



TCS - Test Case Specification

CodeFace4Smells

Riferimento	
Versione	1.0
Data	07/07/2025
Destinatario	Prof. Andrea De Lucia
Presentato da	Antonio Ferrentino, Francesco Perilli, Giuseppe Napolitano

Composizione gruppo	
Antonio Ferrentino	0522501898
Francesco Perilli	0522502073
Giuseppe Napolitano	0522501961

Cronologia revisioni

Data	Versione	Descrizione	Autori
11/07/2025	0.1	Inizio stesura del documento	Giuseppe Napolitano
12/07/2025	0.2	Fine scrittura del documento	Giuseppe Napolitano
14/07/2025	0.3	Ulteriori modifiche e correzioni del documento	Giuseppe Napolitano
16/07/2025	1.0	Revisione documento	Giuseppe Napolitano, Antonio Ferrentino, Francesco Perilli



Contents

Cronologia revisioni	2
1 Introduzione	4
2 Test Case Specification	4
2.1 Test configurazione ambiente	4
2.2 Test analisi cppstats	5
2.3 Test script R	5
2.4 Configurazione batchjob con rilevamento revisioni/tag	6
2.5 Test clustering sviluppatori	6
2.6 Test estrazione feature linee	7
2.7 Test CLI Codeface	7
2.8 Test logger	8
2.9 Test database MySQL	8
2.10 Test configurazione progetto	9
2.11 Installazione automatica tramite Vagrant	9
2.12 Configurazione MySQL	10
2.13 Importazione repository Git	10
2.14 Esecuzione analisi	10
2.15 Rilevamento revisioni e tag	11
2.16 Analisi temporale	11
2.17 Analisi Strutturale	11
2.18 Rilevamento Code Smells	12
2.19 Generazione Report	12
2.20 Visualizzazione grafi	12
2.21 Esportazione risultati	13
2.22 Esecuzione Test	13
2.23 Gestione ambienti separati	13



1 Introduzione

Un test case è un insieme di input e di risultati attesi che servono a testare una componente per individuare comportamenti diversi da quelli attesi, cioè i failure. Il seguente documento riporta la specifica dei test case, attraverso la descrizione dei seguenti attributi:

- Nome;
- ID;
- Pre-Condizioni;
- Flusso degli eventi;
- Input;
- Oracolo;

2 Test Case Specification

2.1 Test configurazione ambiente

Test Case ID:					
TCU_01					
Pre-Condizioni:					
Sistema funzionale e correttamente installato					
Flusso degli eventi:					
1. L'utente esegue il comando					
<table><tr><th>Input</th><th>Valore</th></tr><tr><td>Comando</td><td>'test_configuration.py'</td></tr></table>	Input	Valore	Comando	'test_configuration.py'	
Input	Valore				
Comando	'test_configuration.py'				
2. L'utente visualizza le dipendenze create					
Oracolo:					
L'ambiente Python è installato correttamente, senza errori					



2.2 Test analisi cppstats

Test Case ID:		
TCU_02		
Pre-Condizioni:		
Presenza del file eseguibile <code>cppstats</code> nella path specificata, progetto Git importato e configurazione valida nel file <code>codeface.conf</code> .		
Flusso degli eventi:		
<ol style="list-style-type: none">1. L'utente esegue il comando per avviare i test.2. Il sistema avvia le analisi statiche tramite <code>cppstats</code>.3. L'output generato viene salvato nella directory specificata.		
Input:		
Parametro	Valore	
Comando	<code>codeface test -c codeface.conf</code>	
Configurazione	File <code>codeface.conf</code> con path e parametri validi	
Progetto Git	Repository precedentemente importata	
Oracolo:		
L'output di <code>cppstats</code> viene generato correttamente, senza errori. I file di metrica risultano popolati e coerenti con la struttura del progetto analizzato.		

2.3 Test script R

Test Case ID:		
TCU_03		
Pre-Condizioni:		
Presenza dei file di script <code>.R</code> all'interno della directory di progetto. Ambiente R installato correttamente nella VM.		
Flusso degli eventi:		
<ol style="list-style-type: none">1. L'utente accede alla cartella contenente gli script R.2. Esegue uno script tramite comando da terminale R.3. Lo script analizza i dati di input e produce output in CSV/HTML.4. I file vengono salvati nella directory specificata nel codice R.		
Input:		
Parametro	Valore	
Script R	<code>analyse-ts.R</code> , <code>graph-generator.R</code> , ecc.	
Comando	<code>Rscript analyse-ts.R</code>	
Input dati	File CSV/SQLite generati dalla fase precedente	
Oracolo:		
Gli script R producono i risultati previsti (grafici, tabelle o file CSV/HTML). Non devono verificarsi errori di esecuzione o crash. I risultati devono essere coerenti con i dati forniti in input.		



