Manuale Utente PyUCC

Table of Contents

Versione Documento: 1.0
Applicazione: PyUCC (Python Unified Code Counter)

---

1. Introduzione

PyUCC è uno strumento avanzato per l'analisi statica del codice sorgente. Il suo obiettivo principale è fornire metriche quantitative sullo sviluppo software e, soprattutto, tracciare l'evoluzione del codice nel tempo attraverso un potente sistema di *Differing* (confronto).

A cosa serve?

1. **Contare:** Sapere quante righe di codice, commenti e righe vuote compongono il tuo progetto.
2. **Misurare:** Calcolare la complessità e la manutenibilità del software.
3. **Confrontare:** Capire esattamente cosa è cambiato tra due versioni (file aggiunti, rimossi, modificati e come la complessità è variata).

---

2. Concetti Fondamentali

Prima di iniziare, è utile comprendere i termini chiave utilizzati nell'applicazione.

2.1 Baseline

Una **Baseline** è una "fotografia" istantanea del tuo progetto in un preciso momento.
* Quando crei una baseline, PyUCC salva una copia dei file e calcola tutte le metriche.
* Le baseline servono come punto di riferimento (o "pietra di paragone") per i confronti futuri.

2.2 Metriche Supportate

- **SLOC (Source Lines of Code):**
  - *Physical Lines:* Totale righe del file.
  - *Code Lines:* Righe che contengono codice eseguibile.
  - *Comment Lines:* Righe di documentazione.
  - *Blank Lines:* Righe vuote (spesso usate per formattazione).
- **Complessità Ciclomatica (CC):** Misura quanto è complesso il flusso di controllo (quanti `if`, `for`, `while`, ecc.). Più è bassa, meglio è.
- **Maintainability Index (MI):** Un indice da 0 a 100 che stima quanto il codice sia facile da mantenere. Più è alto, meglio è (sopra 85 è ottimo, sotto 65 è problematico).

2.3 Profilo

Un **Profilo** è una configurazione salvata che dice a PyUCC:
* Quali cartelle analizzare.
* Quali linguaggi includere (es. solo Python e C++).
* Cosa ignorare (es. cartelle `venv`, `build`, file temporanei).

---

3. Interfaccia Utente (GUI)

L'interfaccia è divisa in zone funzionali per mantenere il flusso di lavoro ordinato.

1. **Top Bar (Barra Superiore):**
   - Selezione del **Profilo**.
   - Accesso alle impostazioni (**Settings**) e gestione profili (**Manage**).
2. **Actions Bar (Azioni):** I pulsanti principali per avviare le operazioni (`Scan`, `Countings`, `Metrics`, `Differing`).
3. **Progress Area:** Barra di avanzamento e contatore dei file elaborati.
4. **Results Table:** La grande tabella centrale dove appaiono i dati.
5. **Log & Status:** In basso, un pannello di log per vedere cosa sta succedendo e una barra di stato con il monitoraggio risorse (CPU/RAM).

---

4. Guida Passo-Passo

4.1 File di Configurazione e Posizione

PyUCC salva tutte le sue configurazioni in file JSON nella stessa cartella dell'eseguibile, rendendo il software completamente portatile.

File di configurazione:
- **`profiles.json`**: Contiene tutti i profili di analisi salvati (percorsi, filtri, impostazioni).
- **`settings.json`**: Contiene le impostazioni globali dell'applicazione (cartella baseline, duplicati, ecc.).

Posizione:
- **Esecuzione da eseguibile**: I file si trovano nella stessa cartella di `pyucc.exe`
- **Esecuzione da sorgente**: I file si trovano nella cartella radice del progetto (accanto a `README.md`)

Questi file vengono creati automaticamente al primo avvio dell'applicazione. Se non esistono, PyUCC utilizza valori predefiniti.

