[ITADINFO]
CONVEGNO ITALIANO
SULLA DIDATTICA DELL'INFORMATICA

VALUTARE IL PROCESSO DI SVILUPPO SOFTWARE REPOSITORY GITHUB, RUBRICHE E FEEDBACK RAPIDO

Relatore: Francesco Adriani



VALUTARE IL PROCESSO DELLO SVILUPPO SOFTWARE

(1) Il codice finale



(2) Il processo nascosto



(3) Cosa manca all'immagine?



CHE COSA VEDE DAVVERO IL DOCENTE?

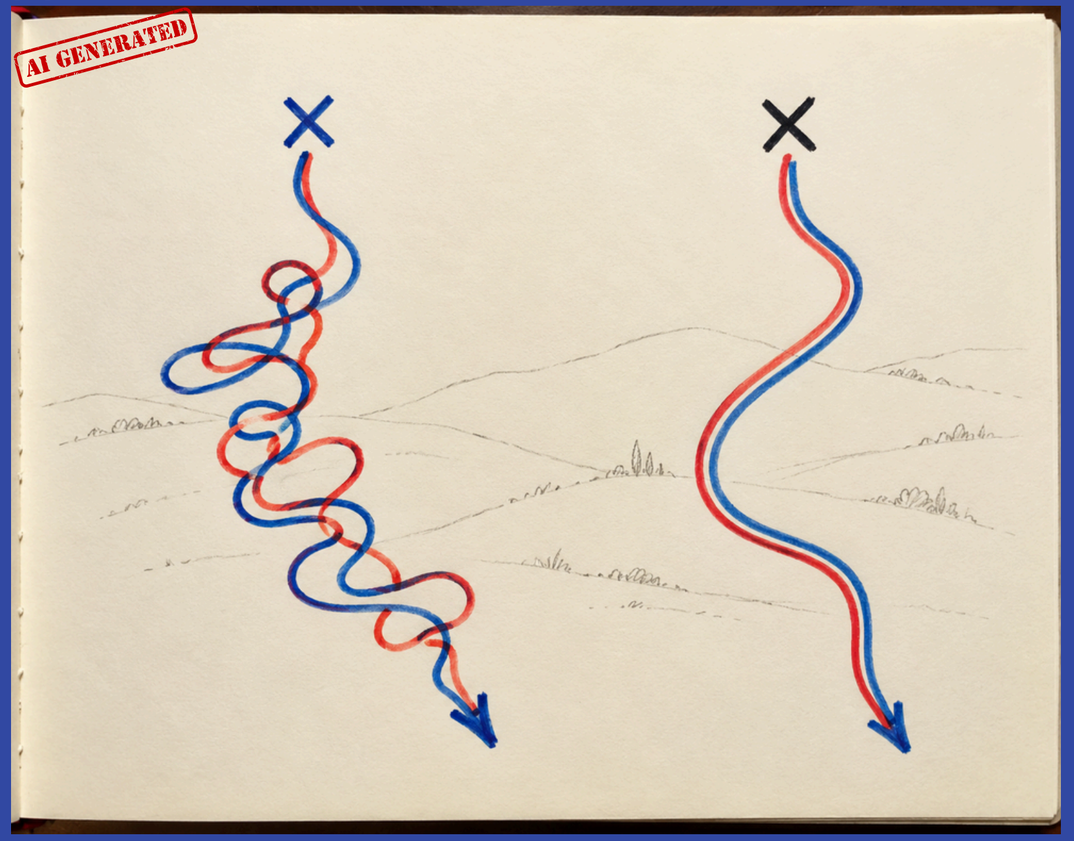
Il codice finale racconta solo una parte dell'apprendimento. Quando il docente osserva il prodotto consegnato, vede il risultato — ma non il percorso: le scelte fatte, gli errori superati, la progressione del pensiero. Valutare solo il prodotto finale rischia di rendere invisibile ciò che conta di più nel processo di apprendimento dell'informatica.

VALUTARE IL PROCESSO DELLO SVILUPPO SOFTWARE

(1) Risultato simile



(2) Processo differente e nascosto



DUE PRODOTTI SIMILI RACCONTANO LO STESSO PERCORSO?

Due studenti consegnano un progetto quasi identico: stesso output, stessa valutazione? Non necessariamente. Il percorso seguito — i passi, gli errori, le revisioni — racconta qualcosa che il prodotto finale da solo non può mostrare.
Percorsi diversi possono portare a risultati simili, ma con approcci, competenze e apprendimenti profondamente differenti.

VALUTARE IL PROCESSO
DELLO SVILUPPO
SOFTWARE

COME RENDERE OSSERVABILE

il processo

Il codice finale non mostra tutto ciò che accade durante lo sviluppo. Come tracciare il lavoro a step, la collaborazione e la documentazione prima della consegna?
Rendere visibile il processo significa costruire evidenze: commit, branch, messaggi e revisioni diventano segnali concreti di progresso e apprendimento.



OBIETTIVI DIDATTICI

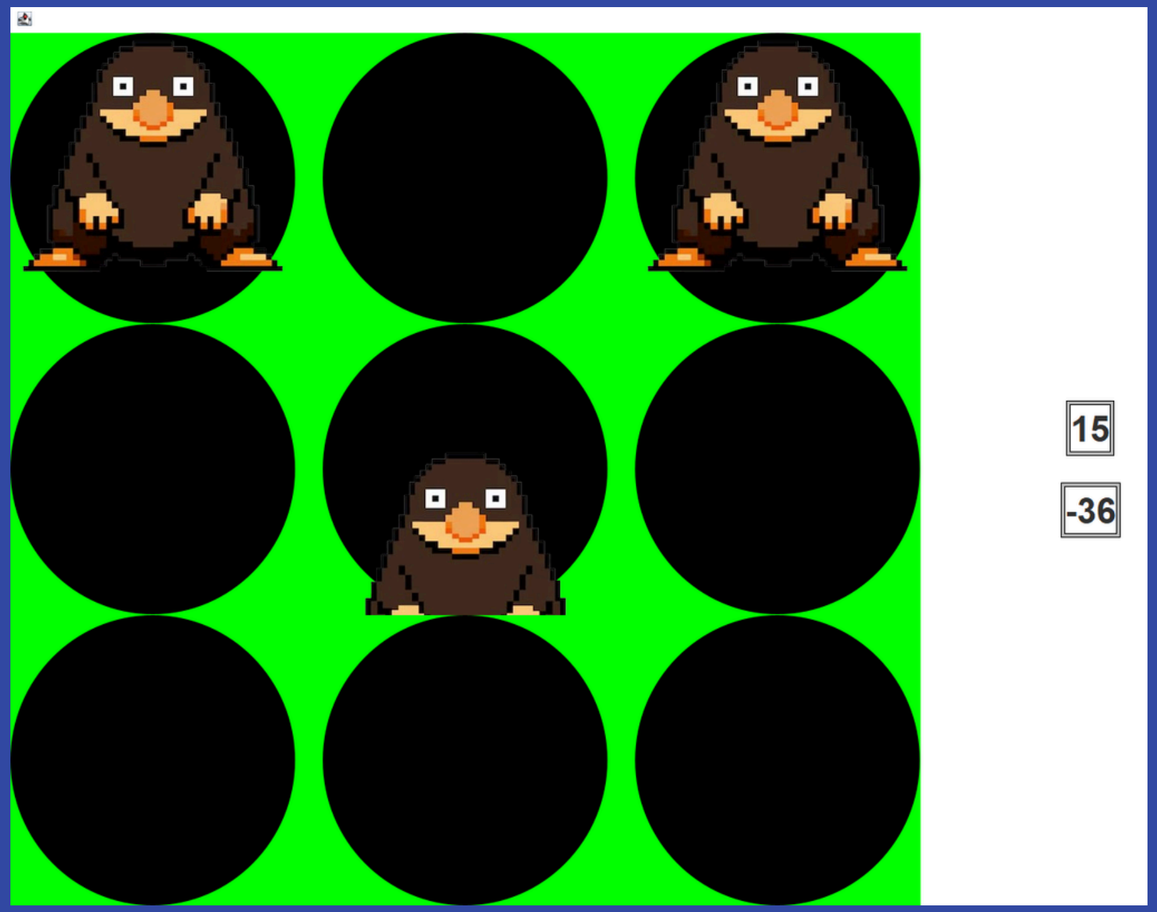
1. Affrontare un problema con metodo
2. Usare l’errore per imparare
3. Sviluppare autonomia e autoregolazione
4. Collaborare assumendosi responsabilità individuali
5. Comunicare e riflettere sul lavoro svolto
6. Mantenere il lavoro focalizzato sul problema assegnato
7. Comprendere e usare i criteri di valutazione

OBIETTIVI DISCIPLINARI

1. Analizzare requisiti e delimitare il problema
2. Progettare una soluzione prima di codificarla
3. Applicare le conoscenze di Java e dei thread
4. Organizzare lo sviluppo in modo incrementale
5. Verificare e correggere il codice
6. Usare Git per versionare e rendere tracciabile il lavoro
7. Documentare le scelte tecniche

OBIETTIVI DEL DOCENTE

1. Valutare il percorso insieme al risultato
2. Rendere trasparenti i criteri
3. Rendere osservabile il processo di sviluppo
4. Restituire feedback in tempi più brevi e utili
5. Rendere sostenibile la raccolta delle evidenze
6. Mantenere il giudizio professionale del docente
7. Sostenere gli studenti nelle diverse fasi del compito



