UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

TEST COMPLETATION REPORT

INGEGNERIA, GESTIONE ED EVOLUZIONE DEL SOFTWARE

DOCENTI

Prof. Andrea De Lucia

TUTOR

Dott. Stefano Lambiase

Università degli Studi di Salerno

STUDENTI

Francesco Maria Torino (0522501879)

Francesco Alessandro Pinto (0522501981)

Stefano Guida (0522502054)

Anno Accademico 2024-2025

Indice

Elenco delle figure

Elenco delle tabelle

CAPITOLO 1

Introduzione

Il presente documento costituisce il **Test Completion Report** relativo al progetto di manutenzione evolutiva del sistema **GUIDO** (Gathering and Understanding Socio-Technical Aspects in Development Organizations). L'obiettivo principale del documento è duplice: da un lato, fornire una panoramica sintetica dei risultati ottenuti dall'esecuzione dei test preesistenti nel sistema prima dell'attività di manutenzione (già analizzati in dettaglio nel *Test Plan*); dall'altro, riportare in modo strutturato ed esaustivo gli esiti dei test condotti a seguito dell'implementazione di ciascuna *Change Request* (CR) prevista nel piano evolutivo del progetto. Per ogni CR verranno descritte le strategie di verifica adottate, la tipologia di test eseguiti e i principali risultati emersi.

Il documento si conclude con una valutazione complessiva dello stato del sistema al termine del processo di verifica, includendo metriche quantitative quali la percentuale di test superati e la *branch coverage*, oltre a considerazioni qualitative sull'efficacia complessiva delle attività di testing.

CAPITOLO 2

Esito dei Test Pre-Manutenzione

2.1 Panoramica generale

Prima dell'avvio delle attività di manutenzione evolutiva, il sistema GUIDO disponeva già di una suite di test articolata su tre livelli: *unit test*, *integration test* e *system test*. La presenza di un numero consistente di casi di test ha costituito una buona base di partenza per il lavoro successivo, dimostrando una precedente attenzione alle attività di verifica e validazione del software.

Tuttavia, una prima esecuzione della suite ha evidenziato diverse criticità. In particolare, circa il **20% dei test** non sono stati superati con successo. I malfunzionamenti erano distribuiti trasversalmente tra i vari livelli di testing e spesso riconducibili a problemi di configurazione ambientale, codice non aggiornato o incompleto, e debole coerenza tra la progettazione e l'effettiva implementazione dei test. Per quanto riguarda i *test di sistema*, sono state riscontrate alcune imperfezioni nella loro progettazione e realizzazione. Inoltre, sebbene fossero stati specificati utilizzando la tecnica del *Category Partitioning*, in diversi casi i alcuni test case non sono stati effettivamente implementati, compromettendo la copertura effettiva delle combinazioni input/output rilevanti.

Di seguito si riporta una sintesi dei risultati ottenuti nell'esecuzione della suite preesistente:

- **Totale test eseguiti**: 55
- **Test superati**: 44
- **Test falliti**: 11
- **Tempo totale di esecuzione**: 91.02 secondi
- **Ambiente di test**: Python 3.10.0, pytest 8.2.0, Windows

2.2 Test di Unità e Integrazione

La suite preesistente includeva **30 test di unità**, dei quali **26 superati** e **4 falliti**. I test erano implementati principalmente con `pytest`, seguendo un approccio *white-box*.

Nome del Test	Esito
TestIntentResolverWebService::test_missing_data	Passed
TestIntentResolverWebService::test_missing_entities	Passed
TestIntentResolverWebService::test_resolver_exception_handling	Passed
TestIntentResolverWebService::test_resolve_success	Passed
TestCultureInspectorUnit::test_check_list_null	Failed
TestCultureInspectorUnit::test_check_list_format	Failed
TestCultureInspectorUnit::test_success	Passed
TestIntentManager::test_detect_intent_report	Failed
TestIntentManager::test_detect_intent_info	Passed
TestIntentManager::test_detect_intent_get_smells	Passed
TestIntentManager::test_detect_intent_get_smells_date	Failed
TestToolSelectorUT::test_setter	Passed