4.2 Impostazioni dell'Applicazione (settings.json)

Per configurare correttamente PyUCC al primo avvio, è importante comprendere le impostazioni disponibili:

`baseline_dir` (Cartella Baseline)
- **Cosa fa**: Specifica dove vengono salvate le baseline (snapshot del progetto per i confronti).
- **Predefinito**: `./baseline` (nella cartella dell'eseguibile)
- **Come configurare**:
- Vai su □□ **Settings** → **Baseline Directory**
- Scegli una cartella dedicata (es. `C:\MyProjects\baselines`)
- **Quando configurare**: Al primo avvio, soprattutto se hai più progetti da analizzare
- **Consiglio**: Usa una cartella separata per ogni progetto o una cartella centrale con sottocartelle

`max_keep` (Massime Baseline da Mantenere)
- **Cosa fa**: Numero massimo di baseline da conservare per ogni profilo.
- **Predefinito**: 10
- **Come configurare**: Vai su □□ **Settings** → **Max Baseline To Keep**
- **Quando configurare**: Se lavori su progetti grandi o hai poco spazio disco
- **Consiglio**: 5-10 per progetti piccoli, 20-30 per progetti grandi con molte release

`zip_baselines` (Comprimi Baseline)
- **Cosa fa**: Comprime automaticamente le baseline per risparmiare spazio.
- **Predefinito**: `true`
- **Come configurare**: Vai su □□ **Settings** → checkbox **Zip Baselines**
- **Quando configurare**: Lascia abilitato a meno che non abbia problemi di performance
- **Consiglio**: Tienilo attivo, riduce l'uso del disco del 70-90%

Impostazioni Duplicati (`duplicates_settings`)
- **`threshold`** (Soglia di Similarità Fuzzy): 0.0-1.0, default 0.85
- Valori più alti = meno falsi positivi, ma potrebbero perdere duplicati reali
- Valori più bassi = più duplicati trovati, ma più falsi positivi
- **`k`** (K-gram Size): 3-10, default 5
- K-gram più grandi = meno sensibile a piccole modifiche
- K-gram più piccoli = più sensibile a piccole modifiche
- **`window`** (Finestra Winnowing): 3-20, default 4
- Finestre più grandi = meno hash da confrontare (più veloce ma meno preciso)
- Finestre più piccole = più hash da confrontare (più lento ma più preciso)

Come configurare i duplicati:
1. Vai su □□ **Settings** → **Duplicates Detection**
2. Regola i parametri in base alle tue esigenze
3. Fai test con diversi valori sul tuo codice

Raccomandazioni per il primo avvio:
1. **Configura `baseline_dir`** prima di creare la prima baseline
2. Lascia gli altri valori predefiniti e regolali solo se necessario
3. Le impostazioni si applicano a tutti i profili

4.3 Configurazione Profilo

Non appena apri PyUCC, la prima cosa da fare è dire al programma *cosa* analizzare.

1. Clicca su □□ **Manage...** in alto.
2. Clicca su □ **New** per pulire i campi.
3. Inserisci un **Nome** per il profilo (es. "Mio Progetto Backend").
4. Nella sezione **Paths**, usa **Add Folder** per selezionare la cartella radice del tuo codice.
5. Nella sezione **Filter Extensions**, seleziona i linguaggi che ti interessano (es. Python, Java).
6. Nella casella **Ignore patterns**, puoi lasciare i default (che escludono già `.git`, `__pycache__`, ecc.).
7. Clicca □ **Save**.

4.4 Analisi Semplice (Scan, Countings, Metrics)

Se vuoi solo analizzare lo stato attuale senza confronti:

- □ **Scan:** Verifica semplicemente quali file vengono trovati in base ai filtri del profilo. Utile per controllare se stai includendo i file giusti.
- □ **Countings:** Analizza ogni file e ti dice quante righe di codice, commenti e vuote ci sono.
- □ **Metrics:** Calcola la complessità ciclomatica e l'indice di manutenibilità per ogni file.

> **Tip:** Puoi fare doppio click su un file nella tabella dei risultati per aprirlo nel **File Viewer** integrato, che evidenzia la sintassi e mostra una minimappa colorata (blu=codice, verde=commenti).

4.5 Il Flusso di Lavoro "Differing" (Confronto)

Questa è la funzione più potente di PyUCC.

Passo A: Creare la prima Baseline
1. Seleziona il tuo profilo.
2. Clicca su □ **Differing**.
3. Se è la prima volta che analizzi questo progetto, PyUCC ti avviserà: *"No baseline found"*.
4. Conferma la creazione. PyUCC scatterà una "foto" del progetto (Baseline).

Passo B: Lavorare sul codice
Ora puoi chiudere PyUCC e lavorare al tuo codice (modificare file, aggiungerne, cancellarne).