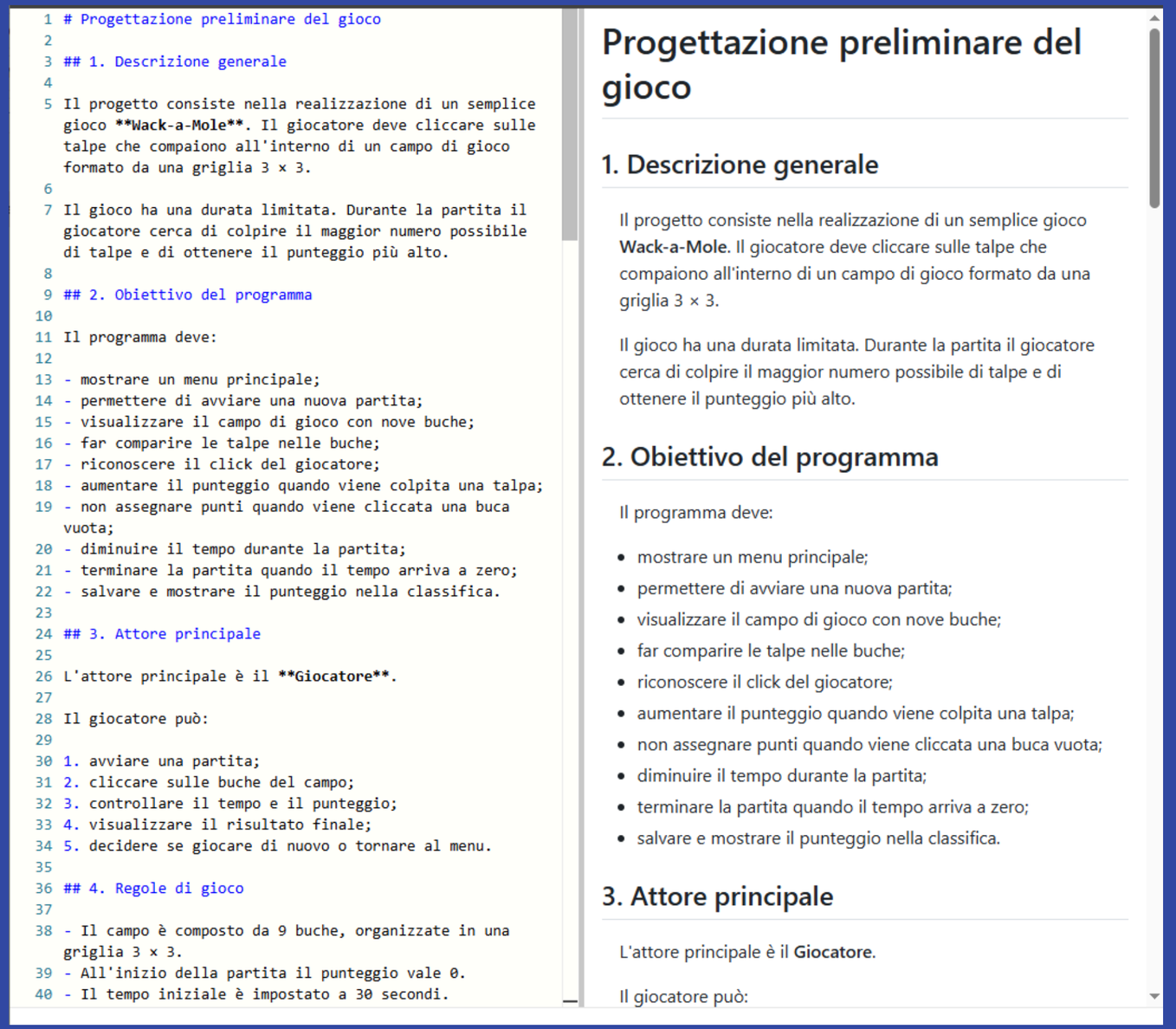
IL CASO DI STUDIO: ACCHIAPPA LA TALPA

requisiti →
mockup →
UML →
codice →
verifica →
revisione.

L'artefatto finale è il punto di arrivo; la valutazione osserva anche come lo studente costruisce, verifica e migliora la soluzione.

ANALISI DEI REQUISITI

Il corretto approccio ad un progetto informatico deve necessariamente iniziare con l'analisi dei requisiti e la stesura su un documento strutturato.
Si richiede allo studente di scrivere in markdown in modo che ogni modifica successiva possa essere tracciata in git come se fosse una revisione del codice.
Inoltre su github i documenti in markdown possono essere visualizzati direttamente dal browser e questo facilita l'accesso alla documentazione anhe senza aver installato sul pc un viewer di documenti



```
# Progettazione preliminare del gioco

## 1. Descrizione generale

Il progetto consiste nella realizzazione di un semplice gioco **Wack-a-Mole**. Il giocatore deve cliccare sulle talpe che compaiono all'interno di un campo di gioco formato da una griglia 3 × 3.

Il gioco ha una durata limitata. Durante la partita il giocatore cerca di colpire il maggior numero possibile di talpe e di ottenere il punteggio più alto.

## 2. Obiettivo del programma

Il programma deve:

- mostrare un menu principale;
- permettere di avviare una nuova partita;
- visualizzare il campo di gioco con nove buche;
- far comparire le talpe nelle buche;
- riconoscere il click del giocatore;
- aumentare il punteggio quando viene colpita una talpa;
- non assegnare punti quando viene cliccata una buca vuota;
- diminuire il tempo durante la partita;
- terminare la partita quando il tempo arriva a zero;
- salvare e mostrare il punteggio nella classifica.

## 3. Attore principale

L'attore principale è il **Giocatore**.

Il giocatore può:

1. avviare una partita;
2. cliccare sulle buche del campo;
3. controllare il tempo e il punteggio;
4. visualizzare il risultato finale;
5. decidere se giocare di nuovo o tornare al menu.

## 4. Regole di gioco

- Il campo è composto da 9 buche, organizzate in una griglia 3 × 3.
- All'inizio della partita il punteggio vale 0.
- Il tempo iniziale è impostato a 30 secondi.
```


Progettazione preliminare del gioco

1. Descrizione generale

Il progetto consiste nella realizzazione di un semplice gioco **Wack-a-Mole**. Il giocatore deve cliccare sulle talpe che compaiono all'interno di un campo di gioco formato da una griglia 3 × 3.

Il gioco ha una durata limitata. Durante la partita il giocatore cerca di colpire il maggior numero possibile di talpe e di ottenere il punteggio più alto.

2. Obiettivo del programma

Il programma deve:

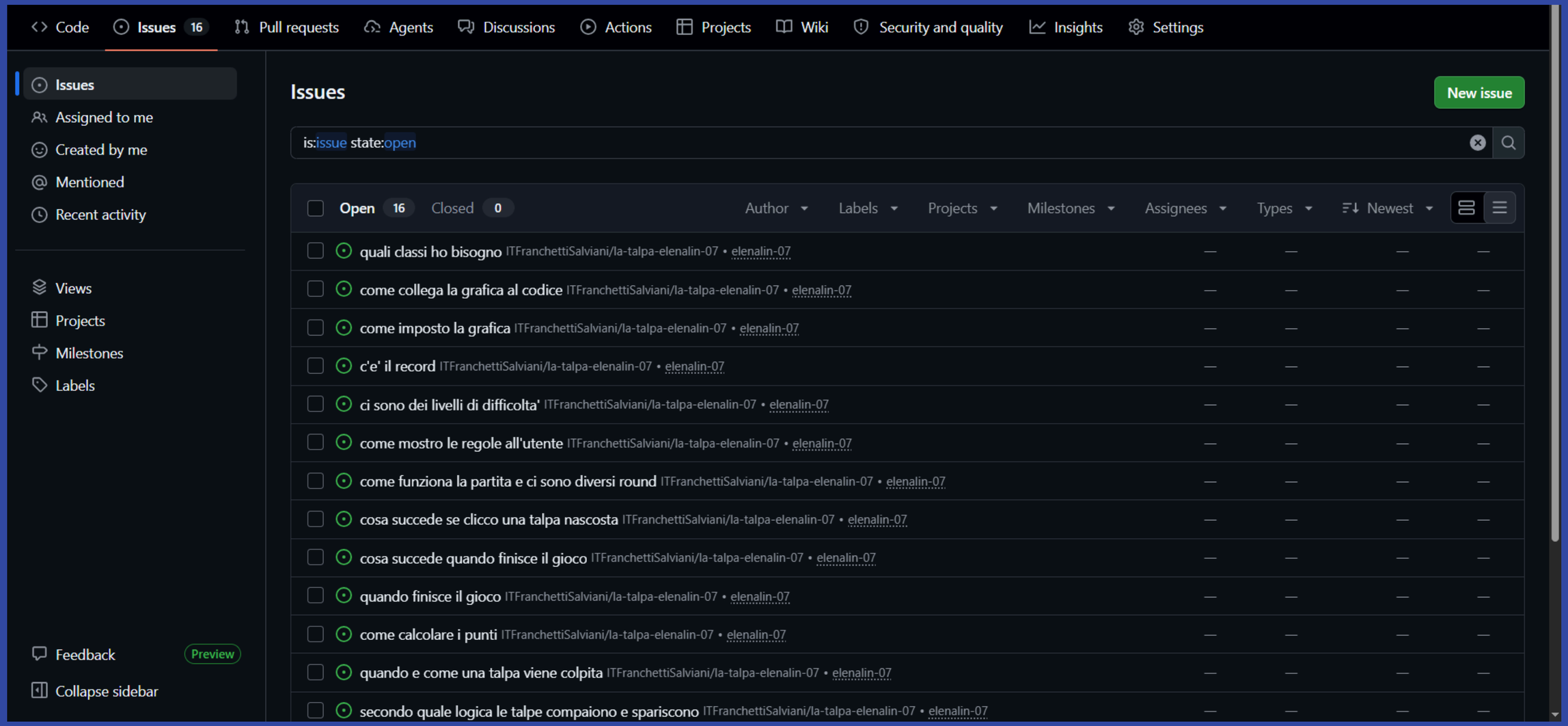
- mostrare un menu principale;
- permettere di avviare una nuova partita;
- visualizzare il campo di gioco con nove buche;
- far comparire le talpe nelle buche;
- riconoscere il click del giocatore;
- aumentare il punteggio quando viene colpita una talpa;
- non assegnare punti quando viene cliccata una buca vuota;
- diminuire il tempo durante la partita;
- terminare la partita quando il tempo arriva a zero;
- salvare e mostrare il punteggio nella classifica.

