Politecnico di Torino

Dipartimento di Ingegneria Gestionale e della Produzione

Laurea Magistrale in Ingegneria Gestionale

Anno accademico 2023/2024

Sessione di Laurea Dicembre 2024

Differenza semantica tra dati strutturati di progetti

Relatore: Prof. Alberto De Marco
Correlatore: Dott. Massimo Rebuglio

Candidato: Rodolfo Iannitti

Abstract

Il presente lavoro di tesi si focalizza sull'analisi e la comparazione di progetti strutturati tramite l'utilizzo della differenziazione semantica, una metodologia avanzata per mettere in luce le diverse caratteristiche tra varie versioni di progetti o progetti differenti. Il principale scopo di questa ricerca è stato creare un progetto in Python per confrontare strutture dati complesse relative a progetti al fine di individuare ed evidenziare le modifiche necessarie per passare da una versione di progetto all'altra. La tesi indaga l'importanza del confronto semantico, andando oltre la comparazione tradizionale orientata solo alle differenze sintattiche e cercando di rappresentare le variazioni in modo chiaro e significativo. Ciò permette di comprendere più a fondo come evolve un progetto, risultando particolarmente utile nel settore del project management e nella registrazione dei cambiamenti. Nell'introduzione della tesi viene anche presentata una panoramica sul project management per contestualizzare il significato e l'importanza del confronto tra progetti, insieme a una descrizione del semantic diffing. Si discute inoltre come la differenziazione semantica nelle pratiche di gestione del progetto possa semplificare la comprensione delle modifiche effettuate e il loro impatto sui processi commerciali. Nella ricerca, viene impiegata una metodologia che include l'analisi di vari casi studio, in cui viene utilizzato lo strumento creato per confrontare progetti reali o simulati ed evidenziare le variazioni strutturali e semantiche tra le versioni successive. Questi casi consentono di valutare l'efficacia dello strumento e di dimostrare che l'approccio sviluppato facilita la comprensione dei cambiamenti e aiuta nelle decisioni strategiche durante il ciclo di vita di un progetto. I risultati dimostrano come la ricostruzione dei passaggi evolutivi tra diverse versioni di progetto possa favorire la gestione delle risorse e ottimizzare i processi decisionali fornendo un utile strumento per comprendere e analizzare le modifiche tra le varie versioni dei progetti. Il progetto in Python è stato sviluppato garantendo flessibilità e adattabilità, consentendo quindi la possibilità di implementare nuovi tipi di strutture e per integrarsi con gli strumenti di gestione già esistenti. In conclusione, la tesi propone di esplorare futuri sviluppi del lavoro, come l'utilizzo di machine learning per migliorare l'interpretazione delle modifiche, la creazione di un'interfaccia grafica per rendere più intuitiva la visualizzazione delle modifiche e l'integrazione con software di project management esterni.

Indice

1. Introduzione al Project Management

Nell'era dell'innovazione tecnologica le organizzazioni affrontano sfide sempre più complesse per rimanere competitive sul mercato ed in questo contesto, la capacità di gestire efficacemente i progetti è diventata un elemento cruciale per il successo aziendale, di conseguenza, la figura del project manager svolge un ruolo centrale in questo scenario, fungendo da punto di collegamento tra la strategia organizzativa dell'azienda e la sua esecuzione operativa.

Il project management, considerato come campo di studio, ha origini che si collocano nei primi anni del XX secolo anche se elementi di gestione dei progetti, sebbene in maniera empirica, possono essere identificati in costruzioni antiche come le piramidi egiziane e la Grande Muraglia cinese. L'organizzazione dei progetti contemporanea è iniziata a svilupparsi durante l'era industriale, quando la complessità crescente delle iniziative richiedeva un metodo più organizzato e metodico. Dopo la Seconda Guerra Mondiale, c'è stata una notevole evoluzione nel campo quando settori come l'aerospazio e la difesa iniziarono a creare strumenti per gestire progetti complessi, infatti, in questo momento storico emersero metodologie come il Critical Path Method (CPM) e il Program Evaluation Review Technique (PERT) ideate per migliorare il suddetto campo di studio.