TestToolSelectorUT::test_run	Passed
TestLanguageHandlerUT::test_return_same_instance	Passed
TestLanguageHandlerUT::test_detect_language	Passed
TestLanguageHandlerUT::test_get_current_language	Passed
TestLanguageHandlerUT::test_replace_message	Passed
TestUtils::test_valid_link_with_valid_url	Passed
TestUtils::test_valid_link_with_invalid_url	Passed
TestUtils::test_valid_date_with_valid_date	Passed
TestUtils::test_valid_date_with_valid_date_exception	Passed
TestUtils::test_valid_date_with_invalid_date	Passed
TestUtils::test_enum_values	Passed
TestIntentResolverUT::test_resolve_intent_get_smells_and_date1	Passed
TestIntentResolverUT::test_resolve_intent_get_smells_and_date2	Passed
TestIntentResolverUT::test_resolve_intent_info_and_report1	Passed
TestIntentResolverUT::test_resolve_intent_info_and_report2	Passed
TestCsDetectorToolUT::test_execute_tool_w_date[data0-files0]	Passed
TestCsDetectorToolUT::test_execute_tool_w_date[data1-files1]	Passed
TestCsDetectorToolUT::test_execute_tool_w_date[data2-files2]	Passed

Totale test: 30 **Passati:** 26 **Falliti:** 4

I **13 test di integrazione** registravano **8 successi** e **5 fallimenti**. La metodologia adottata era *bottom-up*, coerente con la struttura modulare del sistema.

Nome del Test	Esito
TestIntentResolverIntegration::test_resolve_intent_with_message	Failed
TestIntentResolverIntegration::test_resolve_intent_with_entities	Failed
TestIntentResolverIntegration::test_resolve_intent_invalid_request	Passed

TestIntentResolverIntegration::test_resolve_intent_exception	Passed
TestCultureInspectorIntegration::test_check_list_null	Failed
TestCultureInspectorIntegration::test_check_list_format	Failed
TestCultureInspectorIntegration::test_success	Failed
TestToolSelectorIT::test_setter	Passed
TestToolSelectorIT::test_run	Passed
TestIntentResolverIT::test_resolve_intent_get_smells_and_date1	Passed
TestIntentResolverIT::test_resolve_intent_get_smells_and_date2	Passed
TestIntentResolverIT::test_resolve_intent_info_and_report1	Passed
TestIntentResolverIT::test_resolve_intent_info_and_report2	Passed

Totale test: 13 **Passati:** 8 **Falliti:** 5

Le principali criticità riscontrate in questi livelli di test includevano:

- la presenza di test riferiti a funzionalità deprecate o non più utilizzate;
- errori dovuti a configurazioni ambientali necessarie per la corretta esecuzione ma non documentate;
- test obsoleti a causa di modifiche nel codice non accompagnate da un aggiornamento della suite.

2.3 Test di Sistema

I test di sistema erano **8 in totale**, dei quali **7 superati** e **1 fallito**. Tali test erano automatizzati con `Selenium` e progettati secondo la tecnica del *Category Partitioning*.

Nome del Test	Esito
RF1_get_smells::test_tc_GS1	Passed
RF1_get_smells::test_tc_GS_2	Passed

RF1_get_smells::test_tc_GS_pass	Passed
RF1_get_smells_by_date::test_GSBD_fail	Failed
RF1_get_smells_by_date::test_GSBD_pass	Passed
RF3_info::test_tc_IN1	Passed
RF3_info::test_tc_IN_PASS	Passed
RF4_geo_dispersion_tool::test_compute	Passed

Totale test: 8 **Passati:** 7 **Falliti:** 1

L'analisi ha evidenziato anche in questo caso delle criticità:

- alcuni test case definiti nei test frame non risultavano implementati;
- in altri casi, la suddivisione in categorie non era ottimale e presentava alcuni errori metodologici;
- la struttura di alcuni test automatizzati non rispettava fedelmente quanto specificato nella documentazione.

Ne deriva che, sebbene presenti, i test di sistema richiedevano una revisione metodologica per garantire una copertura coerente con i requisiti funzionali e i casi d'uso del sistema.

2.4 Copertura del Codice e Strategia di Reimpiego

L'analisi della **branch coverage** ha evidenziato una copertura mediamente soddisfacente nei moduli principali. Alcuni file, come `tool_selector.py` e `utils.py`, raggiungevano una copertura del 100%, mentre altri (es. `cadocs_messages.py`, `custmException.py`) risultavano trascurati o obsoleti.

I test correttamente funzionanti sono stati mantenuti e riutilizzati come **base per il testing di regressione** durante la fase di manutenzione. I test unitari e di integrazione

non funzionanti, soprattutto quelli relativi a componenti effettivamente modificati durante le *Change Request*, sono stati analizzati, corretti e integrati nella suite finale.