3. Attore principale

L'attore principale è il **Giocatore**.

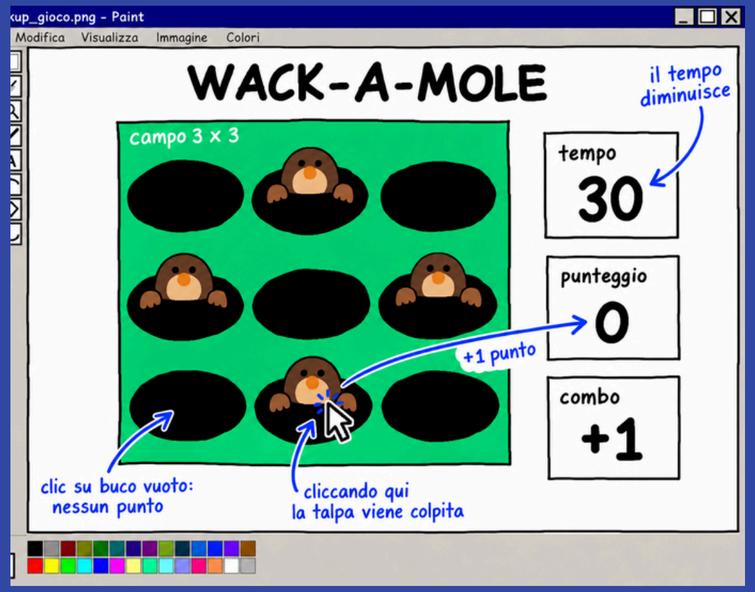
Il giocatore può:

TRASCRIZIONE DEI REQUISITI COME ISSUE

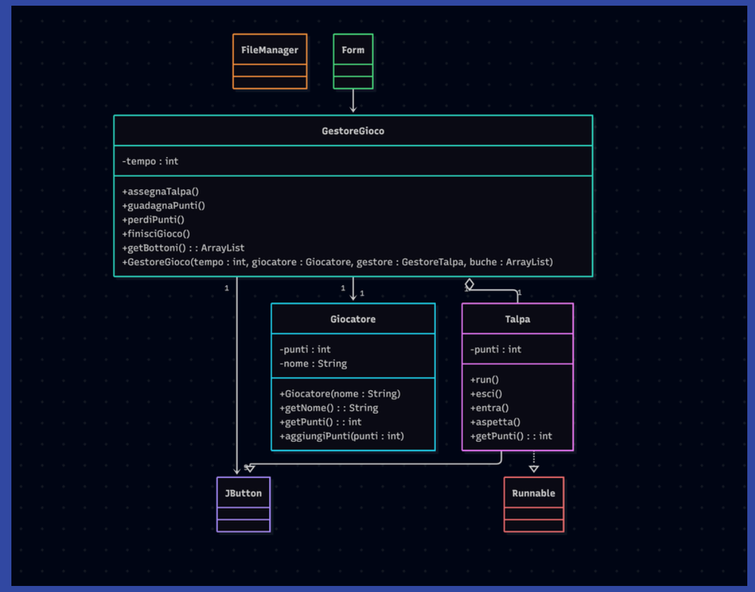


VALUTARE IL PROCESSO DELLO SVILUPPO SOFTWARE

(1) Wireframe e mockup UI in png



(2) Diagramma UML delle classi in mermaid



(3) Flusso di gioco e logica in mermaid



MOCKUP E UML

Come strutturare il lavoro prima di scrivere codice? La progettazione con mockup e diagrammi UML guida lo sviluppo del gioco Whac-a-Mole in Java, rendendo esplicite le scelte architetturali e le responsabilità di ogni componente prima dell'implementazione.
La fase di progettazione anticipa i problemi, favorisce la pianificazione incrementale e rende il processo valutabile fin dall'inizio.

AVANZAMENTI RICONOSCIBILI

Ogni commit dovrebbe rappresentare uno step comprensibile e coerente con l'obiettivo dichiarato. Messaggio, file coinvolti e diff aiutano a ricostruire l'evoluzione del lavoro.
Non conta il numero di commit: conta ciò che ciascun avanzamento racconta.

01 Commit iniziale: struttura base

```
 * @author cucchiarini.cesare
 */
public class Form extends javax.swing.JFrame {

    private static final java.util.logging.Logger logger = java.util.logging.Logger.getLogger(Form.

    JButton[] bottoni;
    GestoreGioco g;
    /**
     * Creates new form Form
     */
    public Form() {
        initComponents();
        g = new GestoreGioco(new Talpa(3), new Giocatore());
        bottoni = g.getBottoni();

        this.getContentPane().setLayout(new GridLayout(3, 3));

        for (int i = 0; i < 9; i++) {
            JPanel p = new JPanel();
            p.add(bottoni[i]);
            this.getContentPane().add(p);
        }

        g.assegnaTalpa();
    }
```

02 Aggiunta logica di gioco

```
public void esci(){
    cliccabile = true;
    altezza = crescita;
    while(altezza < altezzaMax && cliccabile){
        try{
            Thread.sleep(50);
            SwingUtilities.invokeLater(() ->{
                altezza+=crescita;
                this.setPreferredSize(new Dimension(larghezza, altezza));
                buco.revalidate();
                buco.repaint();
            });
        }
        catch(InterruptedException e){}
    }
}

public void entra(){
    synchronized(box){
        box.setPunti((!cliccabile && gioco) ? punti : -punti);
        box.notifyAll();
    }
    cliccabile = false;
    while(altezza > crescita){
        try{
            Thread.sleep(50);
        }
```

03 Refactoring e test finali

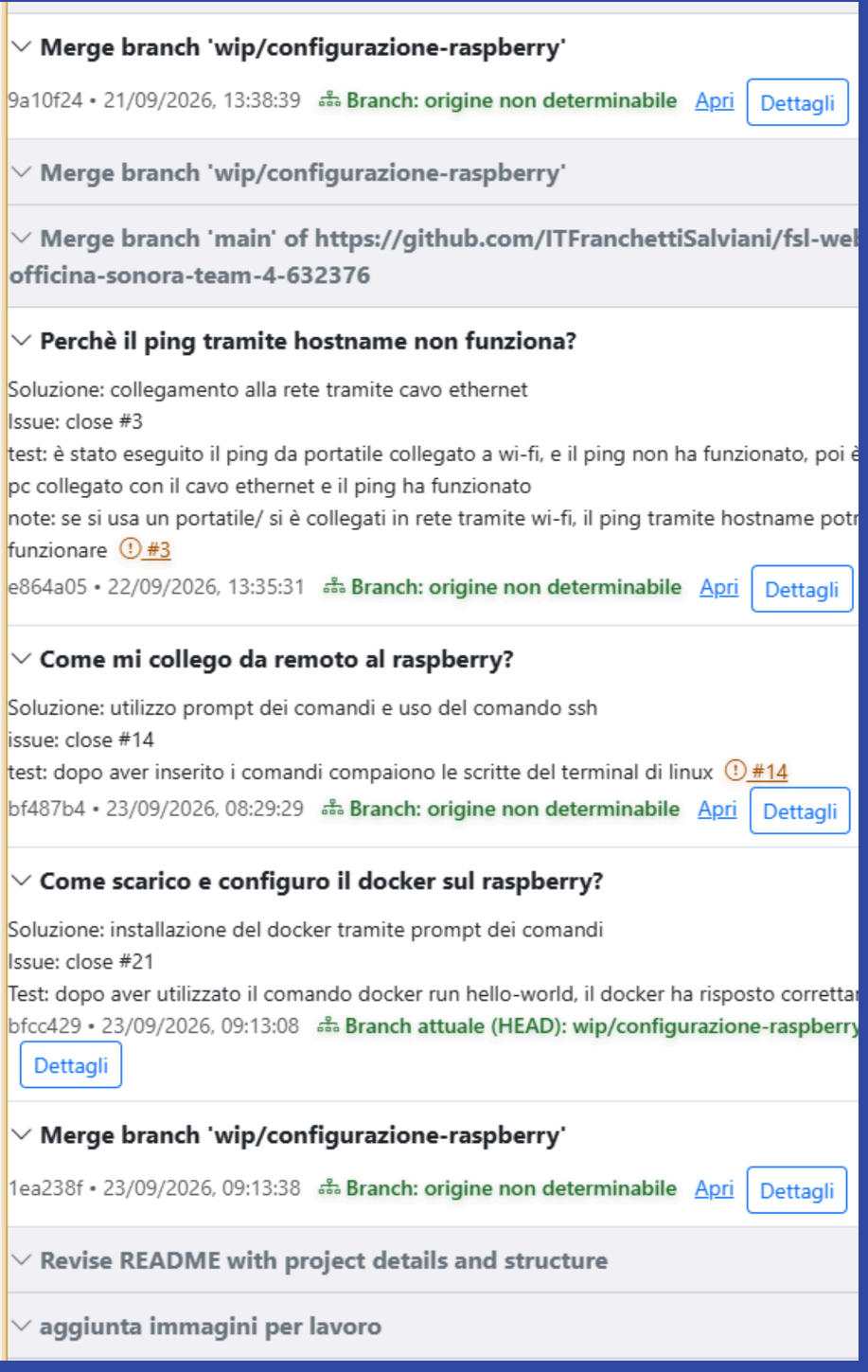
```
ublic class GestoreGioco extends Thread{
  private Talpa talpa;
  private Giocatore g;
  private JPanel[] panels;
  private IntBox box;
  private ActionListener l;

  public GestoreGioco(Talpa talpa, Giocatore g, JPanel[] panels, IntBox box) {
      this.g = g;
      this.box = box;
      this.talpa = talpa;
      talpa.setBox(box);
      this.panels = panels;

      l = new ActionListener(){
          @Override
          public void actionPerformed(ActionEvent ae){
              talpa.colpita();
          }
      };
      this.talpa.addActionListener(l);
  }

  public void scegliBuco(){
      int buco = new Random().nextInt(9);
      assegnaTalpa(panels[buco]);

      panels[buco].revalidate();
```



DOCUMENTARE SCELTE E TEST

Messaggi di commit descrittivi

Un buon messaggio di commit non si limita a dire "cosa" è stato fatto, ma spiega il "perché". Esempio: "Aggiunto controllo input utente – evita crash con valori nulli (issue #12)".