2.4 Configurazione batchjob con rilevamento revisioni/tag

Test Case ID:

TCU_04

Pre-Condizioni:

Repository Git valido disponibile. File di configurazione batchjob correttamente compilato.

Flusso degli eventi:

1. L'utente avvia l'analisi batch specificando il file di configurazione del progetto.
2. Il sistema avvia la pipeline completa di elaborazione.
3. Il sistema esegue il rilevamento delle revisioni e dei tag dal repository associato.

Input:

Parametro	Valore
File progetto	project.conf
File configurazione	config.conf
Comando	codeface run -c config.conf -p project.conf outputDir

Oracolo:

Il sistema produce correttamente i file di output dell'analisi, incluso l'elenco delle revisioni e dei tag trovati nel repository Git. Non devono esserci errori nella logica di configurazione batchjob.

2.5 Test clustering sviluppatori

Test Case ID:

TCU_05

Pre-Condizioni:

Presenza del file Python `test_cluster.py`. Repository Git correttamente importata e cronologia dei commit disponibile. Database popolato con dati di commit.

Flusso degli eventi:

1. L'utente esegue lo script `test_cluster.py` in ambiente Python.
2. Lo script accede ai dati dei commit e applica algoritmi di clustering.
3. Viene prodotto un output con i cluster individuati, salvati o visualizzati a video.

Input:

Parametro	Valore
Script Python	<code>test_cluster.py</code>
Dati	Commit estratti dalla repository e salvati nel database
Comando	<code>python3 test_cluster.py</code>

Oracolo:

Lo script restituisce gruppi di sviluppatori correttamente clusterizzati in base a metriche strutturali (es. file modificati, frequenza commit, co-evoluzione). Nessun errore di esecuzione deve verificarsi.



2.6 Test estrazione feature linee

Test Case ID:

TCU_06

Pre-Condizioni:

Presenza del file Python `test_features.py`. Disponibilità di file sorgente (C/C++ o altri) da analizzare. Modulo `cppstats` o simile configurato.

Flusso degli eventi:

1. L'utente esegue lo script `test_features.py`.
2. Lo script apre i file sorgente e individua caratteristiche come numero linee, funzioni, metodi o direttive particolari.
3. I risultati vengono visualizzati a console o salvati in un file di output.

Input:

Parametro	Valore
Script Python	<code>test_features.py</code>
File da analizzare	Sorgenti C/C++ all'interno del progetto
Comando	<code>python3 test_features.py</code>

Oracolo:

Lo script identifica correttamente le caratteristiche strutturali previste nei file analizzati (numero righe, funzioni, direttive, ecc.). Nessun errore di parsing o analisi deve verificarsi.

2.7 Test CLI Codeface

Test Case ID:

TCU_07

Pre-Condizioni:

Presenza di una configurazione valida nel file `codeface.conf`, progetto Git importato correttamente, ambiente Python attivo.

Flusso degli eventi:

1. L'utente apre un terminale nella root del progetto.
2. Esegue il comando CLI per avviare l'analisi.
3. Codeface processa i file secondo le impostazioni presenti nel file di configurazione.
4. I risultati vengono prodotti nella directory di output.

Input:

Parametro	Valore
Comando	<code>codeface run -c codeface.conf -p project.conf outputDir</code>
File conf	<code>codeface.conf</code> , <code>project.conf</code>
Repository	Progetto Git precedentemente importato

Oracolo:

Il sistema completa l'esecuzione del comando senza errori. I file di output sono generati correttamente nella directory indicata. Gli output corrispondono alle aspettative in base alla configurazione fornita.



2.8 Test logger

Test Case ID:

TCU_08

Pre-Condizioni:

Sistema inizializzato, file di configurazione validi (`codeface.conf`, `project.conf`), ambiente Python attivo e directory di output definita.