File	Stmt	Miss	Branch	Part	Cov.%
src\intent_handling\cadocs_intent.py	7	0	0	0	100.0
src\intent_handling\tool_selector.py	15	0	0	0	100.0
src\service\utils.py	22	0	4	0	100.0
src\service\language_handler.py	24	0	4	1	96.0
src\intent_web_service.py	42	2	8	2	92.0
src\intent_handling\intent_resolver.py	27	2	12	2	90.0
src\chatbot\intent_manager.py	23	1	8	2	90.0
src\nlu_model\CADOCS.py	35	4	0	0	89.0
src\nlu_model\model_selector.py	15	2	0	0	87.0
src\intent_handling\tool_strategy.py	6	1	0	0	83.0
src\intent_handling\tools.py	33	7	4	1	78.0
src\nlu_model\prediction_service.py	15	3	2	1	76.0
src\chatbot\cadocs_utils.py	38	22	8	1	41.0
src\service\cadocs_messages.py	125	87	66	7	27.0
src\intent_handling\custmException.py	14	14	4	0	0.0
src\oauth\oauth.py	30	30	2	0	0.0
Totale	**491**	**175**	**122**	**17**	**58.0**

Per quanto riguarda i **test di sistema**, il *Category Partitioning* è stato rivisitato ove necessario, come documentato nel *Test Design Document* e nella *Test Case Specification*. I test case mancanti o non implementati sono stati progettati ex novo e aggiunti alla suite prima del testing finale, assicurando la completa copertura delle combinazioni significative anche per i vecchi requisiti funzionali del sistema.

CAPITOLO 3

Risultati Post-Manutenzione

3.1 CR1 – Integrazione di TOAD in GUIDO

La prima Change Request ha riguardato l'integrazione del tool **TOAD** all'interno della piattaforma **GUIDO**. In particolare, è stato necessario realizzare un **wrapper RESTful** per consentire di interfacciarsi con TOAD tramite chiamate HTTP asincrone e poi è stato necessario modificare alcuni componenti di GUIDO per supportare l'integrazione del nuovo tool.

3.1.1 Test di Unità e di Integrazione

In relazione al sistema **GUIDO**, sono stati sviluppati nuovi test di unità e integrazione per coprire tutte le componenti introdotte e modificate durante l'implementazione della CR1. Alcuni dei moduli testati erano già presenti nella suite preesistente, ma i relativi test risultavano non funzionanti o incompleti: in questi casi, sono stati opportunamente corretti e riadattati. Durante questa fase sono stati anche sviluppati ulteriori test per coprire codice che non risultava già coperto a sufficienza. Tutti i test sviluppati per la componente GUIDO sono stati eseguiti con successo e non sono state riscontrate anomalie.

Nome del Test	Esito
CommunityInspectorToolTest::test_analyze_success	Passed
CommunityInspectorToolTest::test_analyze_failure	Passed
CommunityInspectorToolTest::test_status_pending	Passed
CommunityInspectorToolTest::test_status_failed	Passed
CommunityInspectorToolTest::test_status_success_result_ok	Passed
CommunityInspectorToolTest::test_status_success_result_error	Passed
CommunityInspectorToolTest::test_malformed_input	Passed
IntentResolverTest::test_community_inspector_analyze	Passed
IntentResolverTest::test_community_inspector_results	Passed
IntentWebServiceTest::test_resolve_message_intent	Passed
IntentWebServiceTest::test_resolve_job_id	Passed
IntentWebServiceTest::test_resolve_repo_analysis	Passed
TestIntentResolverIntegration::test_real_selector	Passed
TestIntentResolverIntegration::test_results_real_selector	Passed
TestToolSelectorIntegration::test_real_tools	Passed
TestToolSelectorIntegration::test_analyze	Passed
TestToolSelectorIntegration::test_results_success	Passed
TestToolsIntegration::test_community_inspector_tool_analyze_error	Passed
TestToolsIntegration::test_community_inspector_tool_analyze_success	Passed
TestToolsIntegration::test_community_inspector_tool_bad_parameters	Passed
TestToolsIntegration::test_community_inspector_tool_results_pending	Passed
TestToolsIntegration::test_community_inspector_tool_results_result_error	Passed
TestToolsIntegration::test_community_inspector_tool_results_status_error	Passed
TestToolsIntegration::test_community_inspector_tool_results_success	Passed

Totale test: 25 **Passati:** 25 **Falliti:** 0

Per quanto riguarda la componente **TOAD**, sono stati sviluppati test di unità e di integrazione dedicati esclusivamente al **wrapper** RESTful. In particolare, i test hanno verificato il corretto funzionamento degli endpoint REST, delle funzioni realizzate per la simulazione degli input e per il parsing degli output di TOAD. Anche in questo caso, tutti i test sono stati superati con esito positivo.