Documentare i test effettuati

I messaggi riportano i test eseguiti prima del commit: casi limite verificati, risultati attesi e ottenuti. Esempio: "Testato con input vuoto, stringa lunga e caratteri speciali – tutti i casi superati".

Motivare le scelte progettuali

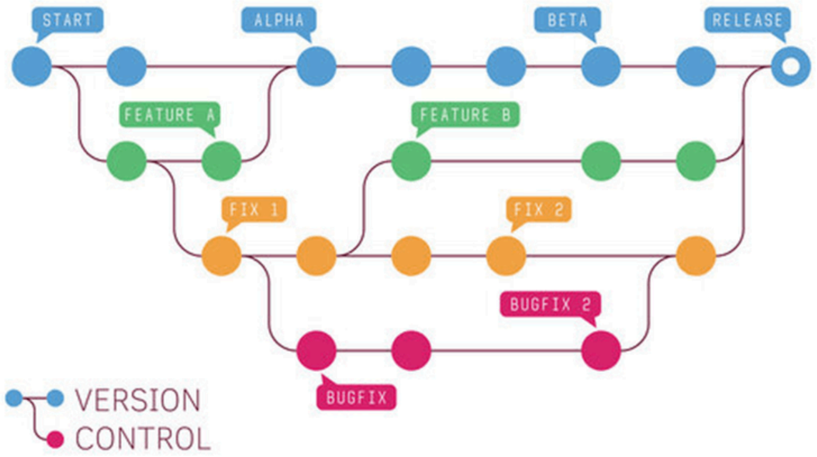
Ogni commit può includere riferimenti a decisioni architetturali, alternative scartate e ragioni della scelta adottata. La cronologia diventa un diario tecnico leggibile e valutabile.

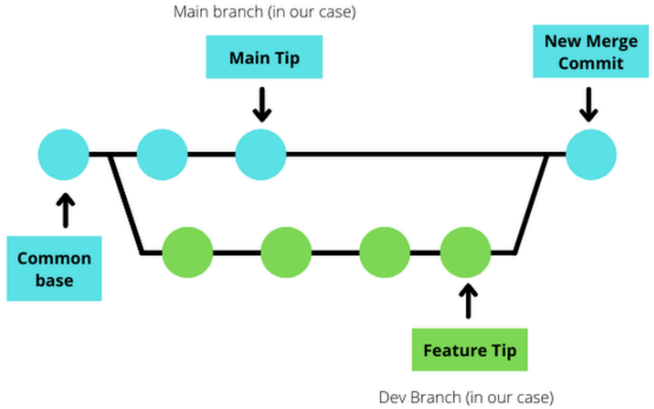
Domanda-guida

Perché la documentazione è parte integrante del processo e non un'aggiunta finale? Come trasformare la cronologia dei commit in una fonte di apprendimento e valutazione autentica?

BRANCH E INTEGRAZIONI

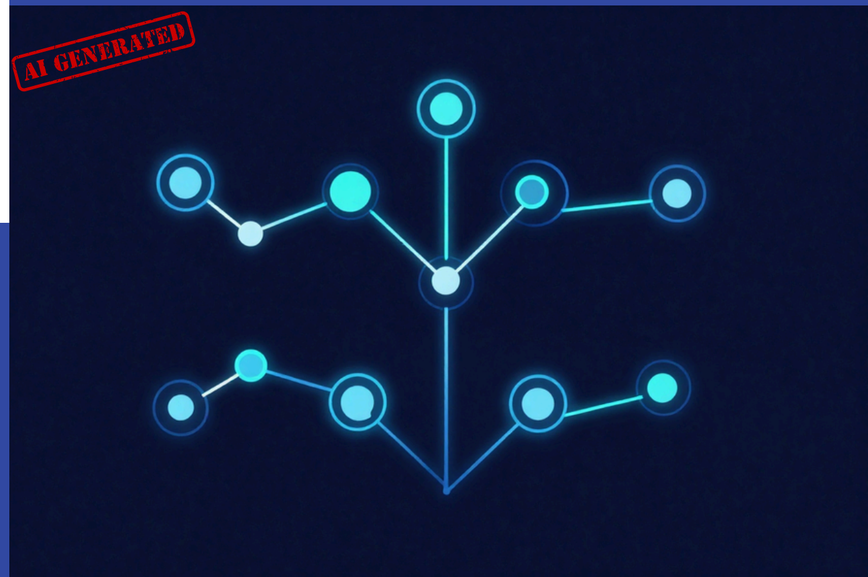
1 Quando richiesto, un branch può isolare una funzionalità o un fix e rendere più leggibile l'evoluzione del progetto.





2 Il merge documenta l'integrazione del lavoro completato nel ramo principale.

3 Nel lavoro condiviso, branch e merge aiutano a rendere osservabili contributi, integrazioni e responsabilità tecniche.



REVISIONE E MIGLIORAMENTO

Il refactoring e la correzione progressiva del codice non sono segnali di debolezza, ma evidenze di apprendimento autentico. Ogni modifica successiva racconta un percorso di crescita: lo studente che rivede il proprio lavoro dimostra capacità critica e autonomia.
L'errore, tracciato e corretto tramite commit successivi, diventa parte integrante del diario tecnico e della valutazione del processo.

(1) Prima versione del codice



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package wac.a.mole;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JFrame;
import javax.swing.Timer;

/**
 *
 * @author bernicchi.davide
 */
public class GestoreGioco {
            Timer rt = new Timer(1000, new ActionListener(){
                public void actionPerformed(ActionEvent e) {

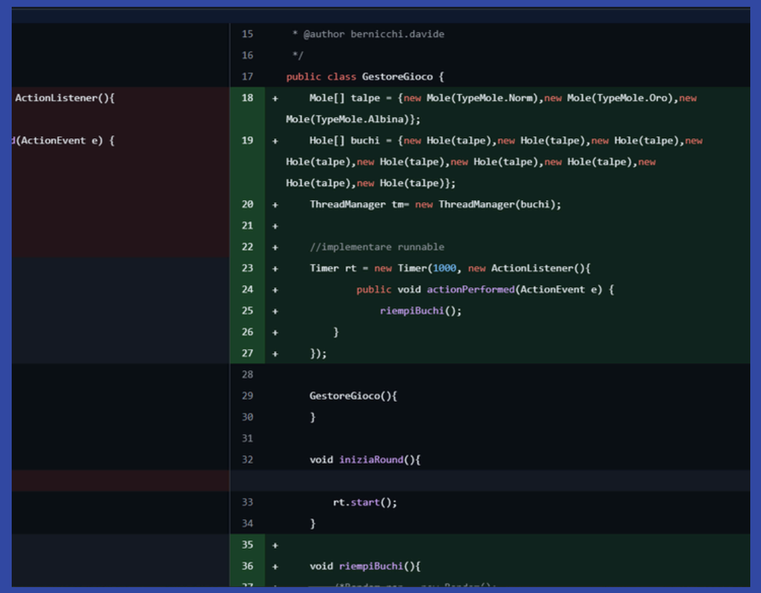
            }
        });

    GestoreGioco(){
    }

    void iniziaRound(){

        rt.start();
    }
}
```

(2) Refactoring e correzioni



```
 * @author bernicchi.davide
 */
public class GestoreGioco {
+    Mole[] talpe = {new Mole(TypeMole.Norm),new Mole(TypeMole.Oro),new
Mole(TypeMole.Albina)};
+    Hole[] buchi = {new Hole(talpe),new Hole(talpe),new Hole(talpe),new
Hole(talpe),new Hole(talpe),new Hole(talpe),new Hole(talpe),new
Hole(talpe),new Hole(talpe)};
+    ThreadManager tm= new ThreadManager(buchi);
+
+    //implementare runnable
+    Timer rt = new Timer(1000, new ActionListener(){
+            public void actionPerformed(ActionEvent e) {
+                riempiBuchi();
+        }
+    });