Passo C: Confrontare
1. Riapri PyUCC e seleziona lo stesso profilo.
2. Clicca su □ **Differing**.
3. Questa volta, PyUCC vedrà che esiste una Baseline precedente e ti chiederà con quale confrontare (se ne hai più di una).
4. Il risultato sarà una tabella con colori specifici:
* **Verde:** File aggiunti o metriche migliorate.
* **Rosso:** File cancellati o metriche peggiorate (es. complessità aumentata).
* **Giallo/Arancio:** File modificati.
* **Colonne Δ (Delta):** Mostrano la differenza numerica (es. `+50` righe di codice, `-2` complessità).

> **Visualizzatore Differenze:** Se fai doppio click su una riga nel risultato del Differing, si aprirà una finestra che mostra i due file affiancati, evidenziando esattamente le righe cambiate.

---

5. Casi d'Uso Esemplificativi

Caso 1: Refactoring del codice

- **Obiettivo:** Vuoi pulire il codice e assicurarti di non aver aumentato la complessità.
- **Azione:** Fai una Baseline prima di iniziare. Fai il refactoring. Esegui il *Differing*.
- **Verifica:** Controlla la colonna **Δ avg_cc**. Se è negativa (es. `-0.5`), ottimo! Hai ridotto la complessità. Se la colonna **Δ comment_lines** è positiva, hai migliorato la documentazione.

Caso 2: Code Review

- **Obiettivo:** Un collega ha aggiunto una nuova feature. Cosa è cambiato?
- **Azione:** Esegui il *Differing* rispetto alla versione master precedente.
- **Verifica:** Ordina per "Status". Vedi subito i file **Added** (A) e **Modified** (M). Apri il Diff Viewer sui file modificati per vedere le righe esatte.

---

6. Filosofia di Sviluppo (Per Sviluppatori)

PyUCC è stato scritto seguendo principi di ingegneria del software rigorosi, che si riflettono nella stabilità dell'uso.

6.1 Codice Pulito e Standard PEP8

Il codice rispetta lo standard **PEP8** di Python. Questo garantisce che, se un giorno volessi estendere il tool o scriverne degli script di automazione usando i moduli `core`, troveresti un codice leggibile, standardizzato e prevedibile.

6.2 Separazione delle Responsabilità (SoC)

L'applicazione è divisa nettamente in due parti:
1. **Core (`pyucc.core`):** Contiene la logica pura (scansione, calcolo metriche, algoritmi di diff). Non sa nulla dell'interfaccia grafica.
2. **GUI (`pyucc.gui`):** Gestisce solo la visualizzazione.
Filosofia: Questo permette di cambiare la grafica senza rompere la logica, o usare la logica da riga di comando senza avviare la grafica.

6.3 Non-Blocking UI (Worker Manager)

Hai notato che l'interfaccia non si blocca mai, anche analizzando migliaia di file?
Questo è grazie al **WorkerManager**. Tutte le operazioni pesanti vengono eseguite in thread separati (background). L'interfaccia grafica riceve aggiornamenti tramite una coda sicura (`queue`).
* **Per l'utente significa:** Puoi sempre premere "Cancel" se un'operazione è troppo lunga.

6.4 Algoritmo di Matching Intelligente (Gale-Shapley)

Nel *Differing*, PyUCC non guarda solo se il nome del file è identico. Usa un algoritmo ispirato al "problema del matrimonio stabile" (Gale-Shapley) combinato con la distanza di Levenshtein sui percorsi.
* **Filosofia:** Se sposti un file da una cartella all'altra, il sistema cerca di capire che è lo stesso file spostato, invece di segnarne uno come "Deleted" e uno come "Added".

6.5 Determinismo

Il sistema usa l'hashing (SHA1/MD5) del contenuto dei file per ottimizzare i calcoli (caching) e per determinare se un file è *veramente* cambiato, ignorando la data di modifica del file system se il contenuto è identico.

---

7. Risoluzione Problemi Comuni

- **Il programma non trova file:** Controlla nel Profile Manager se l'estensione del file è nella lista dei linguaggi o se la cartella è inclusa negli "Ignore patterns".
- **Lentezza estrema:** Se hai incluso cartelle con migliaia di file piccoli non di codice (es. `node_modules` o cartelle di immagini), aggiungile agli "Ignore patterns".
- **Diff Viewer vuoto:** Assicurati che i file sorgente esistano ancora sul disco. Se hai cancellato la cartella del progetto dopo aver fatto la baseline, il viewer non potrà mostrare il file corrente.