Negli anni '60 e '70, invece, si è affermata l'importanza delle procedure formali con la creazione di istituzioni come il Project Management Institute (PMI) nel 1969, che successivamente ha creato e diffuso il PMBOK Guide, una guida che racchiude le migliori pratiche del settore. Con il crescere della complessità dei progetti in settori come l'ingegneria, l'informatica e l'edilizia, la richiesta di formalizzare le metodologie di gestione dei progetti è diventata sempre più chiara. La fase industriale non ha segnato la fine dell'evoluzione del project management tanto è vero che con l'arrivo dell'era digitale e l'aumento dell'interconnessione tra mercati globali, il project management è stato ulteriormente trasformato. La comparsa delle metodologie Agile negli anni '90 ha portato a un cambiamento significativo nel modo di pianificare e condurre progetti, specialmente nel settore dello sviluppo software. L'approccio agile, che offre una gestione flessibile e iterativa, si è rivelato decisivo per adattarsi prontamente ai frequenti cambiamenti in contesti lavorativi dinamici e tecnologici. Sta di fatto che col passare del tempo, il project management ha trasformato la sua funzione da operativa a essenziale nella pianificazione strategica delle aziende. Al giorno d'oggi, le imprese comprendono che una buona gestione dei progetti non solo permette di rispettare i tempi e i budget, ma è anche fondamentale per allineare le attività progettuali agli obiettivi strategici dell'azienda. Nel contesto attuale, sempre più competitivo, le organizzazioni devono conciliare obiettivi ambiziosi con risorse limitate ed il project management è fondamentale per affrontare tali sfide. Le imprese non si limitano più a seguire procedure standard, ma sono in continuo consolidamento di nuove idee per introdurre sul mercato prodotti, servizi e progetti innovativi. Ciò comporta che un'efficace gestione dei progetti assicura un miglior utilizzo delle risorse, una corretta gestione dei rischi e la soddisfazione degli stakeholder per i risultati ottenuti. In sintesi, il project management è diventato una componente strategica che ha un impatto diretto sul successo competitivo di un'azienda.

Una figura critica nella buona riuscita di un progetto è il project manager (PM), la responsabilità principale di quest'ultimo è quella di raggiungere gli obiettivi del progetto assicurandosi che venga realizzato il prodotto (o il servizio) nel rispetto dei tempi, della qualità e dei costi stabiliti. Esso è responsabile della gestione del progetto dall'avvio fino alla messa in opera. Competenze come la comunicazione, la gestione delle risorse umane e delle risorse materiali sono abilità fondamentali al giorno d'oggi per un PM. Il ruolo chiave di questa figura è essenziale per coordinare con successo gli obiettivi, i tempi e le risorse in qualsiasi tipo e dimensione di progetto. Il responsabile del progetto organizza, supervisiona, dirige e monitora ogni fase del progetto,

fungendo da collegamento fra i gruppi di lavoro, i clienti e gli stakeholder. Definire gli obiettivi, gestire i rischi, allocare le risorse e comunicare efficacemente sono i compiti principali che deve svolgere. Dal punto di vista operativo, il project manager non si limita più a svolgere compiti progettuali, ma assume un ruolo attivo nel favorire l'applicazione delle strategie aziendali e nell'aiutare a ottenere vantaggi competitivi duraturi, infatti, un progetto ben gestito può portare a notevoli benefici economici, al potenziamento della qualità ed alla diminuzione dei costi e dei tempi di sviluppo, generando valore aggiunto all'azienda presso cui il PM lavora e alle aziende committenti. Questo scenario fa sì che il project management sia sempre più richiesto e apprezzato nei vari settori industriali. Come esposto, il project manager non è solo un coordinatore di attività, ma ha un ruolo fondamentale nel soddisfare le richieste degli stakeholder e nel gestire le risorse necessarie per completare il progetto. Il suo ruolo non si ferma alla supervisione delle attività delle risorse umane, ma comprende anche la direzione strategica del progetto nel suo complesso, assicurando che sia in linea con gli obiettivi dell'organizzazione.

Un progetto è caratterizzato dalla sua temporaneità e unicità, avendo un inizio e una fine ben definiti. In questa situazione, il responsabile del progetto ha il compito di gestire l'intero processo del progetto, partendo dalla fase iniziale di ideazione, passando per la pianificazione, l'implementazione, il monitoraggio e il controllo, fino alla conclusione. Una delle principali responsabilità del project manager è mantenere sotto controllo l'equilibrio tra gli elementi del progetto, quali ambito, tempi, costi e qualità per assicurare il raggiungimento degli obiettivi fissati. In aggiunta alla gestione pratica e alla soluzione dei problemi di tutti i giorni, il project manager influisce in modo strategico. È necessario quindi garantire che il progetto apporti benefici all'azienda e sia conforme alle strategie aziendali. Questo implica che è cruciale comprendere appieno le priorità, le risorse e le esigenze degli stakeholder dell'azienda. Ciò che differenzia un project manager efficace è la capacità di collegare le operazioni tattiche agli obiettivi strategici.