    GestoreGioco(){
    }

    void iniziaRound(){

        rt.start();
    }
+
+    void riempiBuchi(){
```

(3) Versione migliorata finale



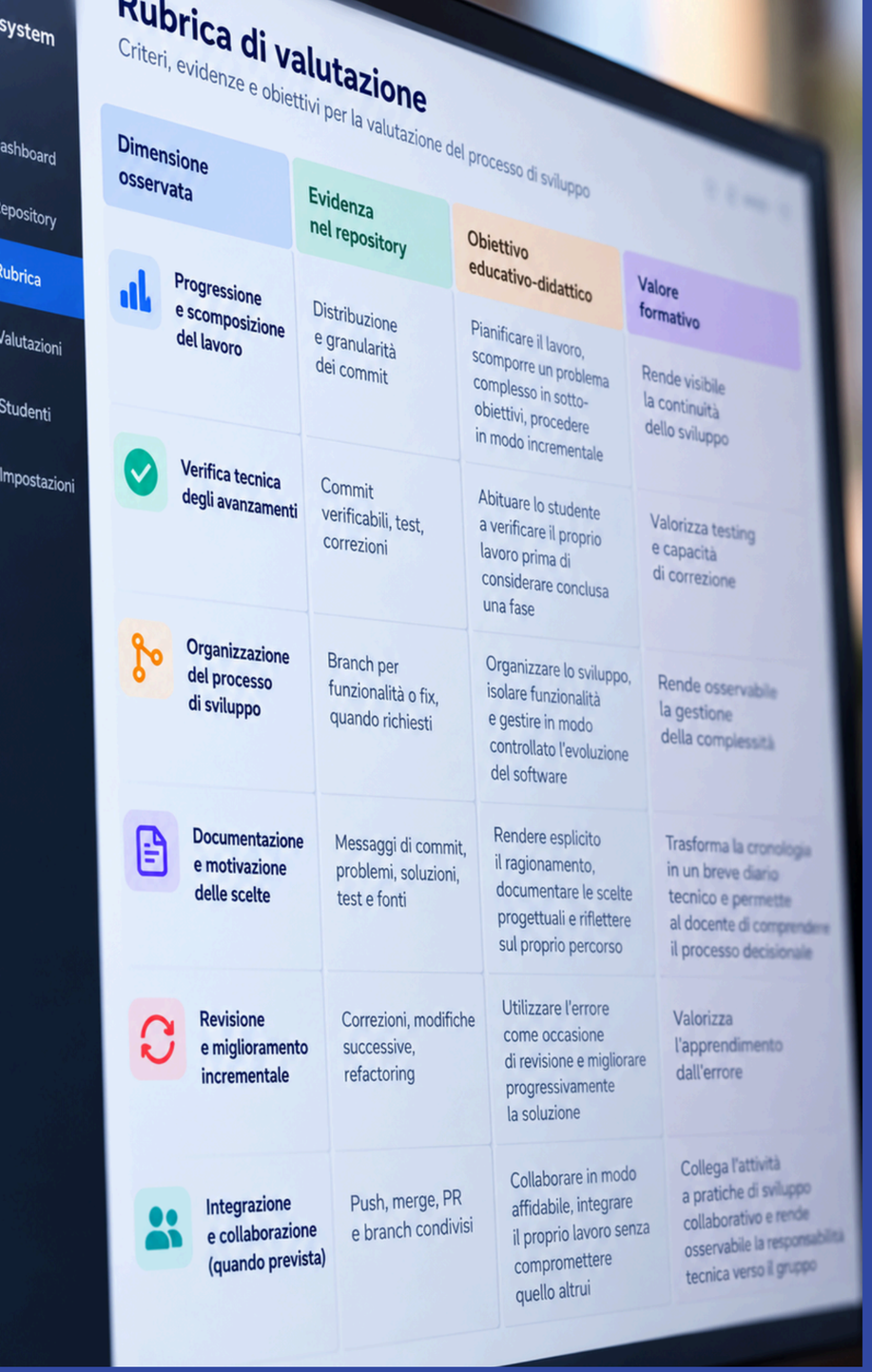
```
/**
 *
 * @author bernicchi.davide
 */
public class GestoreGioco {
    Mole[] talpe = {new Mole(TypeMole.Norm),new Mole(TypeMole.Oro),new Mole(TypeMole.Albina)};
    Hole[] buchi = {new Hole(talpe),new Hole(talpe),new Hole(talpe),new Hole(talpe),new Hole(ta
    ThreadManager tm= new ThreadManager(buchi);

    //implementare runnable
    Timer rt = new Timer(1000, new ActionListener(){
            public void actionPerformed(ActionEvent e) {
                riempiBuchi();
        }
    });