---

8. Nuove Funzionalità (Da v1.0)

Questa release introduce funzionalità che migliorano l'analisi della qualità del codice, la riproducibilità delle baseline e la ricerca di duplicazioni nel codice. Di seguito una descrizione sintetica delle novità e come usarle.

8.1 Rilevamento Duplicati (GUI + CLI)

- **Cosa fa:** Individua duplicati esatti e fuzzy all'interno del progetto. I duplicati esatti sono individuati tramite hashing del contenuto (SHA1). I duplicati fuzzy usano fingerprinting a k-gram con una fase di winnowing e una misura di similarità Jaccard per identificare coppie simili.
- **Parametri:** `k` (dimensione dei k-gram), `window` (finestra di winnowing) e `threshold` (soglia di similarità in percentuale). I valori di default sono bilanciati per precisione/recall ma possono essere modificati dall'utente.
- **Esecuzione (GUI):** Usa il nuovo pulsante **Duplicates** nella barra Azioni (posizionato prima del pulsante Differ). Una finestra di dialogo permette di scegliere estensioni, soglia e parametri di fingerprinting. Le impostazioni sono persistenti.
- **Esecuzione (CLI):** `python -m pyucc duplicates <path> --threshold 5.0 --ext .py .c` stampa in output JSON i duplicati trovati.
- **Esportazione:** Risultati esportabili in `CSV` e in un report testuale in stile UCC inserito nella cartella baseline (quando eseguito durante la creazione della baseline).

8.2 Report UCC-style per Duplicati e Differenze

- **Tabella compatta in stile UCC:** Il differ ora può generare una tabella compatta simile all'output UCC, con colonne Δ (delta) aggiuntive: `ΔCode`, `ΔComm`, `ΔBlank`, `ΔFunc`, `ΔAvgCC`, `ΔMI`, per vedere rapidamente le variazioni numeriche.
- **Report duplicati:** Viene creato un file testuale `duplicates_report.txt` (se richiesto) che elenca i gruppi di duplicati con la similarità percentuale e i parametri usati. Le baseline salvano i parametri per garantire la riproducibilità.

Esempio (snippet compatto in stile UCC):

```
File                                  Code   Comm  Blank  Func  AvgCC   MI   ΔCode  ΔComm  ΔBlank  ΔFunc  ΔAvgCC  ΔMI
-------------------------------------------------------------------------------------------------------------------
src/module/a.py                        120     10      8      5     2.3    78   +10     -1      0       +0      -0.1    +2
src/module/b_copy.py                   118      8     10      5     2.4    76   -2      -2      +2      0       +0.1    -2
```

8.3 Scanner e Migliorie alle Baseline

- **Scansione centralizzata:** Lo `scanner` è il fornitore canonico della lista file. Moduli pesanti (Differ, Duplicates) possono ricevere la `file_list` prodotta dallo scanner per evitare ricerche ripetute e garantire lo stesso filtro.
- **Normalizzazione dei pattern di ignore:** Voci di ignore come `.bak` vengono normalizzate in `*.bak` e il matching è case-insensitive per default; questo evita di includere file di backup nelle baseline.
- **Riproducibilità delle baseline:** Le baseline memorizzano i parametri usati per la ricerca duplicati e la lista dei file snapshot. Se

in seguito viene installato `lizard`, PyUCC tenta di rieseguire l'analisi per ottenere informazioni sulle funzioni anche nelle baseline create in precedenza.

8.4 Note sulle Dipendenze

- Le metriche a livello di funzione (numero di funzioni, CC per funzione) richiedono `lizard`. Se `lizard` non è installato, PyUCC produrrà comunque SLOC e metriche di base, ma i dettagli per funzione potrebbero mancare. La creazione della baseline registra questo stato e tenterà una rianalisi se `lizard` diventa disponibile.

---

9. Rilevamento Duplicati: Algoritmi e Dettagli Tecnici

Questa sezione fornisce una comprensione più approfondita di come PyUCC identifica il codice duplicato, cosa fanno gli algoritmi e come interpretare i risultati.

9.1 Rilevamento Duplicati Esatti

Come funziona:
- PyUCC normalizza ciascun file (rimuove spazi iniziali/finali da ogni riga, converte in minuscolo opzionalmente).
- Calcola un hash SHA1 del contenuto normalizzato.
- I file con hash identici sono considerati duplicati esatti.

