



# Master Test Plan

MARK 2.0 Plus

|            |                               |
|------------|-------------------------------|
| Data       | 19/01/2026                    |
| Presentato | Cerchia Giovanni (NF22500202) |
| da         | Medica Vincenzo (NF22500203)  |



## Sommario

|   |   |
|---|---|
| 1 Introduzione                                      | 3 |
| 2 Strategia di Testing                              | 3 |
| 2.1 Basic Unit Testing (white-box)                  | 3 |
| 2.2 Integration Testing (white-box)                 | 3 |
| 2.3 System Testing (black-box, Category Partition)  | 3 |
| 3 Componenti incluse, componenti escluse e coverage | 4 |
| 3.1 Componenti incluse                              | 4 |
| 3.2 Componenti escluse                              | 4 |
| 3.3 Coverage  | 4 |

# 1 Introduzione

Il presente **Master Test Plan** descrive l'approccio di testing adottato per **MARK 2.0 Plus**, con riferimento alle funzionalità della baseline e alle evoluzioni introdotte dalle tre Change Requests (CR1-CR3). L'obiettivo è definire in modo chiaro: i livelli di test previsti (unità, integrazione, sistema), le tecniche utilizzate e i componenti su cui concentrare lo sforzo di verifica, includendo anche un criterio di copertura strutturale per le parti testate in white-box.

## 2 Strategia di Testing

La strategia è articolata su tre livelli:

### 2.1 Basic Unit Testing (white-box)

Verrà effettuato un **basic unit testing** focalizzato sui **metodi** delle classi selezionate, con l'obiettivo di verificare la logica interna dei rami principali. Quando necessario, le dipendenze verranno **mockate** per isolare il metodo sotto test e controllare lo stato dell'ambiente (ad es. file system, servizi del tool). La progettazione dei test avverrà in modalità **white-box**: per ciascun metodo target verrà costruito il relativo **Control Flow Graph (CFG)** tramite uno script Python basato su **AST**. A partire dal CFG verranno individuati i percorsi che coprono il maggior numero di branch e verranno scelti gli input in grado di attivarli. I percorsi selezionati verranno poi implementati come casi di test.

### 2.2 Integration Testing (white-box)

Verrà eseguito anche **integration testing** sui componenti scelti, per verificare le interazioni tra classi e servizi senza isolare artificialmente il comportamento. In questa fase **non verranno mockate le dipendenze**: i test utilizzeranno interazioni reali (file system reale/temporaneo, servizi del tool).

Anche qui la selezione dei test verrà guidata dal CFG, con l'obiettivo di coprire i branch più significativi dei metodi di interesse.

### 2.3 System Testing (black-box, Category Partition)

Il **system testing** verrà condotto in modalità **black-box** progettando i casi di test tramite **Category Partition**. La definizione delle categorie e delle scelte riguarderà i parametri di input e lo stato dell'ambiente, che insieme agli oracoli

saranno ricavati dalle specifiche delle funzionalità del sistema espresse in forma di casi d'uso.

I test di sistema saranno pianificati:

- sul **sistema pre-modifica**, così da ottenere una suite riutilizzabile per **regressione** post-CR;
- sulle funzionalità introdotte da **CR1-CR3**, con casi dedicati per ciascuna nuova feature.

Per favorire ripetibilità ed evitare modifiche manuali a parametri hard-coded, dove opportuno l'esecuzione dei test di sistema verrà effettuata tramite `main_args.py` con pochi flag essenziali e valori di default invariati (es. analisi su directory o cloning da CSV).

## 3 Componenti incluse, componenti escluse e coverage

### 3.1 Componenti incluse

Unit testing (basic, sui metodi):

- tutti i metodi del sistema che saranno **modificati dalla CR1**;
- metodi delle classi **Service** della GUI, responsabili dell'interfacciamento tra Controller e servizi del sistema/file system.

Integration testing:

- gli stessi metodi scelti per lo unit-testing;
- metodi della classe **Controller** della GUI, per verificare la logica di orchestrazione tra componenti grafiche e classi Service.

### 3.2 Componenti escluse

Le classi **View** della GUI verranno escluse dal testing, poiché si occupano prevalentemente di **logica di presentazione** (layout, rendering) e non contengono logica applicativa core; lo sforzo di verifica viene quindi concentrato su Controller/Services e sui componenti di analisi modificati.

### 3.3 Coverage

Per unit e integration testing verrà adottato un obiettivo di **≥ 80% di branch coverage** sulle classi di interesse del testing. La copertura verrà misurata con **pytest-cov** e calcolata **solo** sul sottoinsieme di componenti target.