    GestoreGioco(){
    }

    void iniziaRound(){
        rt.start();
    }

    void riempiBuchi(){
        /*Random ran = new Random();
        Hole[] output = {buchi[ran.nextInt(1,9)], buchi[ran.nextInt(1,9)], buchi[ran.nextInt(1,9)]};
        return output; */
        tm.ThreadStarter();
```



LA RUBRICA DI VALUTAZIONE

Progressione e scomposizione

Distribuzione e granularità dei commit: continuità del lavoro, sotto-obiettivi e sviluppo incrementale.

Organizzazione e documentazione

Branch quando richiesti; messaggi, problemi, soluzioni, test e fonti rendono ricostruibile il processo.

Coerenza degli avanzamenti

Messaggio del commit, file coinvolti e diff coerenti con l'obiettivo dichiarato.

Revisione e collaborazione

Correzioni e refactoring; push, merge e pull request quando l'attività prevede lavoro condiviso.



RUBRICA:

PROGRESSIONE E SCOMPOSIZIONE

Distribuzione nel tempo
La distribuzione temporale dei commit è un'evidenza del percorso, non un punteggio automatico: va interpretata nel contesto del compito.

Sequenza degli step
La sequenza degli avanzamenti aiuta a osservare se il lavoro procede per fasi riconoscibili e coerenti con la progettazione.

Granularità contestuale
La granularità è utile quando i commit corrispondono a sotto-obiettivi comprensibili. Il numero di commit, da solo, non costituisce merito.

Nessuna metrica automatica
La rubrica interpreta insieme distribuzione, sequenza e scomposizione per ricostruire la progressione del lavoro.

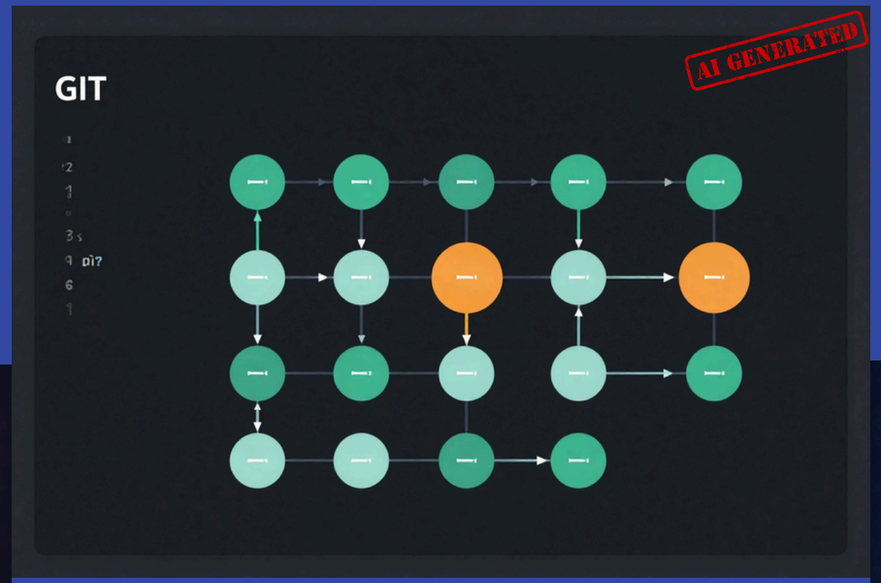
COERENZA E SIGNIFICATIVITÀ DEGLI AVANZAMENTI

Che cosa osservo nel commit

1. Obiettivo dichiarato nel messaggio
2. File coinvolti coerenti con l'obiettivo
3. Diff riconoscibile e circoscritto
4. Ampiezza delle modifiche proporzionata
5. Relazione chiara tra messaggio e contenuto
6. Modifiche ampie motivate quando necessarie

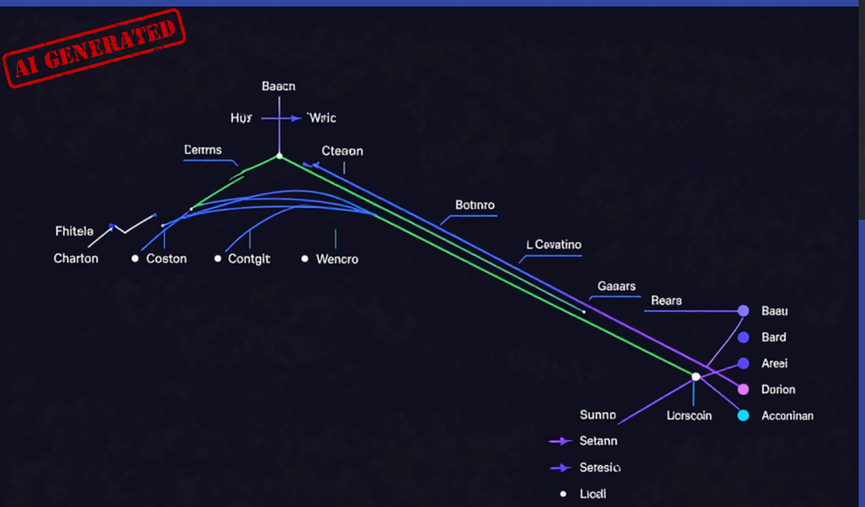
RUBRICA: ORGANIZZAZIONE DEL PROCESSO

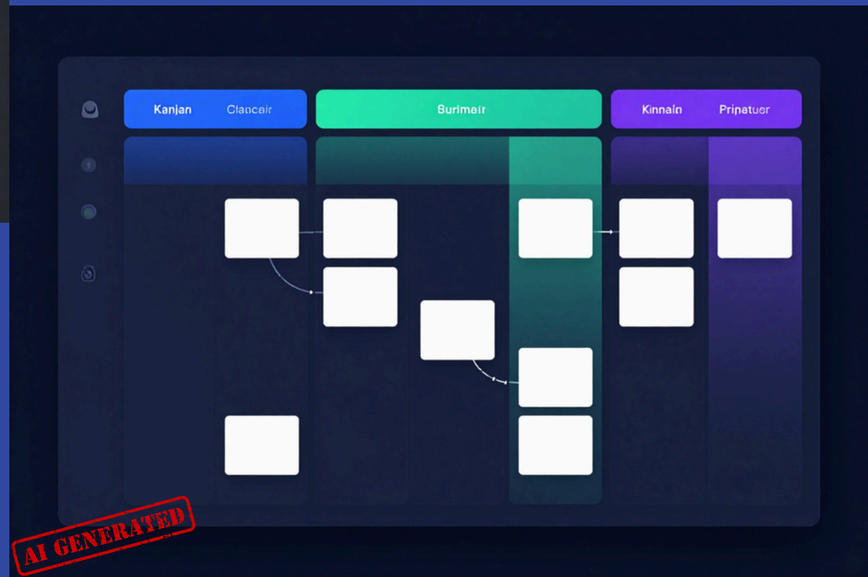
1 Usare branch dedicati per isolare ogni funzionalità, evitando interferenze nel codice principale durante lo sviluppo.

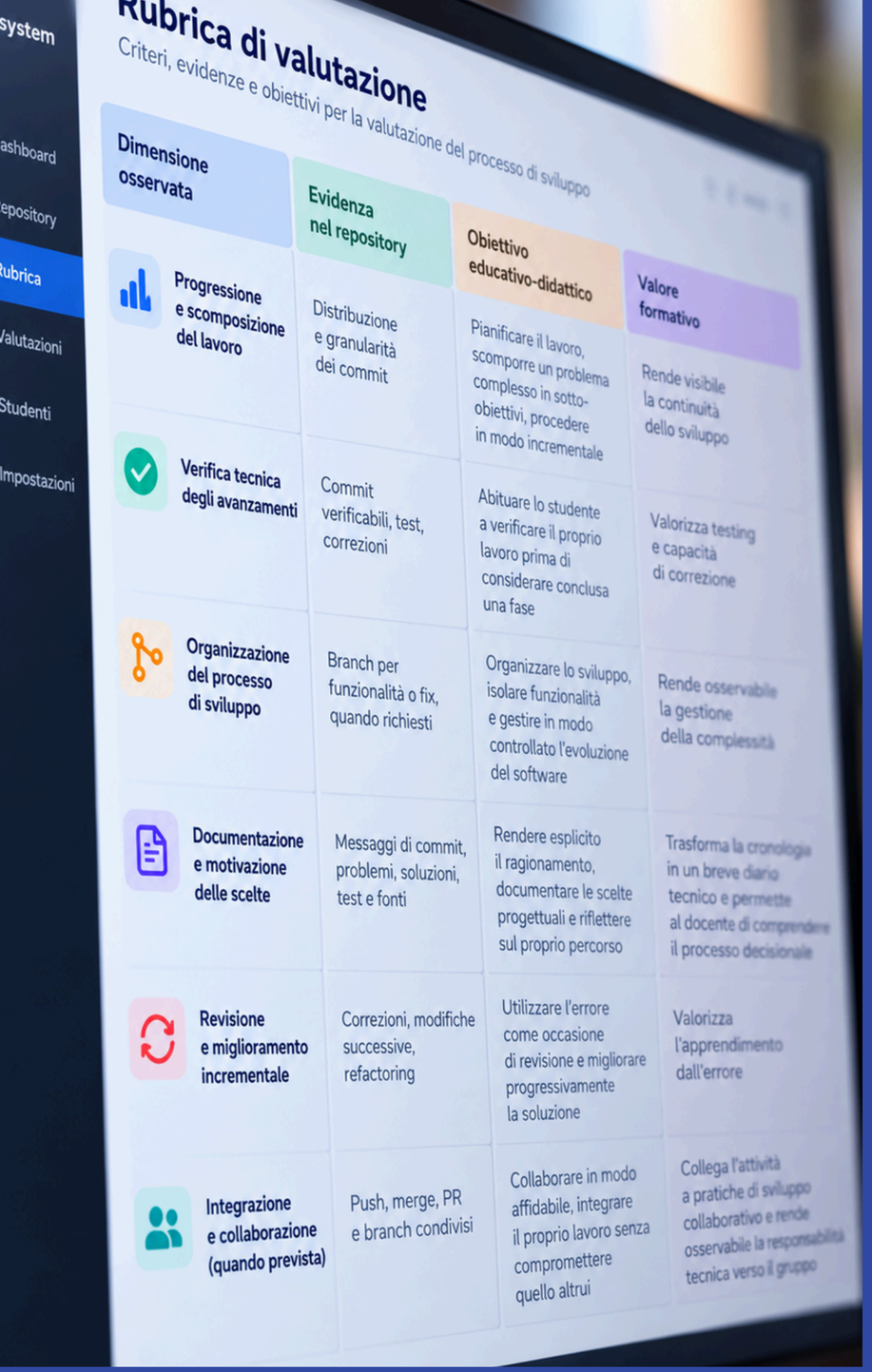


2 Gestire il flusso di lavoro con merge consapevoli e pull request, documentando l'integrazione tra le parti del progetto.

3 Pianificare le fasi di sviluppo con una struttura ordinata di commit e branch che rispecchi la scomposizione del problema.







RUBRICA: DOCUMENTAZIONE E MOTIVAZIONE

Messaggi di commit strutturati

Un buon messaggio di commit descrive cosa è stato fatto, perché e come. Es: "Aggiunto controllo input utente per evitare NullPointerException – testato con casi limite."

La cronologia come diario tecnico

La sequenza dei commit diventa un diario tecnico leggibile: mostra l'evoluzione del pensiero, le difficoltà affrontate e le soluzioni adottate passo dopo passo.

Commenti nel codice

