria.

# Chapter 4

# Testing Strutturale

# Chapter 5

# Esercizi di Testing Combinatorio

## 5.1 Esercizio 1

```
int deleteIfEqual(String[] theSet, String toBeDeleted);
```

La funzione **deleteIfEqual** rimuove gli elementi di un array di al massimo 100 elementi che sono uguali a **toBeDeleted**. Se l'array ha piu' di 100 elementi ritorna -100. Se l'array e' vuoto ritorna -1. Altrimenti ritorna il numero di elementi eliminati.

**Analisi delle Categorie**

- lunghezza dell'array di input
- la quantita' di elementi eliminati

**Obiettivi**

- array con lunghezza (0, 1, $1 < x < 100$, 100, 101)
- quantita' di elementi eliminati (0, $0 < x \leq 100$)

## 5.2 Esercizio 2

```
double registratoreDiCassa(Prodotto[] carrello, PoliticaDiSconto pSconto);
```

**Analisi delle Categorie**

- lunghezza del carrello
- numero di prodotti di tipo differente
- numero di prodotti per i quali sono presenti piu' prodotti dello stesso tipo
- numero di prodotti per un certo tipo
- politica con sconto senza soglia minima
- politica con sconto con soglia minima
- valore della soglia minima per lo sconto su un tipo di prodotto

**Obiettivi**

## 5.3 Esercizio 3

**1**

- $t_1$ throws IllegalArgumentException
- $t_2$ returns 0

**2**

- $t_1 Rightarrow \frac{1}{8}\%$
- $t_1 \wedge t_2 Rightarrow \frac{5}{8}\%$

## 5.4 Esercizio 4

**1** basic condition coverage with $t_1 \wedge t_2 \wedge t_3 \wedge t_4 \Rightarrow \frac{10}{12}$

## 5.5 Esercizio 5

**1** basic condition coverage = $\frac{11}{14}$

**2**

- t5 = 3,"", "", "",2,0
- t6 = 2,"", "", "",0,0
- t7 = 1,"", "", "",2,0

**3** NO, copa

## 5.6 Esercizio 6

- 1,0,100
- -1,-1,99

# Part II

# Sicurezza

# Chapter 6

# Crittografia