Nome del Test	Esito
MainTest::test_analyze_endpoint	Passed
MainTest::test_status_success	Passed
MainTest::test_status_pending_with_redis_metadata	Passed
MainTest::test_result_ready	Passed
MainTest::test_result_from_log_file	Passed
MainTest::test_result_not_found	Passed
CleanUpTest::test_clean_up_removes_all_paths	Passed
DetectToadFailureTest::test_detect_known_toad_error	Passed
DetectToadFailureTest::test_detect_known_toad_error_case_insensitive	Passed
DetectToadFailureTest::test_detect_unknown_error_returns_none	Passed
DetectToadFailureTest::test_detect_partial_match_does_not_trigger	Passed
PrepareJobDirectoryTest::test_prepare_job_directory_creates_files	Passed
ReadAllGraphsTest::test_read_all_graphs_directory_not_found	Passed
ReadAllGraphsTest::test_read_all_graphs_valid_single_file	Passed
ReadAllGraphsTest::test_read_all_graphs_file_with_missing_weight	Passed
ReadAllGraphsTest::test_read_all_graphs_multiple_files	Passed
ReadAllGraphsTest::test_read_all_graphs_parsing_error	Passed
ReadMetricsTest::test_read_metrics_file_not_exists	Passed
ReadMetricsTest::test_read_metrics_invalid_json	Passed
ReadMetricsTest::test_read_metrics_empty_sections	Passed
ReadMetricsTest::test_read_metrics_complete_file	Passed

ReadPatternsTest::test_read_patterns_file_missing	Passed
ReadPatternsTest::test_read_patterns_empty_file	Passed
ReadPatternsTest::test_read_patterns_valid_file_with_true_and_false	Passed
ReadPatternsTest::test_read_patterns_skips_unknown_keys	Passed
ReadPatternsTest::test_read_patterns_handles_read_error	Passed
UtilsTest::test_valid_date	Passed

Totale test: 27 **Passati:** 27 **Falliti:** 0

3.1.2 Test di Regressione

Come pianificato all'interno del *Test Plan*, è stato adottato un approccio **Test All** per l'esecuzione dei test di regressione. Per il sistema **GUIDO**, sono stati rieseguiti tutti i test preesistenti che risultavano funzionanti prima della manutenzione. L'esecuzione non ha evidenziato alcuna regressione: le funzionalità già presenti nel sistema non sono state compromesse dall'integrazione del nuovo tool.

Per quanto riguarda **TOAD**, durante la CR1 non sono state apportate modifiche al codice sorgente dello strumento. Di conseguenza, non si è reso necessario alcun test di regressione. Come previsto nel piano iniziale, era stata predisposta un'eventuale verifica basata sull'esecuzione comparativa dei risultati di TOAD su un insieme di repository GitHub selezionate. Tuttavia, poiché non sono stati modificati né gli algoritmi né i file core di TOAD, tale verifica non è stata necessaria.

3.2 CR2 – Aggiunta della GUI per utilizzare TOAD

La seconda Change Request ha riguardato l'estensione dell'interfaccia grafica della piattaforma **GUIDO** al fine di permettere l'utilizzo interattivo delle funzionalità TOAD integrate nella CR1. In particolare, è stata progettata e implementata una nuova sezione denominata *Community Inspector*, accessibile dal menù principale dell'applicazione, che consente agli utenti di:

- avviare una nuova analisi TOAD, specificando la repository GitHub e la data di riferimento;
- monitorare lo stato delle richieste inviate (PENDING, STARTED, FAILED, SUCCESS);
- visualizzare i risultati, ovvero i community pattern rilevati e le metriche analitiche calcolate per stabilire la loro presenza.

3.2.1 Test di Sistema

L'introduzione della nuova GUI ha permesso la definizione e l'automazione dei **test di sistema** per il nuovo requisito funzionale **RF05 – Get Community Patterns**, sfruttando il framework **Selenium** per simulare le interazioni utente end-to-end. I test sono stati progettati in coerenza con il *Test Design Document* e la *Test Case Specification*, adottando la tecnica del *Category Partitioning*. Ciascun test è stato sviluppato per coprire combinazioni significative di input, sia validi che errati, in condizioni realistiche.