I commenti non spiegano solo il "cosa" ma il "perché": motivano scelte progettuali, segnalano dubbi e tracciano decisioni tecniche prese durante lo sviluppo.

Criteri di valutazione

La rubrica valuta: chiarezza e completezza dei messaggi, coerenza tra codice e documentazione, capacità di motivare le scelte e di collegare teoria e pratica.

RUBRICA: REVISIONE E MIGLIORAMENTO

Valorizzare l'apprendimento dall'errore:

1. Refactoring del codice con motivazione esplicita nei commit
2. Correzione degli errori documentata e tracciabile
3. Test ripetuti e verifica progressiva del funzionamento
4. Miglioramenti successivi alla prima versione funzionante
5. Uso dell'errore come opportunità di apprendimento attivo
6. Confronto tra versioni: evidenza della crescita nel tempo

VALUTARE IL PROCESSO DELLO SVILUPPO SOFTWARE

25 REPOSITORY. CENTINAIA DI COMMIT. LI APRO UNO PER UNO?

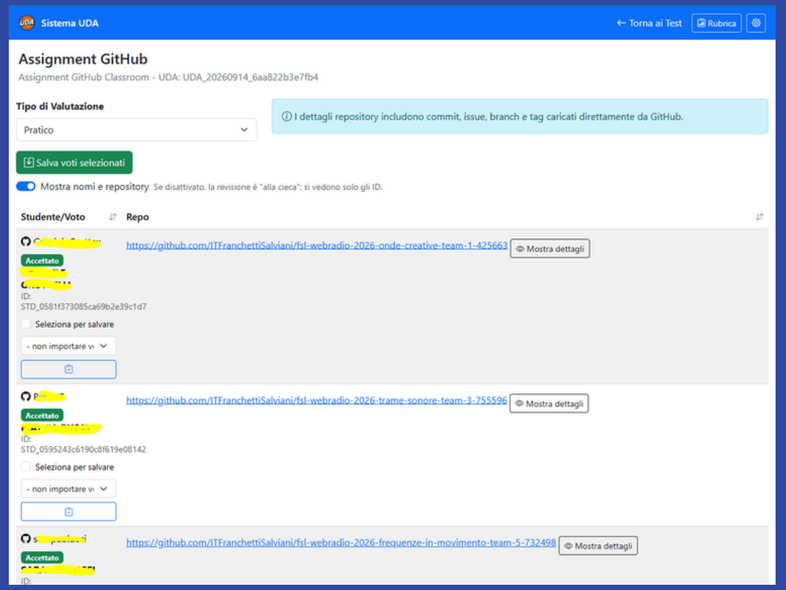


UDA-SYSTEM

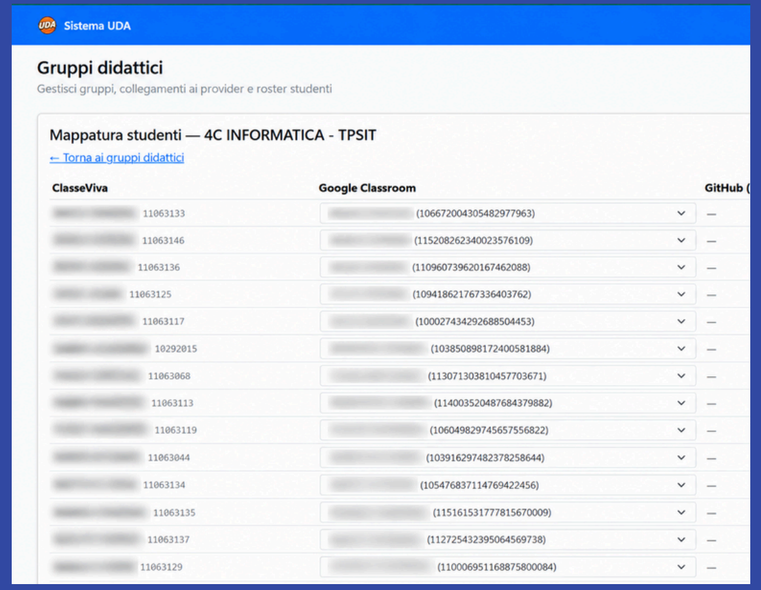
https://github.com/il-prof-f-a/uda-system

UDA-system associa i repository GitHub degli studenti alle attività del corso. Tramite le API di GitHub raccoglie e organizza commit, branch e modifiche al codice, permettendo al docente di consultare rapidamente l'evoluzione del processo e restituire feedback in tempi utili.

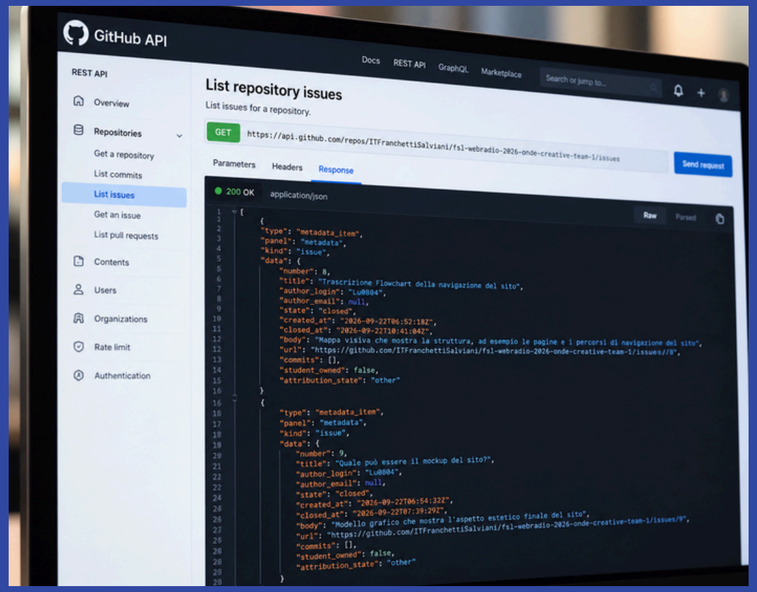
(1) Dashboard principale



(2) Associazione repo e studenti



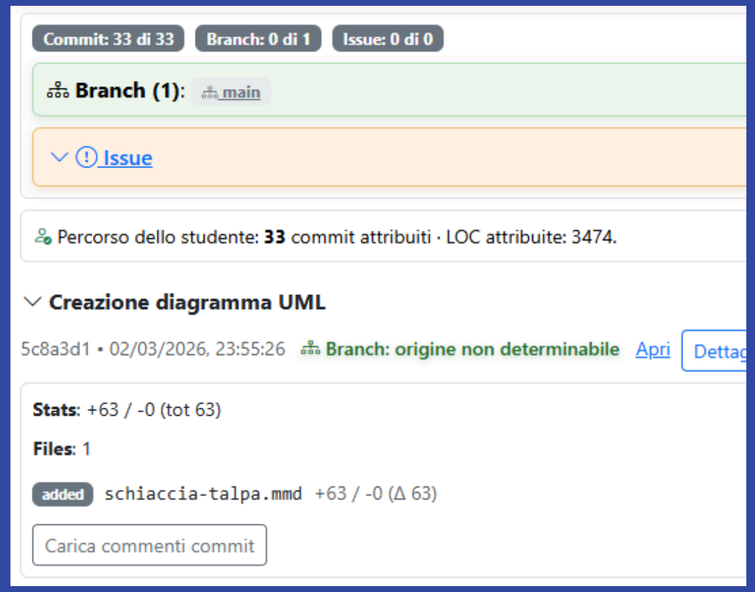
(3) Raccolta dati via API



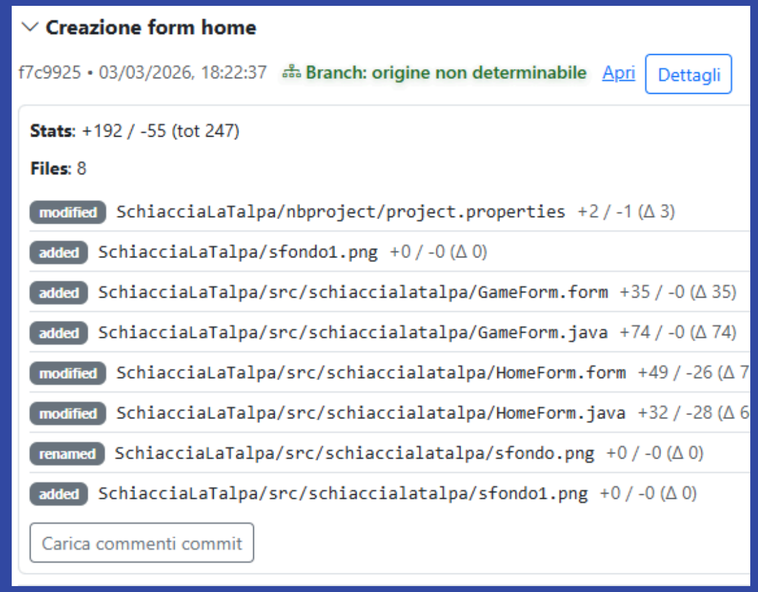
API GITHUB E DATI RACCOLTI

UDA-system interroga le API REST di GitHub per raccogliere dati sui repository: commit, branch, file modificati e righe aggiunte o rimosse.
Questi dati non sono una valutazione: organizzano le evidenze e rendono la consultazione più rapida, documentata e tempestiva.

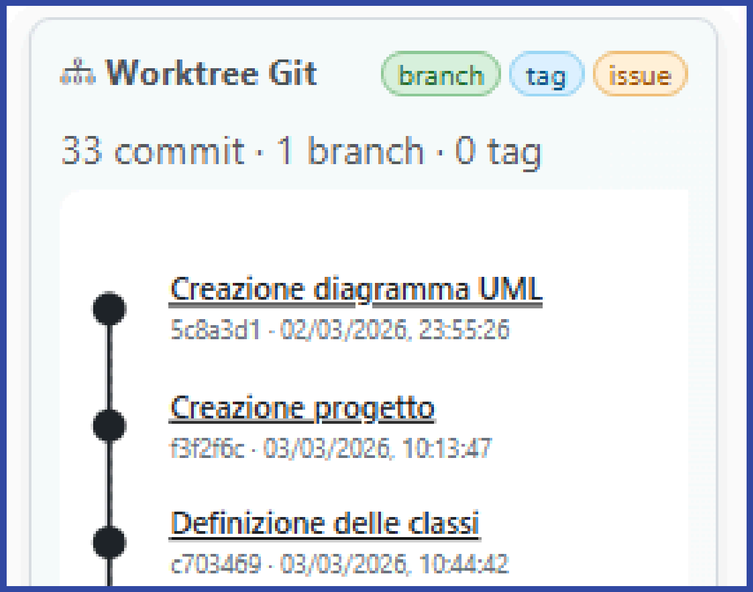
(1) Commit, branch e statistiche file



(2) Righe aggiunte e rimosse per commit



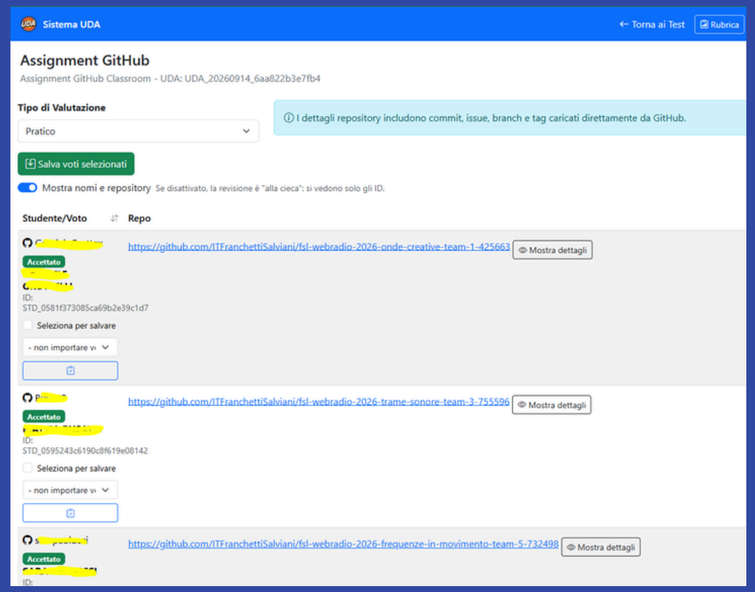
(3) Ricostruzione del worktree



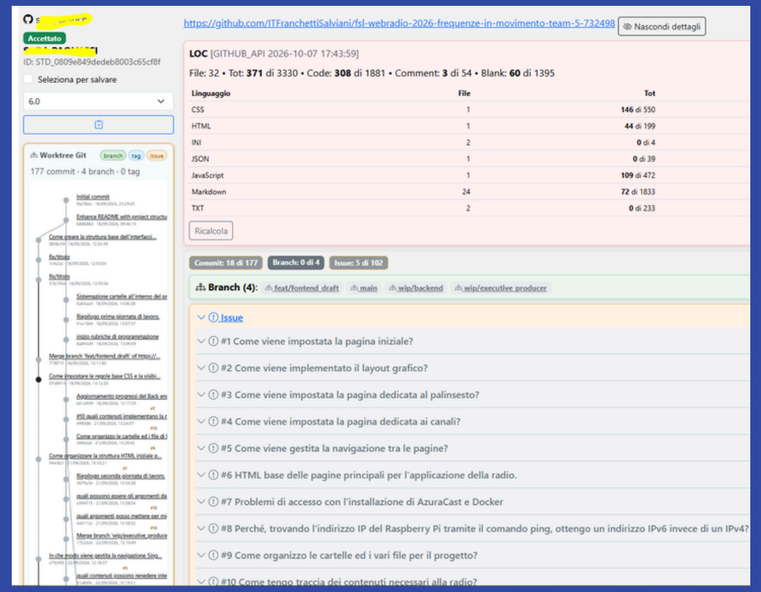
SINTESI DATI

Il docente può navigare direttamente dai dati raccolti ai repository degli studenti. I collegamenti attivi ai commit permettono di accedere immediatamente alle evidenze del processo: ogni riga della tabella riassuntiva è un punto di accesso alla storia del lavoro svolto. La navigazione è rapida, contestuale e integrata nella piattaforma UDA-system.

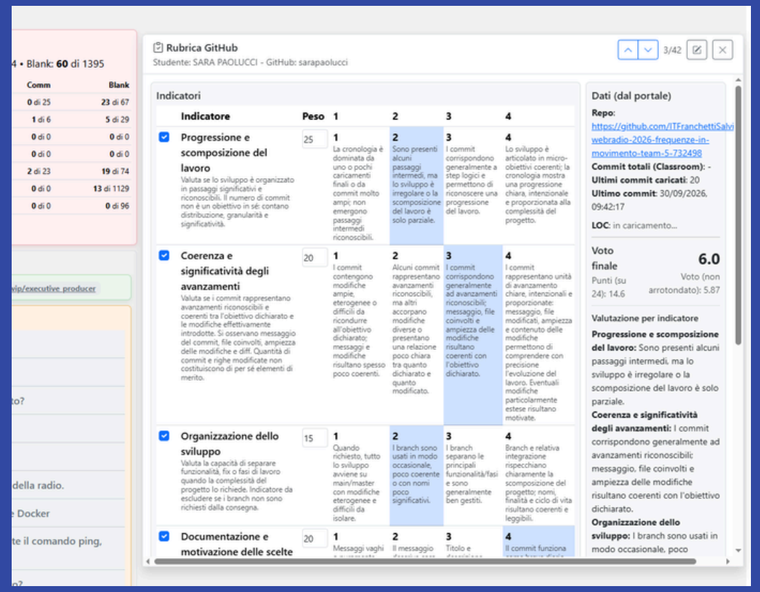
(1) Tabella riepilogativa studenti



(2) Dettaglio commit e repository



(3) Navigazione rapida alle evidenze



VALUTARE IL PROCESSO DELLO SVILUPPO SOFTWARE

UDA-SYSTEM
ALLEGGERIRE LA RACCOLTA DATI

Il sistema automatizza la raccolta dei dati meccanici (commit, branch, file modificati, righe aggiunte/rimosse) tramite API GitHub, liberando il docente dal lavoro ripetitivo.
Il docente mantiene il ruolo centrale: interpreta le evidenze, esercita il giudizio professionale e fornisce un feedback significativo e contestualizzato.