1.1. L'evoluzione storica del Project Management

L'evoluzione del Project Management rappresenta una lunga storia che si sviluppa attraverso secoli di innovazioni organizzative, sociali e tecnologiche, culminando nella formalizzazione della disciplina nel XX secolo. Nonostante la gestione strutturata dei progetti come la conosciamo oggi sia un concetto relativamente moderno, le pratiche che hanno dato vita a questa disciplina risalgono a migliaia di anni fa, quando grandi civiltà come quella egizia e romana realizzarono progetti monumentali. Lungo il percorso evolutivo, la rivoluzione industriale, i conflitti mondiali e lo sviluppo tecnologico hanno contribuito alla nascita di un corpus teorico e pratico che oggi è alla base delle moderne metodologie di project management. Le radici del project management possono essere rintracciate nelle antiche civiltà che riuscirono a completare opere di grande complessità. Esempi storici come la costruzione delle piramidi egizie, con la piramide di Cheope che si stima abbia coinvolto circa 100.000 lavoratori per vent'anni, e la Grande Muraglia cinese, mostrano l'esistenza di una primitiva, ma efficace, gestione organizzativa e logistica. Questi grandi progetti non avrebbero potuto essere completati senza un'attenta pianificazione e coordinamento delle risorse umane e materiali, implicando rudimentali pratiche di project management che, per quanto inconsapevoli, erano comunque fondamentali per il loro successo. In epoca romana, la costruzione di infrastrutture come acquedotti, anfiteatri e strade rappresentava un ulteriore esempio di gestione strutturata dei progetti. Il Colosseo, completato in circa dieci anni, e gli acquedotti che rifornivano la città di Roma di acqua potabile, richiedevano avanzate competenze ingegneristiche e una gestione accurata delle risorse, dimostrando come già in tempi antichi vi fosse una cultura nella gestione dei progetti sebbene non formalizzata.

L'avvento della Rivoluzione Industriale, alla fine del XVIII secolo, segnò una tappa fondamentale nell'evoluzione del project management, in quanto portò a cambiamenti significativi nell'organizzazione del lavoro e nello sviluppo di nuove tecnologie. Tuttavia, fu solo nei primi anni del XX secolo che iniziarono a emergere approcci scientifici alla gestione delle risorse, del tempo e dei processi di lavoro.

Una figura chiave in questa fase fu Frederick Taylor, considerato il padre del "management scientifico". Con la pubblicazione del suo trattato *The Principles of Scientific Management* (1911), Taylor introdusse una serie di principi che trasformarono radicalmente il modo di organizzare e ottimizzare il lavoro. Secondo Taylor, la produttività poteva essere aumentata non solo attraverso l'intensificazione degli sforzi dei lavoratori, ma tramite lo studio scientifico delle operazioni e la razionalizzazione delle attività. Il suo contributo più significativo fu l'introduzione del concetto di "one best way" (un solo modo migliore) per eseguire un'operazione, utilizzando metodi standardizzati per massimizzare l'efficienza. Poco dopo, Henry Gantt, collaboratore di Taylor, sviluppò il celebre *diagramma di Gantt*, uno strumento grafico per la pianificazione e il monitoraggio delle attività di un progetto. Questo diagramma, che visualizza la sequenza temporale delle attività e il loro stato di avanzamento, rimane uno degli strumenti più utilizzati nel project management moderno.

L'esplosione dell'industrializzazione e la crescente complessità dei progetti tecnologici e infrastrutturali nel XX secolo richiedevano metodi di gestione sempre più sofisticati. Un punto di svolta si verificò durante la Seconda Guerra Mondiale, quando il bisogno di gestire grandi progetti su scala nazionale portò all'introduzione di tecniche formali di project management. Un esempio emblematico è il *Progetto Manhattan*, avviato nel 1942 con l'obiettivo di sviluppare la prima bomba atomica. Questo progetto, diretto dal fisico Robert Oppenheimer, coinvolse migliaia di scienziati e tecnici, e rappresenta uno dei primi casi di applicazione strutturata di tecniche moderne di gestione dei progetti, combinando la pianificazione scientifica con l'organizzazione delle risorse umane e materiali. Parallelamente, in questo periodo furono introdotti nuovi strumenti come il *Program Evaluation and Review Technique* (PERT) e il *Critical Path Method* (CPM), sviluppati rispettivamente per la gestione del progetto Polaris della Marina statunitense e per la manutenzione degli impianti industriali della azienda DuPont. Questi strumenti si basavano sull'analisi delle dipendenze tra le attività di un progetto, consentendo di identificare il percorso critico e ottimizzare i tempi di completamento.