I risultati dell'esecuzione mostrano che tutti i test automatizzati sono stati superati con successo, ad eccezione di uno scenario non testato per mancanza di dati adeguati. Infatti, il test **TC_GCP_6** non è stato incluso nell'esecuzione automatica, in quanto prevedeva la verifica di un caso limite: repository con almeno 100 commit, almeno 1 milestone ma **meno di 2 membri attivi**. Non è stato possibile reperire una repository pubblica con tali caratteristiche strutturali, pertanto il caso è stato escluso dalla validazione automatica.

Nome del Test	Esito
TC_GCP_1	Passed
TC_GCP_2	Passed
TC_GCP_3	Passed
TC_GCP_4	Passed
TC_GCP_5	Passed
TC_GCP_7	Passed

Totale test automatizzati eseguiti: 6 **Passati:** 6 **Falliti:** 0

3.2.2 Test di Regressione

In linea con quanto pianificato nel *Test Plan*, è stato eseguito un ciclo completo di **test di regressione** per garantire che l'introduzione della nuova interfaccia grafica non avesse effetti collaterali sulle funzionalità già presenti nel sistema. Sono stati rieseguiti con esito positivo:

- tutti i test preesistenti alla manutenzione (unità, integrazione e sistema);
- tutti i test sviluppati in occasione della **CR1**.

Unica eccezione è stata rilevata in uno dei **test di sistema preesistenti**, relativo all'interazione con il *Community Inspector*. A seguito di un'analisi più approfondita, è stato riscontrato che l'anomalia non era legata a un malfunzionamento del sistema, bensì a una variazione stilistica della GUI: durante l'implementazione della CR2 era stato modificato lo stile grafico dei pulsanti, causando il fallimento del test automatico che selezionava gli elementi tramite identificatori CSS non più validi. Dopo l'analisi, il test è stato aggiornato per riflettere la nuova struttura dell'interfaccia, ed è stato poi superato correttamente nelle esecuzioni successive.

3.3 CR3 – Estensione della GUI per la visualizzazione del grafo TOAD

La terza Change Request ha avuto l'obiettivo di estendere la GUI introdotta con la CR2, al fine di migliorare l'esperienza utente nella consultazione dei risultati restituiti dal tool TOAD. In particolare, è stata aggiunta alla sezione *Community Inspector* una nuova componente grafica che consente di visualizzare in modo interattivo e tridimensionale il **grafo delle interazioni sociali** tra i membri attivi della repository analizzata. Oltre alla visualizzazione del grafo, è stato introdotto anche un pannello riepilogativo con la **lista dei membri attivi**, ognuno dei quali è accompagnato da un collegamento ipertestuale al relativo profilo GitHub.

3.3.1 Testing

L'introduzione di queste nuove funzionalità non ha comportato la necessità di sviluppare ulteriori test, in quanto l'intervento ha interessato esclusivamente la **logica di presentazione**, il backend infatti era stato già predisposto per restituire tutte le informazioni necessarie alla costruzione del grafo in fase di implementazione della CR1.

Sono stati tuttavia aggiornati i **test di sistema** precedentemente realizzati per la CR2, in modo da includere anche la verifica della corretta visualizzazione dei risultati di TOAD, tra cui il rendering del grafo in formato 3D e la presenza della lista dei membri attivi della repository; Dopo l'aggiornamento dei test, è stata eseguita nuovamente l'intera suite di test a disposizione, comprensiva di:

- test preesistenti al ciclo di manutenzione;
- test sviluppati per la CR1 (integrazione TOAD);
- test automatizzati di sistema introdotti con la CR2 (aggiornati dopo la CR3).

Tutti i test sono stati superati con esito positivo, a conferma della piena compatibilità e corretto funzionamento della nuova funzionalità all'interno del sistema esistente.

3.4 CR4 – Dockerizzazione di TOAD

La quarta Change Request ha riguardato la **dockerizzazione del tool TOAD**, comprensivo del relativo wrapper REST sviluppato nella CR1, e la sua integrazione all'interno dell'ecosistema **GUIDO**, già interamente contenitoreizzato tramite `docker-compose`. L'obiettivo dell'attività era quello di rendere TOAD un servizio deployabile insieme a GUIDO in modo automatico, portabile e replicabile su diverse macchine, migliorando la manutenibilità e l'efficienza dell'intero sistema.