Caso d'uso: Trovare file che sono stati copiati e incollati senza o con modifiche minime (es. `utils.py` e `utils_backup.py`).

Cosa vedrai:
- Nella tabella GUI: coppie di file contrassegnate come duplicati "esatti" con similarità al 100%.
- Nel report: elencate nella sezione "Duplicati esatti".

9.2 Rilevamento Duplicati Fuzzy (Avanzato)

Il rilevamento fuzzy identifica file che sono *simili* ma non identici. È utile per trovare:
- Codice che è stato copiato e poi leggermente modificato.
- Moduli rifatti che condividono grandi blocchi di logica.
- Branch sperimentali o "quasi-duplicati" che dovrebbero essere fusi.

Panoramica dell'Algoritmo:

1. **Hashing K-gram (Rolling Hash con Rabin-Karp):**
2. Ogni file è diviso in sequenze sovrapposte di `k` righe consecutive (k-gram).
3. Viene calcolato un rolling hash (hash polinomiale Rabin-Karp) per ogni k-gram.
4. Questo produce un grande insieme di valori hash che rappresentano tutti i k-gram del file.
5. **Winnowing (Selezione delle Impronte Digitali):**
6. Per ridurre il numero di hash (e migliorare le prestazioni), PyUCC applica una tecnica di "winnowing".
7. Una finestra scorrevole di dimensione `w` si sposta sulla sequenza di hash.
8. In ogni finestra, viene selezionato il valore hash minimo come impronta digitale.
9. Questo crea un insieme compatto di impronte rappresentative per il file.
10. **Proprietà chiave:** Se due file condividono una sottostringa di almeno `k + w - 1` righe, condivideranno almeno un'impronta digitale.
11. **Indice Invertito:**
12. Tutte le impronte digitali di tutti i file vengono memorizzate in un indice invertito: `{impronta -> [lista di file che la contengono]}`.
13. Questo permette una ricerca veloce di quali file condividono impronte.
14. **Similarità di Jaccard:**
15. Per ogni coppia di file che condividono almeno un'impronta digitale, PyUCC calcola la similarità di Jaccard: `Jaccard(A, B) = |A ∩ B| / |A ∪ B|`
16. Dove A e B sono gli insiemi di impronte digitali per i due file.
17. Se il punteggio Jaccard è sopra la soglia (default: 0.85, ovvero 85% di similarità), la coppia viene segnalata come duplicato fuzzy.
18. **Calcolo Percentuale di Modifica:**
19. PyUCC stima anche la percentuale di righe che differiscono tra i due file.
20. Se `pct_change <= threshold` (es. ≤5%), i file sono considerati duplicati.

Parametri regolabili:

- **`k` (dimensione k-gram):** Numero di righe consecutive in ogni k-gram. Default: 25.
- `k` maggiore → meno falsi positivi, ma può perdere duplicati piccoli.
- `k` minore → più sensibile, ma può produrre falsi positivi.
- **`window` (dimensione finestra winnowing):** Dimensione della finestra per selezionare le impronte. Default: 4.
- Finestra maggiore → meno impronte, elaborazione più veloce, ma può perdere alcune corrispondenze.
- Finestra minore → più impronte, più lento, ma più accurato.
- **`threshold` (soglia percentuale di modifica):** Differenza massima consentita (in %) per considerare ancora due file come duplicati. Default: 5.0%.
- Soglia inferiore → corrispondenza più stretta (solo file molto simili).
- Soglia superiore → più permissiva (cattura file con più differenze).

Impostazioni consigliate:

Caso d'Uso	k	window	threshold
Ricerca duplicati stretta (solo file quasi identici)	30	5	3.0%
Bilanciata (default)	25	4	5.0%
Corrispondenza lassa (cattura codice rifatto)	20	3	10.0%
Molto aggressiva (sperimentale)	15	2	15.0%

9.3 Interpretare i Report sui Duplicati

Colonne Tabella GUI:

- **File A / File B:** I due file confrontati.
- **Match Type:** "exact" o "fuzzy".
- **Similarity (%):** Per corrispondenze fuzzy, il punteggio di similarità Jaccard (0-100%).
- **Pct Change (%):** Percentuale stimata di righe che differiscono.