Con la fine della Seconda Guerra Mondiale, il project management iniziò a diffondersi in molteplici settori, inclusi l'edilizia, l'ingegneria, la difesa e, successivamente, l'informatica. Negli anni '60 e '70, l'importanza della pianificazione e del controllo dei progetti venne riconosciuta a livello globale, portando alla nascita di associazioni professionali come il *Project Management Institute* (PMI) nel 1969. Il PMI si prefissò l'obiettivo di sviluppare uno standard comune per il project management, che culminò nella pubblicazione della prima edizione del *Project Management Body of Knowledge* (PMBOK) nel 1987, un riferimento fondamentale che raccoglie i principi, i processi e le migliori pratiche per la gestione dei progetti. Contemporaneamente, si diffuse l'uso di metodologie basate su metriche di performance come il *Earned Value Management* (EVM), un approccio che integrava il controllo dei costi e dei tempi di un progetto, fornendo una visione più accurata dello stato di avanzamento e delle eventuali deviazioni rispetto ai piani. Gli anni '80 videro un'espansione delle tecniche di project management anche in nuovi settori come l'informatica e la produzione di software. L'introduzione dei personal computer permise di automatizzare molti processi di pianificazione e monitoraggio, rendendo più facile la gestione delle attività e delle risorse. Le tecnologie dell'informazione trasformarono radicalmente la gestione dei progetti, grazie alla possibilità di raccogliere e analizzare dati in tempo reale, accelerando i processi decisionali. La crescente globalizzazione e l'aumento della concorrenza mondiale portarono anche a una maggiore attenzione alla qualità e all'efficienza. L'approccio del *Total Quality Management* (TQM), insieme allo sviluppo di benchmark e standard internazionali, contribuì a far sì che la gestione dei progetti fosse sempre più orientata al miglioramento continuo e alla soddisfazione del cliente. Con l'ingresso nel XXI secolo, il project management ha continuato a evolversi, adattandosi a un contesto sempre più globalizzato e digitalizzato. Le tecnologie emergenti, come il cloud computing, la collaborazione online e i software avanzati per la gestione dei progetti, hanno reso possibile una comunicazione e un controllo a distanza senza precedenti. La crescente importanza della gestione dei rischi, della sostenibilità e della responsabilità sociale ha portato all'integrazione di nuovi strumenti e metodologie, come l'Agile Project Management, che permette una maggiore flessibilità e adattamento in contesti incerti e dinamici.

Oggi, il project management è una disciplina ben definita e rispettata, praticata in una vasta gamma di settori. La capacità di gestire progetti complessi è considerata essenziale non solo per le grandi organizzazioni, ma anche per le piccole e medie imprese, in quanto consente di affrontare la complessità crescente dei mercati e delle tecnologie moderne. Il Project Management, nato come un insieme di pratiche empiriche e soluzioni operative, si è evoluto in una disciplina scientifica e multidimensionale. Dalle piramidi dell'antico Egitto ai progetti tecnologici del XXI secolo, la storia del project management dimostra la sua importanza strategica per la realizzazione di progetti complessi e innovativi. Oggi, grazie all'integrazione di nuove tecnologie e metodologie, il project management si conferma come una disciplina centrale nella gestione del cambiamento e nell'innovazione organizzativa.

1.2. Metodologie, standard di riferimento

Le metodologie e gli standard internazionali rappresentano punti di riferimento essenziali per una gestione efficace ed efficiente dei progetti.

Tra questi, il PMBOK Guide (Project Management Body of Knowledge) sviluppato dal Project Management Institute (PMI) è una delle guide più conosciute a livello globale. Esso fornisce un insieme completo di best practice, processi e terminologie che aiutano i project manager a gestire i progetti in modo strutturato e coerente. Si basa su dieci aree di conoscenza e cinque gruppi di processi (Figura 1) che coprono l'intero ciclo di vita del progetto, offrendo un framework flessibile applicabile a vari settori e tipologie di progetti.

Aree di conoscenza	Gruppi di processi di project management				
	Gruppo di processi di avvio	Gruppo di processi di pianificazione	Gruppo di processi di esecuzione	Gruppo di processi di monitoraggio e controllo	Gruppo di processi di chiusura
4. Gestione dell'integrazione di progetto	4.1 Sviluppare il Project Charter	4.2 Sviluppare il piano di Project Management	4.3 Dirigere e gestire il lavoro del progetto 4.4 Gestire le conoscenze di progetto	4.5 Monitorare e controllare il lavoro del progetto 4.6 Eseguire il controllo integrato delle modifiche	4.7 Chiudere il progetto o una fase
5. Gestione dell'ambito del progetto		5.1 Pianificare la gestione dell'ambito 5.2 Raccogliere i requisiti 5.3 Definire l'ambito 5.4 Creare la WBS		5.5 Convalidare l'ambito 5.6 Controllare l'ambito	
6. Gestione della schedulazione del progetto		6.1 Pianificare la gestione della schedulazione 6.2 Definire le attività 6.3 Sequenzializzare le attività 6.4 Stimare le durate delle attività 6.5 Sviluppare la schedulazione		6.6 Controllare la schedulazione	
7. Gestione dei costi di progetto		7.1 Pianificare la gestione dei costi 7.2 Stimare i costi 7.3 Determinare il budget		7.4 Controllare i costi	
8. Gestione della qualità di progetto		8.1 Pianificare la gestione della qualità	8.2 Gestire la qualità	8.3 Controllare la qualità	
9. Gestione delle risorse del progetto		9.1 Pianificare la gestione delle risorse 9.2 Stimare le risorse per le attività	9.3 Acquisire le risorse 9.4 Sviluppare il gruppo di lavoro 9.5 Gestire il gruppo di lavoro	9.6 Controllare le risorse	
10. Gestione delle comunicazioni di progetto		10.1 Pianificare la gestione delle comunicazioni	10.2 Gestire le comunicazioni	10.3 Monitorare le comunicazioni	
11. Gestione dei rischi di progetto		11.1 Pianificare la gestione dei rischi 11.2 Identificare i rischi 11.3 Eseguire l'analisi qualitativa dei rischi 11.4 Eseguire l'analisi quantitativa dei rischi 11.5 Pianificare le risposte ai rischi	11.6 Eseguire le risposte ai rischi	11.7 Monitorare i rischi	
12. Gestione dell'approvvigionamento di progetto		12.1 Pianificare la gestione degli approvvigionamenti	12.2 Definire gli approvvigionamenti	12.3 Controllare gli approvvigionamenti	
13. Gestione degli stakeholder del progetto	13.1 Identificare gli stakeholder	13.2 Pianificare il coinvolgimento degli stakeholder	13.3 Gestire il coinvolgimento degli stakeholder	13.4 Monitorare il coinvolgimento degli stakeholder	