VALUTARE IL PROCESSO DI SVILUPPO SOFTWARE

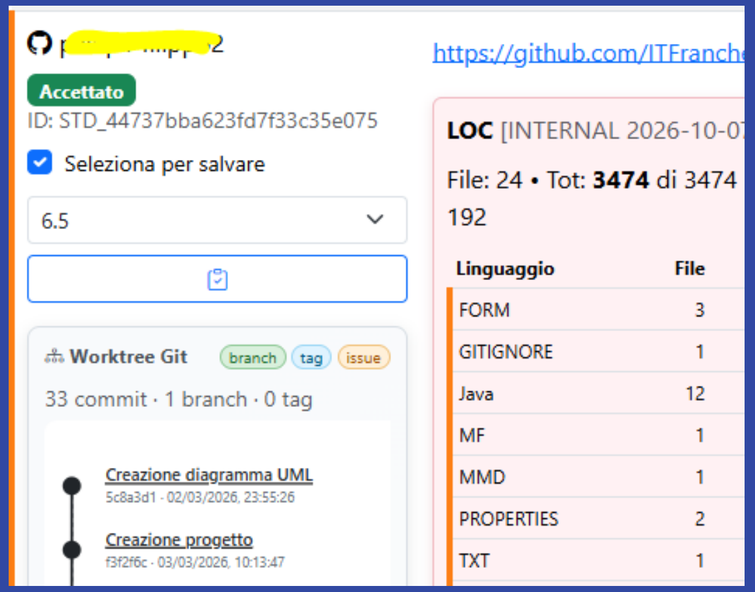
(1) Punteggi per dimensione



(2) Commenti e motivazioni



(3) Voto finale modificabile



FEEDBACK E VOTO NEL SISTEMA

UDA-system propone un voto aggregato a partire dai punteggi assegnati dal docente alle dimensioni della rubrica. Il docente esamina le evidenze, inserisce commenti e motivazioni e può modificare il voto finale prima della pubblicazione. L'automazione organizza i dati; la responsabilità del giudizio rimane al docente.

VALUTARE IL PROCESSO DELLO SVILUPPO SOFTWARE

FEEDBACK RAPIDO E *trasparente*

Tempi ridotti: il docente fornisce riscontri in tempi brevi grazie all'automazione dei dati meccanici.
Criteri chiari: le rubriche condivise rendono la valutazione comprensibile e prevedibile per gli studenti.
Attenzione al percorso: il feedback si concentra sul processo, non solo sul risultato finale.

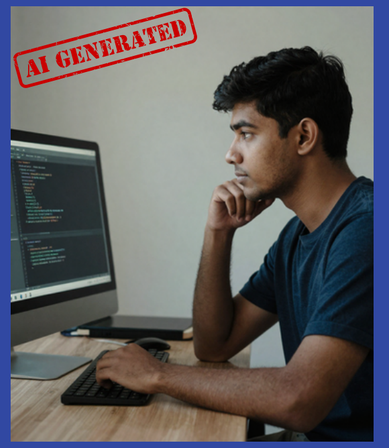


OSSERVAZIONI QUALITATIVE

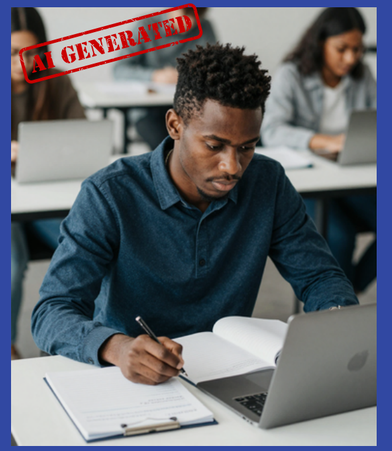
Dalle osservazioni in classe e dalle restituzioni informali sono emersi: maggiore attenzione agli step, criteri percepiti come più trasparenti e, in alcuni casi, maggiore autonomia nell'avvio del compito.



Metodo e pianificazione
Competenza trasversale



Lavoro incrementale
Competenza tecnica



Collaborazione attiva
Competenza collaborativa



Autonomia crescente
Autoregolazione



Uso dell'errore
Apprendimento continuo



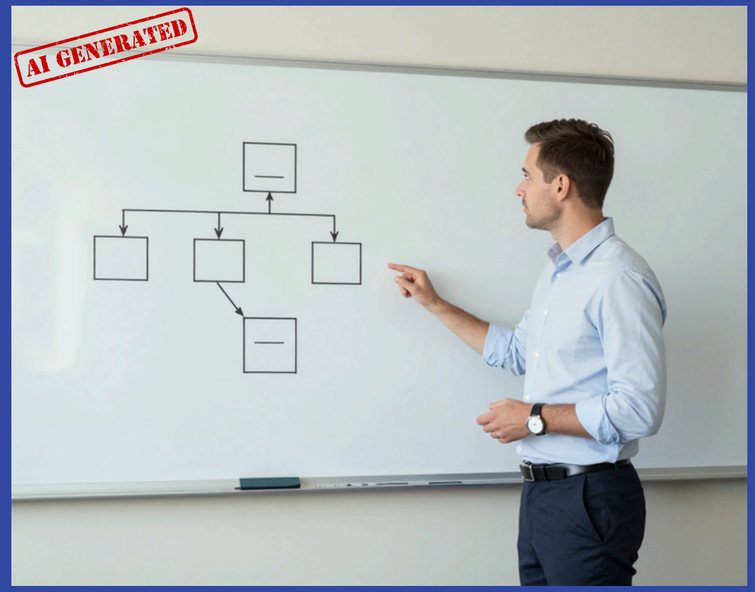
Problem solving
Comunicazione tecnica

VALUTARE IL PROCESSO DELLO SVILUPPO SOFTWARE

1 Scomposizione del problema



2 Progressione del lavoro



3 Autonomia nell'avvio



IMPATTO SULLA DIDATTICA

In alcuni casi, studenti con difficoltà nell'avviare autonomamente il compito hanno utilizzato la sequenza richiesta — requisiti, mockup, UML, sviluppo e revisione — come struttura per orientare il lavoro.
Nelle attività successive si è osservata una maggiore attenzione alla scomposizione del problema e alla progressione per step.

LIMITI E PROSPETTIVE FUTURE

Regole e formazione per Git
L'uso efficace di Git richiede regole condivise e formazione dedicata. Senza linee guida chiare, i commit risultano disorganizzati e il valore documentale si perde.

Questionari e analisi longitudinali
Sviluppi futuri includono l'integrazione di questionari di autovalutazione e analisi longitudinali per osservare la crescita degli studenti nel tempo.

Rubriche e checkpoint
Le rubriche devono essere accompagnate da checkpoint intermedi per guidare gli studenti nel percorso e rendere la valutazione formativa e continua.

Scalabilità del sistema
Estendere UDA-system ad altre discipline e contesti scolastici, verificando la trasferibilità del modello e la sostenibilità del carico per il docente.



IL CODICE MOSTRA IL RISULTATO, LA REPOSITORY RACCONTA IL PERCORSO.

GRAZIE PER L'ATTENZIONE!