Report Testuale (duplicates_report.txt):

Il report è diviso in due sezioni:

1. **Duplicati Esatti:**
   ```
   Exact duplicates: 3

src/utils.py <=> src/backup/utils_old.py
src/module/helper.py <=> src/module/helper - Copy.py
```

1. **Duplicati Fuzzy:**
   ```
   Fuzzy duplicates (threshold): 5

src/processor.py <=> src/processor_v2.py
Similarity: 92.5% | Pct Change: 3.2%

src/core/engine.py <=> src/experimental/engine_new.py
Similarity: 88.0% | Pct Change: 4.8%
```

Interpretazione:

- **Alta similarità (>95%):** Forti candidati per la deduplicazione. Considera di mantenere solo una versione o di fonderle.
- **Media similarità (85-95%):** Rivedi manualmente. Può indicare codice rifatto o variazioni intenzionali.
- **Violazioni della soglia:** I file che superano la soglia `pct_change` non appariranno nel report, anche se condividono alcune impronte.

---

10. Leggere e Interpretare i Report Differ

La funzionalità Differ produce diversi tipi di output. Comprendere ciascuno aiuta a tracciare l'evoluzione del codice con precisione.

10.1 Tabella Compatta in Stile UCC

Quando esegui *Differing*, PyUCC genera una tabella di riepilogo compatta simile allo strumento UCC originale:

Esempio:

```
File                                    Code   Comm  Blank  Func  AvgCC   MI   ΔCode  ΔComm  ΔBlank  ΔFunc  ΔAvgCC  ΔMI
-------------------------------------------------------------------------------------------------------------------
src/module/a.py                          120     10      8      5    2.3    78   +10     -1      0       +0     -0.1    +2
src/module/b.py                          118      8     10      5    2.4    76   -2      -2      +2      0      +0.1   -2
src/new_feature.py                        45      5      3      2    1.8    82   +45     +5      +3      +2     +1.8   +82
src/old_code.py                           --     --     --     --     --    --   -30     -5      -2      -1     -2.1   -75
```

Significato delle Colonne:

Colonna	Significato
File	Percorso relativo del file
Code	Numero corrente di righe di codice
Comm	Numero corrente di righe di commenti
Blank	Numero corrente di righe vuote
Func	Numero di funzioni rilevate (richiede `lizard`)
AvgCC	Complessità ciclomatica media per funzione
MI	Indice di Manutenibilità (0-100, più alto è meglio)
ΔCode	Variazione nelle righe di codice (corrente - baseline)
ΔComm	Variazione nelle righe di commenti
ΔBlank	Variazione nelle righe vuote
ΔFunc	Variazione nel conteggio funzioni
ΔAvgCC	Variazione nella complessità ciclomatica media
ΔMI	Variazione nell'indice di manutenibilità

Codifica Colori (GUI):

- **Righe verdi:** File nuovi (Aggiunti) o metriche migliorate (es. ΔAvgCC < 0, ΔMI > 0).
- **Righe rosse:** File eliminati o metriche peggiorate (es. ΔAvgCC > 0, ΔMI < 0).
- **Righe gialle/arancioni:** File modificati con cambiamenti misti.
- **Righe grigie:** File non modificati (identici alla baseline).

Cosa cercare:

- **ΔCode >> 0:** Espansione significativa del codice. È giustificata da nuove funzionalità?
- **ΔComm < 0:** Documentazione diminuita. Considera di aggiungere più commenti.
- **ΔAvgCC > 0:** Complessità aumentata. Può indicare necessità di refactoring.
- **ΔMI < 0:** Manutenibilità peggiorata. Rivedi le modifiche.
- **Nuovi file con alto AvgCC:** Il nuovo codice è già complesso. Segnala per revisione.

10.2 Report Diff Dettagliato (diff_report.txt)

Un report testuale viene salvato nella cartella baseline:

Struttura:

```
PyUCC Baseline Comparison Report
================================
Baseline ID: MyProject__20251205T143022_local
Snapshot timestamp: 2025-12-05 14:30:22

Summary:
  New files: 3
  Deleted files: 1
  Modified files: 12
  Unchanged files: 45

Metric Changes:
  Total Code Lines: +150
  Total Comments: -5
  Average CC: +0.2 (slight increase in complexity)
  Average MI: -1.5 (slight decrease in maintainability)

[Tabella compatta in stile UCC qui]

Legend:
  A = Added file
  D = Deleted file
  M = Modified file
  U = Unchanged file
  ...
```

10.3 Esportazioni CSV

Puoi esportare qualsiasi tabella dei risultati in CSV per ulteriori analisi in Excel, pandas o strumenti BI.