Figura 1: Aree di conoscenza e gruppi di processi del PMBOK

Le aree di conoscenza riguardano l'insieme di conoscenze e competenze necessarie per adempiere ad un insieme di specifiche finalità:

1. Gestione dell'integrazione di progetto:
   Questa area si occupa di tutti i processi necessari per garantire che i diversi aspetti del progetto siano ben coordinati e funzionino in armonia, permettendo così una gestione integrata delle attività.
2. Gestione dell'ambito di progetto:
   Include i processi che assicurano che il progetto contenga tutte le attività necessarie per raggiungere i suoi obiettivi, evitando di includere elementi superflui o non pertinenti.
3. Gestione della schedulazione di progetto:
   Racchiude i processi utili a garantire che il progetto segua un calendario prestabilito, completando le attività entro i tempi definiti.
4. Gestione dei costi di progetto:
   Comprende i processi necessari per garantire che il progetto rispetti il budget approvato, gestendo e controllando le spese dall'inizio alla fine.
5. Gestione della qualità di progetto:
   Questa area raggruppa i processi che servono a garantire che il progetto rispetti i requisiti per cui è stato avviato, soddisfacendo le aspettative del cliente e le specifiche definite.
6. Gestione delle risorse di progetto:
   Si occupa di assicurare l'utilizzo ottimale e più efficiente delle risorse coinvolte nel progetto, siano esse persone, strumenti, materiali o altre risorse necessarie.
7. Gestione della comunicazione di progetto:
   Copre i processi necessari a garantire una gestione efficace delle informazioni, includendo la loro creazione, raccolta, distribuzione e archiviazione durante il ciclo di vita del progetto.
8. Gestione dei rischi di progetto:
   Riguarda i processi che permettono di identificare e valutare i rischi associati al progetto, e di definire le strategie di risposta più appropriate per mitigarli o sfruttarli.
9. Gestione dell'approvvigionamento di progetto:
   Include i processi necessari per ottenere beni e servizi da terze parti, tramite contratti e accordi, al fine di supportare il raggiungimento degli obiettivi di progetto.
10. Gestione degli stakeholder di progetto:
    Questa area riguarda l'identificazione e la gestione di tutte le persone o gruppi che possono influenzare o essere influenzati dal progetto. Include la comprensione delle loro aspettative e l'elaborazione di strategie per coinvolgerli efficacemente nelle decisioni e nelle attività di progetto.

Mentre i cinque gruppi di processi sono:

1. Processi di avvio:
   Includono tutte le attività necessarie per selezionare un progetto in linea con gli obiettivi aziendali, preparare un business case e nominare un Project Manager, trasferendo a quest'ultimo informazioni dettagliate sugli obiettivi e sulle modalità di gestione del progetto.
2. Processi di pianificazione:
   Hanno l'obiettivo di definire chiaramente l'ambito e i risultati del progetto, specificare i requisiti di ogni risultato previsto, e creare il Piano di Project Management che copre dettagli su tempi, risorse, costi, qualità, rischi, comunicazioni e approvvigionamenti.
3. Processi di esecuzione:
   Nel PMBOK, questi processi riguardano la gestione e lo sviluppo del team di progetto, la produzione dei risultati concordati, la verifica del rispetto degli standard di produzione e la gestione del processo di consegna al cliente finale.

4. Processi di monitoraggio e controllo:
   Si occupano di valutare lo stato di avanzamento del progetto, gestire eventuali modifiche e controllare la qualità del lavoro svolto, assicurando che tutto proceda secondo i piani.
5. Processi di chiusura:
   Comprendono le attività necessarie per rilasciare le risorse impiegate e concludere formalmente il progetto, compresa la chiusura dei contratti e la gestione delle commesse.

Ciascun processo fa quindi riferimento ad un'area di conoscenza e a un gruppo di processi cui appartiene. È inoltre descritto in termini di:

- Input (informazioni, piani, strumenti e prodotti di attività precedenti)
- Tecniche (modalità di elaborazione degli input)
- Output (prodotti e documenti ottenuti attraverso l'applicazione delle tecniche descritte)

Di seguito andremo ad esplorare più nel dettaglio ii cinque gruppi di processi (Figura 2).