Flusso degli eventi:

1. L'utente esegue un comando di analisi o test tramite CLI.
2. Il sistema avvia il processo e produce l'output configurato.
3. Durante l'esecuzione, vengono scritti i dettagli in uno o più file di log (es. `run.log`, `error.log`).
4. I log vengono salvati nella directory di output o logs specificata.

Input:

Parametro	Valore
Comando	<code>codeface run -c codeface.conf -p project.conf outputDir</code>
Configurazione	Logging abilitato nel file <code>codeface.conf</code>
Output atteso	File <code>.log</code> generati (es. <code>run.log</code>)

Oracolo:

I file di log vengono effettivamente generati nella directory designata. I contenuti devono riportare timestamp, dettagli delle operazioni eseguite, messaggi di errore (se presenti) e stato finale dell'operazione.

2.9 Test database MySQL

Test Case ID:

TCU_09

Pre-Condizioni:

MySQL installato e attivo. File di configurazione database disponibile (es. `schema.sql`, `codeface.conf`). Ambienti separati definiti (es. test e produzione).

Flusso degli eventi:

1. L'utente esegue lo script SQL per la creazione dello schema.
2. Verifica che il database venga creato correttamente.
3. Codeface si connette al database durante l'esecuzione delle analisi.
4. Si verifica l'utilizzo del database corretto in base all'ambiente (produzione/test).
5. Si verifica che i dati vengano letti/scritti correttamente.

Input:

Parametro	Valore
Script SQL	<code>schema.sql</code>
Comando	<code>mysql -u codeface -p < schema.sql</code>
Configurazione	<code>codeface.conf</code> (con parametri DB)
Ambienti	DB_TEST, DB_PROD separati

Oracolo:

Il database viene creato correttamente, è accessibile e lo schema risulta inizializzato. Il sistema accede al database corretto in base all'ambiente, e le operazioni CRUD funzionano senza errori.



2.10 Test configurazione progetto

Test Case ID:										
TCU_10										
Pre-Condizioni:										
Repository Git accessibile (in locale o remoto), presenza del file di configurazione progetto (es. <code>project.conf</code>), ambiente di esecuzione pronto.										
Flusso degli eventi:										
<ol style="list-style-type: none">1. L'utente clona o indica il percorso del repository da analizzare.2. Configura i parametri nel file <code>project.conf</code>.3. Esegue un'analisi preliminare (es. <code>codeface import</code> o <code>setup</code> progetto).4. Il sistema carica i metadati del progetto nella propria struttura dati.										
Input:										
<table><tr><th>Parametro</th><th>Valore</th></tr><tr><td>Repository</td><td><code>https://github.com/utente/progetto.git</code></td></tr><tr><td>File configurazione</td><td><code>project.conf</code></td></tr><tr><td>Comando</td><td><code>codeface import -c project.conf</code></td></tr></table>	Parametro	Valore	Repository	<code>https://github.com/utente/progetto.git</code>	File configurazione	<code>project.conf</code>	Comando	<code>codeface import -c project.conf</code>		
Parametro	Valore									
Repository	<code>https://github.com/utente/progetto.git</code>									
File configurazione	<code>project.conf</code>									
Comando	<code>codeface import -c project.conf</code>									
Oracolo:										
Il repository viene importato correttamente, e il sistema visualizza i metadati (revisioni, tag, struttura) del progetto all'interno della propria interfaccia dati.										

2.11 Installazione automatica tramite Vagrant

Test Case ID:					
TCS_11					
Pre-Condizioni:					
Sistema con Vagrant installato (progetto clonato).					
Flusso degli eventi:					
1. L'utente esegue il comando					
<table><tr><th>Input</th><th>Valore</th></tr><tr><td>Comando</td><td>vagrant up</td></tr></table>	Input	Valore	Comando	vagrant up	
Input	Valore				
Comando	vagrant up				
2. La macchina viene avviata e configurata.					
Oracolo:					
La macchina viene avviata e configurata automaticamente. Il provisioning viene completato senza errori.					



2.12 Configurazione MySQL

Test Case ID:

TCS_12

Pre-Condizioni:

Sistema MySQL attivo, file schema disponibile.

Flusso degli eventi:

1. L'utente esegue il comando

Input	Valore
Comando	mysql -u codeface -p -e "SHOW VARIABLES LIKE 'local_infile';"

2. Viene visualizzata da shell lo schema caricato.

Oracolo:

Database creato correttamente con schema inizializzato.