Le colonne includono:
- Percorso file
- Tutte le metriche SLOC (codice, commenti, righe vuote)
- Metriche di complessità (CC, MI, conteggio funzioni)
- Delta (se da un'operazione Differ)
- Flag di stato (A/D/M/U)

Casi d'uso:
- Analisi dei trend su più baseline.
- Generazione di grafici (es. complessità nel tempo).
- Integrazione in gate di qualità CI/CD.

---

11. Casi d'Uso Pratici e Workflow

Caso d'Uso 1: Rilevare Codice Copiato Prima della Code Review

Scenario: Il tuo team sta sviluppando un nuovo modulo. Sospetti che alcuni sviluppatori abbiano copiato e incollato codice esistente invece di fare refactoring.

Workflow:
1. Crea un profilo per il tuo progetto.
2. Clicca sul pulsante **Duplicates**.
3. Imposta threshold a 5% (stretto).
4. Rivedi la tabella dei risultati.
5. Per ogni coppia di duplicati fuzzy:
- Fai doppio click per aprire entrambi i file nel visualizzatore diff (se implementato).
- Valuta se la duplicazione è intenzionale o dovrebbe essere rifatta in un'utility condivisa.
6. Esporta in CSV e condividi con il team per discussione.

Risultato atteso: Identifichi 3-5 file quasi duplicati e crei ticket per consolidarli.

Caso d'Uso 2: Tracciare la Complessità Durante uno Sprint di Refactoring

Scenario: Il tuo team pianifica uno sprint di refactoring di 2 settimane per ridurre il debito tecnico.

Workflow:
1. **Prima dello sprint:** Crea una baseline ("Pre-Refactor").
- Clicca **Differing** → Crea baseline.
- Nominala "PreRefactor_Sprint5".
2. **Durante lo sprint:** Gli sviluppatori rifanno il codice, estraggono funzioni, aggiungono commenti.
3. **Dopo lo sprint:** Esegui **Differing** contro la baseline.
4. Rivedi la tabella compatta:
- Controlla ΔAvgCC: Dovrebbe essere negativo (complessità ridotta).
- Controlla ΔMI: Dovrebbe essere positivo (manutenibilità migliorata).
- Controlla ΔComm: Dovrebbe essere positivo (più documentazione).
5. Genera un report diff e allegalo alla retrospettiva dello sprint.

Risultato atteso: Prova quantitativa che il refactoring ha funzionato: "Abbiamo ridotto il CC medio del 15% e aumentato MI di 8 punti."

Caso d'Uso 3: Assicurare che Nuove Funzionalità Non Degradino la Qualità

Scenario: Stai aggiungendo una nuova funzionalità a una codebase matura. Vuoi assicurarti che il nuovo codice non introduca complessità eccessiva.

Workflow:
1. Crea una baseline prima di iniziare lo sviluppo della funzionalità.
2. Sviluppa la funzionalità in un branch.
3. Prima del merge su main:
- Esegui **Differing** per confrontare lo stato corrente vs. baseline.
- Filtra per nuovi file (status = "A").
- Controlla AvgCC e MI dei nuovi file.
- Se AvgCC > 5 o MI < 70, segnala per refactoring prima del merge.
4. Usa **Duplicates** per assicurarti che il nuovo codice non duplichi utility esistenti.

Risultato atteso: Il codice della nuova funzionalità supera i gate di qualità prima del merge.

Caso d'Uso 4: Generare Report di Conformità per Audit

Scenario: La tua organizzazione richiede audit periodici sulla qualità del codice.

Workflow:
1. Crea baseline mensili (es. "Audit_2025_01", "Audit_2025_02", ...).
2. Ogni baseline genera automaticamente:
- `countings_report.txt`
- `metrics_report.txt`
- `duplicates_report.txt`
3. Archivia questi report in una cartella di conformità.
4. Per l'audit, fornisci:
- Trend di SLOC totale nel tempo.
- Trend di CC e MI medi.
- Numero di duplicati rilevati e risolti ogni mese.

Risultato atteso: Gli auditor vedono un miglioramento misurabile nelle metriche di qualità del codice nel tempo.