Figura 2: Gruppi di processi

Alla base del Project Management ci sono i processi di gestione. Si definiscono processi le rappresentazioni schematiche delle sequenze di azioni e attività tra loro collegate in un flusso che vengono eseguite fino al loro esaurimento per ottenere i risultati desiderati.

In particolare, i processi di project management devono essere definiti dalle organizzazioni in funzione della propria struttura organizzativa, del modello di business e della tipologia di prodotto.

I processi individuati dalle organizzazioni per la gestione delle proprie attività vengono, generalmente, suddivisi in cinque gruppi che collegano l'inizio e la fine del progetto: avvio, pianificazione, monitoraggio e controllo, esecuzione e chiusura. Le fasi non sono sequenziali ma interagiscono tra loro, si sovrappongono e, nel corso della loro esecuzione, possono essere anche modificate.

1. Processo di avvio del progetto

In questa fase viene definito ed autorizzato il progetto. È il momento in cui vengono fissati i requisiti del progetto, ne vengono stabiliti gli obiettivi, in termini di tempo, costo e qualità, si definiscono gli stakeholder interessati e non da ultimo viene nominato il project manager che avrà la completa responsabilità del progetto.

Durante la fase di avvio, il progetto viene definito a livello generale al fine di dimostrarne il valore commerciale. Una volta ottenuto il consenso da parte degli stakeholder principali e verificata la fattibilità del progetto, si procede alla fase di pianificazione. In questa fase vengono dettagliati obiettivi specifici, deliverable e roadmap del progetto. L'obiettivo è fornire informazioni sufficienti per ottenere l'approvazione durante l'avvio, per poi approfondire i dettagli nella pianificazione una volta ricevuta l'autorizzazione a procedere. L'inizio di

un nuovo progetto può essere entusiasmante per le persone coinvolte, ma è fondamentale assicurarsi che l'iniziativa porti valore prima di avanzare alla fase di pianificazione. La fase di avvio offre un approccio strutturato per presentare il business case del progetto e dimostrare la fattibilità del lavoro previsto. Inoltre, coinvolge gli stakeholder sin dalle prime fasi, permettendo di assicurarsi risorse essenziali, aumentare la visibilità del progetto ed evitare ostacoli costosi durante la sua esecuzione. Una volta concepita un'idea promettente per una nuova iniziativa, è possibile affrontare la prima fase della gestione del progetto seguendo quattro passaggi chiave per costruire solide fondamenta.

Nel primo passaggio, si espone la necessità del progetto e i vantaggi attesi. Ciò può avvenire tramite la creazione di un project charter o di un business case. Entrambi i documenti mirano a definire i dettagli chiave del progetto e a presentare l'iniziativa agli stakeholder.

Un project charter delinea l'importanza del progetto, il suo contenuto e i membri coinvolti, rispondendo ai seguenti elementi:

- Perché: obiettivi e scopo del progetto.
- Cosa: ambito del progetto, inclusa una bozza del budget.
- Chi: stakeholder principali, sponsor del progetto e risorse umane coinvolte.

Il passo successivo consiste nel determinare i gruppi di persone che hanno interesse nella buona riuscita del progetto, ovvero gli stakeholder. Gli stakeholder principali sono individui che influenzano significativamente gli esiti del progetto, essi possono essere dirigenti, sponsor e per identificarli, è utile porsi le seguenti domande:

- Chi deve approvare il progetto?
- Chi fornirà le risorse necessarie?
- Chi può influenzare il progetto?

È molto importante eseguire un'analisi degli stakeholder, suddividendoli, tramite una mappa, in quattro categorie principali: alta influenza e alto interesse, alta influenza e basso interesse, bassa influenza e alto interesse, bassa influenza e basso interesse. Gli individui con alta influenza e alto interesse sono generalmente i portatori di interesse principali, i quali, dovrebbero approvare il progetto durante la fase di avvio. Inoltre, è opportuno identificare altre persone potenzialmente interessate o influenzate dal progetto in quanto, sebbene non sia necessario il loro consenso formale, informarli anticipatamente può prevenire impatti negativi sulle attività e potrebbe offrire ulteriori supporti in termini di approfondimenti o risorse. Coinvolgere gli stakeholder fin dall'inizio non solo facilita l'ottenimento di approvazioni, supporto e risorse, ma aumenta anche la visibilità del progetto e previene ostacoli costosi durante il suo ciclo di vita.

Dopo aver presentato il progetto e dimostrato il suo valore in relazione al piano strategico aziendale, viene redatto uno studio di fattibilità per confermare la realizzabilità con le risorse disponibili. Gli studi di fattibilità sono generalmente riservati a progetti di grandi dimensioni che richiedono notevoli risorse aziendali. Per progetti minori con impatto limitato, questo passaggio può essere omesso, soprattutto se esistono progetti precedenti simili.

In seguito all'approvazione e la conferma della fattibilità del progetto, si può procedere all'organizzazione delle risorse umane necessarie, dello spazio di lavoro e degli strumenti necessari.