2.13 Importazione repository Git

Test Case ID:

TCS_13

Pre-Condizioni:

URL o path di un progetto Git

Flusso degli eventi:

1. L'utente esegue il comando per clonare la repo

Input	Valore
Comando	git clone 'repo_scelta'

2. La repo viene clonata.

Oracolo:

Repository disponibile nella struttura dati del sistema.

2.14 Esecuzione analisi

Test Case ID:

TCS_14

Pre-Condizioni:

Presenza del .conf del progetto da analizzare e la directory in output

Flusso degli eventi:

1. L'utente esegue il comando per analizzare il progetto clonato

Input	Valore
Comando	'codeface run -c config.conf -p project.conf outputDir'.

2. Il progetto viene analizzato

3. Viene riportato nella dir di output i risultati

Oracolo:

Ottenimento di boxplot e risultato delle analisi del progetto



2.15 Rilevamento revisioni e tag

Test Case ID:	
TCS_15	
Pre-Condizioni:	
Repo Git valida	
Flusso degli eventi:	
1. L'utente esegue l'analisi su un progetto con diversi tag multipli	
Input	Valore
Repository di github	URL della repository
2. Il progetto viene analizzato	
3. Viene riportata una lista delle revisioni	
Oracolo:	
Ottenimento di tutti i tag e le revisioni del progetto	

2.16 Analisi temporale

Test Case ID:	
TCS_16	
Pre-Condizioni:	
Presenza .conf, script R attivi	
Flusso degli eventi:	
1. L'utente esegue lo script	
Input	Valore
Comando	'analyse-ts.R'
2. Il progetto viene analizzato	
3. L'utente ottiene il risultato delle analisi	
Oracolo:	
Ottenimento dei grafici associati e serie temporali nel database	

2.17 Analisi Strutturale

Test Case ID:	
TCS_17	
Pre-Condizioni:	
Presenza di Repository con cronologia commit	
Flusso degli eventi:	
1. L'utente esegue la pipeline strutturale	
2. L'utente visualizza i cluster e i grafi popolati	
Oracolo:	
I risultati devono essere visibili nei report	



2.18 Rilevamento Code Smells

Test Case ID:					
TCS_18					
Pre-Condizioni:					
Repository da analizzare					
Flusso degli eventi:					
1. L'utente esegue il comando					
<table><tr><th>Input</th><th>Valore</th></tr><tr><td>Comando</td><td>'detect-smells'</td></tr></table>	Input	Valore	Comando	'detect-smells'	
Input	Valore				
Comando	'detect-smells'				
2. Vengono rilevati gli smells					
Oracolo:					
JSON contenente tutti gli smells rilevati					

2.19 Generazione Report

Test Case ID:	
TCS_19	
Pre-Condizioni:	
Repository Github presente	
Flusso degli eventi:	
1. L'utente esegue l'analisi tramite comando	
2. Vengono generati i report e salvati correttamente	
Oracolo:	
I report contengono le metriche e i grafici attesi	

2.20 Visualizzazione grafi

Test Case ID:	
TCS_20	
Pre-Condizioni:	
Presenza dei file SVG/HTML	
Flusso degli eventi:	
1. L'utente esegue i file	
2. L'utente visualizza i grafi	
Oracolo:	
Grafici chiari e completi, tutti i nodi sono connessi e visibili	



2.21 Esportazione risultati

Test Case ID:
TCS_21
Pre-Condizioni:
Directory di output correttamente popolata dopo l'analisi
Flusso degli eventi:
1. L'utente visualizza l'analisi
2. L'utente visualizza i risultati correttamente esportati
Oracolo:
File corretti e coerenti

2.22 Esecuzione Test

Test Case ID:	
TCS_22	
Pre-Condizioni:	
Presenza .conf di test	
Flusso degli eventi:	
1. L'utente esegue lo script	
Input	Valore
Comando	'codeface test -c conf'
2. L'utente ottiene i risultati del test	
Oracolo:	
Tutti i test vengono eseguiti correttamente	

2.23 Gestione ambienti separati

Test Case ID:
TCS_23
Pre-Condizioni:
Avere a disposizione diversi DB attivi
Flusso degli eventi:
1. L'utente esegue il test e analisi su due DB separati
2. L'utente ottiene nessuna interferenza tra ambienti
Oracolo:
I dati restano confinati nel DB corretto