Caso d'Uso 5: Onboarding Nuovi Sviluppatori con Metriche del Codice

Scenario: Un nuovo sviluppatore si unisce al team e ha bisogno di comprendere la codebase.

Workflow:
1. Esegui **Metrics** sull'intera codebase.
2. Esporta in CSV.
3. Ordina per AvgCC (decrescente) per identificare i moduli più complessi.
4. Condividi l'elenco con il nuovo sviluppatore:
- "Questi 5 file hanno la complessità più alta. Fai particolare attenzione quando li modifichi."
- "Questi moduli hanno MI basso. Sono candidati per refactoring—buoni esercizi di apprendimento."
5. Usa **Duplicates** per mostrare quali parti del codice hanno ridondanza (spiega perché).

Risultato atteso: Il nuovo sviluppatore comprende più velocemente i punti critici e i problemi di qualità del codice.

---

12. Suggerimenti per un Uso Efficace

12.1 Gestione Profili

- **Crea profili separati** per diversi sottoprogetti o componenti.
- Usa **pattern di ignore** in modo aggressivo per escludere:
- `node_modules`, `venv`, `.venv`
- Output di build (`build/`, `dist/`, `bin/`)
- Codice generato
- Fixture di test o dati mock

12.2 Strategia Baseline

- **Convenzione di denominazione:** Usa nomi descrittivi con date o tag di versione:
- `Release_v1.2.0_20251201`
- `PreRefactor_Sprint10`
- `BeforeMerge_FeatureX`
- **Frequenza:** Crea baseline ai traguardi chiave:
- Fine di ogni sprint
- Prima/dopo refactoring importanti
- Prima dei rilasci
- **Ritenzione:** Mantieni almeno 3-5 baseline recenti. Archivia quelle più vecchie.

12.3 Interpretare le Metriche

Complessità Ciclomatica (CC):
- **1-5:** Semplice, basso rischio.
- **6-10:** Complessità moderata, accettabile.
- **11-20:** Alta complessità, revisione raccomandata.
- **21+:** Complessità molto alta, refactoring fortemente raccomandato.

Indice di Manutenibilità (MI):
- **85-100:** Altamente manutenibile (zona verde).
- **70-84:** Moderatamente manutenibile (zona gialla).
- **Sotto 70:** Bassa manutenibilità (zona rossa), necessita attenzione.

12.4 Best Practice per il Rilevamento Duplicati

- Inizia con **parametri di default** (k=25, window=4, threshold=5%).
- Se ottieni troppi falsi positivi, **aumenta k** o **diminuisci threshold**.
- Se sospetti che i duplicati vengano persi, **diminuisci k** o **aumenta threshold**.
- **Rivedi sempre manualmente i duplicati fuzzy**—non tutte le similarità sono negative (es. implementazioni di interfacce).

---

13. Risoluzione Problemi e FAQ

D: Il rilevamento duplicati è lento su grandi codebase.

R:
- Usa i filtri del profilo per limitare i tipi di file analizzati.
- Aumenta `k` e `window` per ridurre il numero di impronte elaborate.
- Escludi file auto-generati di grandi dimensioni o fixture di test.

D: Perché alcuni file mancano di metriche a livello di funzione?

R:
- L'analisi a livello di funzione richiede `lizard`. Installalo: `pip install lizard`.
- Alcuni linguaggi potrebbero non essere completamente supportati da `lizard`.

D: Differ mostra file come "Modified" ma non li ho modificati.

R:
- Controlla se le terminazioni di riga sono cambiate (CRLF ↔ LF).
- Verifica che il file non sia stato riformattato da un auto-formatter.
- PyUCC usa hashing del contenuto—qualsiasi modifica a livello di byte attiva lo stato "Modified".

D: Come resetto tutte le baseline?

R:
- Le baseline sono memorizzate nella cartella `baseline/` (default).
- Elimina la cartella baseline o specifiche sottocartelle baseline per resettare.

D: Posso eseguire PyUCC in pipeline CI/CD?

R:
- Sì! Usa la modalità CLI:
`bash python -m pyucc differ create /path/to/repo python -m pyucc differ diff <baseline_id> /path/to/repo python -m pyucc duplicates /path/to/repo --threshold 5.0`
- Analizza l'output JSON o i report testuali nei tuoi script di pipeline.