2. Processo di pianificazione

Attraverso la raccolta di informazioni da più fonti viene elaborato un piano di progetto. I requisiti consentono, infatti, l'identificazione dei deliverable che verranno prodotti dal progetto e da essi sarà possibile definire l'elenco delle attività che li costituiscono.

Il processo di pianificazione consente di determinare:

- la data finale del progetto e tutte le milestone intermedie
- gli stakeholder e il relativo piano delle comunicazioni
- il piano dei rischi e della qualità
- i fornitori
- le modalità con cui verranno gestite le risorse
- i costi di progetto.

Per ogni attività verrà definita una "baseline", ovvero un piano temporale con cui verranno effettuate le attività di verifica durante il monitoraggio del progetto.

La fase di pianificazione è un passaggio essenziale per garantire il buon esito di un progetto, si delinea un percorso chiaro per raggiungere gli obiettivi prefissati, identificando le risorse necessarie, prevedendo potenziali ostacoli e stabilendo una strategia operativa. Questo processo porta alla redazione di vari documenti fondamentali, che serviranno da guida per l'intero ciclo di vita del progetto. Il punto di partenza è la definizione degli obiettivi, che devono essere precisi, misurabili e allineati alle aspettative dei principali portatori di interesse (stakeholder). Successivamente, si procede con l'analisi delle risorse necessarie, identificando ed organizzando le persone coinvolte nel progetto, le attrezzature, i materiali e altre risorse importanti. Questo porta alla creazione di un piano di gestione delle risorse (Resource Management Plan), che specifica come verranno impiegate e distribuite le risorse durante il progetto. Inoltre, viene redatto un documento che descrive una scomposizione dettagliata del lavoro da svolgere, la Work Breakdown Structure (WBS), suddividendo il progetto in parti più piccole e gestibili, facilitando così l'assegnazione di compiti e responsabilità. Un importante aspetto della pianificazione è la stima dei tempi e dei costi, attraverso l'elaborazione di un cronoprogramma si definisce la sequenza delle attività, la loro durata e le dipendenze tra le stesse. Il cronoprogramma viene spesso visualizzato tramite strumenti come il diagramma di Gantt. Parallelamente, viene elaborato il budget di progetto, che fornisce una previsione accurata dei costi, includendo riserve (contingency) per eventuali imprevisti. Questi documenti aiutano a tenere sotto controllo tempi e risorse, evitando ritardi e sforamenti di budget. La gestione dei rischi è un'altra componente della pianificazione, infatti viene redatto un registro dei rischi, dove si identificano i potenziali problemi che potrebbero compromettere il successo del progetto. Ogni rischio viene valutato in termini di probabilità e impatto, e vengono delineate strategie di mitigazione o piani alternativi per affrontarli, qualora si presentassero.

Un aspetto, spesso trascurato, è quello della comunicazione. Un piano di comunicazione efficace presenta le linee generali di come verranno comunicate agli stakeholder principali le informazioni relative al progetto in corso. Viene pertanto chiarito quali saranno i canali di comunicazione da utilizzare e con quale frequenza i vari dettagli dovranno essere comunicati. Inoltre, con un piano di comunicazione ben redatto, ogni persona che fa parte del gruppo di lavoro sa a chi rivolgersi e quale canale di comunicazione utilizzare.

Infine, viene sviluppato un piano per la gestione della qualità, che stabilisce gli standard e le soglie di tolleranza che il deliverable del progetto devono rispettare. Il piano include i criteri di controllo della qualità e le modalità di verifica, per assicurarsi che i risultati intermedi del progetto siano conformi ai requisiti previsti.

3. Processo di esecuzione

Il processo di esecuzione consente di portare a termine il lavoro definito durante la fase di pianificazione. Infatti, viene messo in esecuzione il piano di progetto e utilizzata la maggior parte del budget a disposizione. Se dovessero essere richieste delle modifiche durante questo periodo, il costo delle modifiche sarebbe superiore ad analoghe richieste effettuate durante la fase di pianificazione.

Durante questo stadio, i documenti del progetto vengono aggiornati e dove necessario alcune attività vengono ripianificate e vengono determinate le nuove baseline. Gli aggiornamenti possono includere i cambiamenti nelle durate previste delle attività, cambiamenti relativi alla disponibilità e all'impiego delle risorse richieste (umane e materiali). Potrebbero verificarsi rischi non preventivati e quindi in questa eventualità occorre aggiornare il documento di rischio stipulato nelle fasi precedenti. Potrebbe capitare in questa fase che si verifichino degli scostamenti dei costi e dei tempi causati per esempio dalla mancanza delle risorse necessarie. In questi casi il problema va analizzato nei dettagli e bisogna intraprendere le strategie per la risposta al problema in tempo, prima che il problema diventi grave. Il lavoro definito nella fase di definizione del progetto dovrà essere portato a termine con il successo e nei limiti di tempo, di costo e di impegno delle risorse.