3.4.1 Testing

Anche per questa CR non è stato necessario sviluppare nuovi test. L'attività di verifica si è concentrata esclusivamente su un **testing di regressione**, finalizzato ad assicurare che l'introduzione del container TOAD non compromettesse il funzionamento complessivo del sistema. In particolare:

- **Testing TOAD:** poiché alcuni file interni sono stati modificati per rimuovere percorsi assoluti, come previsto nel *Test Plan*, è stato eseguito un confronto tra i risultati ottenuti prima e dopo la modifica su un insieme di repository GitHub selezionate. L'analisi ha confermato che l'output del tool è rimasto coerente, garantendo con un certo livello di fiducia l'assenza di regressioni funzionali.
- **Testing GUIDO:** sono stati rieseguiti tutti i test realizzati nelle precedenti CR (unitari, di integrazione e di sistema), oltre ai test preesistenti al ciclo di manutenzione. Tutti i test sono stati superati con esito positivo.

È importante evidenziare che, nelle CR precedenti, i **test di sistema** erano stati condotti con GUIDO e frontend eseguiti all'interno di contenitori Docker, ma con TOAD avviato localmente al di fuori dell'ambiente containerizzato. La CR4 ha finalmente permesso l'esecuzione completa dell'intero stack in ambiente Docker, rendendo possibile una validazione end-to-end all'interno dell'ecosistema finale. Tutti i test di sistema sono stati rieseguiti in questo nuovo contesto e hanno restituito esito positivo, confermando la **piena integrazione e il corretto funzionamento** dell'architettura containerizzata.

3.5 Risultati Finali

Al termine del ciclo di manutenzione evolutiva del sistema GUIDO, la suite di test è stata significativamente ampliata, corretta e consolidata. In particolare:

- sono stati aggiunti test di unità, di integrazione e di sistema per coprire le funzionalità aggiunte con le CR;
- sono stati corretti tutti i test unitari e di integrazione non funzionanti;
- sono stati integrati nuovi test per coprire funzionalità precedentemente non testate o testate in modo parziale;
- è stato rivisitato il *Category Partitioning* e sono stati implementati i test di sistema mancanti, precedentemente identificati durante la fase di test pre-manutenzione.

L'esecuzione finale della suite ha prodotto risultati pienamente soddisfacenti e **conformi ai criteri di accettazione** definiti all'interno del *Test Plan*. Di seguito si riporta un riepilogo sintetico del testing finale:

- **Totale test eseguiti**: 117
- **Test superati**: 117
- **Test falliti**: 0
- **Tempo totale di esecuzione**: 17 minuti e 13 secondi
- **Ambiente di test**: Python 3.10.0, pytest 8.2.0, Windows

Inoltre, come definito nel piano iniziale, è stato monitorato il valore della **branch coverage** al fine di garantire un'adeguata copertura del codice sorgente. Di seguito viene riportata la tabella riepilogativa (aggiornabile) con le statistiche di coverage per ciascun modulo del sistema GUIDO:

File	Stmt	Miss	Branch	Part	Cov.%
src\intent_handling\cadocs_intent.py	7	0	0	0	100.0
src\intent_handling\tool_selector.py	15	0	0	0	100.0
src\service\utils.py	22	0	4	0	100.0
src\service\language_handler.py	24	0	4	1	96.0
src\intent_web_service.py	42	2	8	2	94.0
src\intent_handling\intent_resolver.py	27	2	12	2	96.0
src\chatbot\intent_manager.py	23	1	8	2	90.0
src\nlu_model\CADOCS.py	35	4	0	0	89.0
src\nlu_model\model_selector.py	15	2	0	0	87.0
src\intent_handling\tool_strategy.py	6	1	0	0	83.0
src\intent_handling\tools.py	33	7	4	1	100.0
src\nlu_model\prediction_service.py	15	3	2	1	76.0
src\chatbot\cadocs_utils.py	38	22	8	1	41.0
src\service\cadocs_messages.py	125	87	66	7	90.0
src\intent_handling\custmException.py	14	14	4	0	0.0
src\oauth\oauth.py	30	30	2	0	0.0
Totale	**514**	**85**	**138**	**22**	**82.0**

Sebbene siano ancora presenti moduli marginali o non più utlizzati con coverage molto basso, l'obiettivo definito di mantenere una **branch coverage minima del 75%** è stato raggiunto con successo, migliorando la coverage della precedente test suite da 58% a 82%. Pertanto, considerando gli ottimi risultati ottenuti, si può concludere che la fase di testing post-manutenzione sia stata superata positivamente, garantendo affidabilità, stabilità e qualità del sistema aggiornato.