Le principali responsabilità del Project Manager in questa fase sono:

- Acquisire, formare e gestire i membri del team di progetto;
- Eseguire ed implementare le modifiche approvate;
- Il controllo della realizzazione dei deliverables previsti;
- La verifica dell'implementazione delle azioni correttive;
- Raccogliere costantemente i dati sulle prestazioni del lavoro inerenti al lavoro che è stato realizzato effettivamente sino a quel momento come, ad esempio, l'avanzamento temporale delle attività (date di inizio e fine effettive), l'avanzamento economico (costi effettivi), i lavori completati, ecc.

Il Project Manager durante la conduzione dell'esecuzione del progetto dovrà essere supportato dal team del progetto e da alcuni principali stakeholder come colleghi, consulenti, esperti, sponsor o il committente (cliente) stesso

I principali processi nella fase di esecuzione sono:

- la conduzione del progetto e la gestione dei cambiamenti
- la gestione delle comunicazioni con gli stakeholder
- la gestione degli approvvigionamenti
- garantire la qualità del progetto
- la gestione dei conflitti
- reporting dello stato avanzamento lavori

4. Processo di monitoraggio e controllo

Il monitoraggio e controllo è caratterizzato da processi attuati per osservare e misurare l'esecuzione del progetto in modo da indentificare in tempo i rischi, gli eventuali ritardi e i potenziali problemi. Quando necessario, è importante intraprendere le azioni correttive volte a rimettere il progetto in linea con i propri obiettivi. Il presupposto principale di questa attività consiste nella possibilità di osservare e misurare regolarmente la

produttività del progetto, identificandone gli scostamenti (chiamati varianze nella teoria dell'Earned Value Management) rispetto alla produttività assunta in fase di pianificazione

Il monitoraggio e controllo include:

- misurazione dell'avanzamento delle attività del progetto (dove ci troviamo);
- confronto con le previsioni del piano di progetti che costituiscono la baseline del progetto (dove dovremmo essere);
- definizione e controllo delle azioni correttive volte a rimuovere i problemi e/o evitare i rischi in modo da ristabilire la produttività desiderata del progetto (come dobbiamo continuare);
- vigilanza verso l'adozione implicita di variazioni di scopo (change request) non concordate e approvate.

Le misurazioni effettuate presuppongono la definizione di metriche di progetto definite all'inizio del progetto di cui fanno parte i costi e la quantità di lavoro erogato (effort).

In questa fase l'attività di identificazione dei problemi e dei rischi richiede il continuo supporto degli utenti chiave; la velocità di identificazione e gestione delle problematiche costituisce un indicatore importante per determinare lo stato di salute del progetto.

In determinati ambiti, ma soprattutto su progetti di media o grande complessità è quasi impossibile che non si verifichino variazioni di scopo. Le variazioni possono essere costituite per esempio da:

- cambiamenti derivanti dalla necessità di risolvere criticità di progettazione;
- condizioni operative riscontrate che differiscono da quelle previste (ad esempio, nel settore edile: caratteristiche del terreno diverse da quelle stimate);
- difficoltà nel procurarsi i materiali necessari;
- modifiche richieste dagli appaltatori o introdotte da soggetti esterni

Le variazioni di contesto nel progetto necessitano di essere documentate, mantenendo evidenza delle differenze con gli scopi originali del progetto, questo consente al committente di capire chiaramente cosa è cambiato e soprattutto trovare le giustificazioni per i diversi tempi e costi di realizzazione del progetto. Le variazioni di scopo possono richiedere anche variazioni al contratto stabilito tra le parti per la realizzazione del progetto.

Il principale strumento della fase di monitoraggio e controllo è il rapporto di Stato Avanzamento Lavori (SAL), noto anche come Project Status Report. Questo documento è redatto dal project manager per aggiornare gli altri soggetti coinvolti nel processo di monitoraggio sullo stato di avanzamento del progetto. Le informazioni chiave contenute nel SAL includono:

- deliverable consegnati (in confronto al piano previsto);
- costi e tempi maturati (in confronto al piano);
- problemi e rischi, sia quelli ancora aperti che quelli già risolti;
- azioni in corso ("chi fa cosa entro quando") per affrontare i problemi e i rischi ancora aperti.

I costi (distinti tra maturati, effettivi e pianificati) sono spesso espressi utilizzando il costo standard, seguendo un approccio di contabilità industriale, piuttosto che il costo puntuale, che rappresenta il costo preciso, tipico della contabilità generale.

5. Processo di chiusura

Si attuano una volta che è stato approvato il deliverable finale del progetto o dopo che il progetto è stato chiuso. In questa fase, oltre a chiudere i contratti con i fornitori, dovranno essere raccolte e catalogate tutte le informazioni. Non è raro assistere a progetti che si concludono ignorandola, ritenendola uno spreco di energie e di tempo, non rendendosi conto però, che una corretta raccolta della documentazione potrebbe garantire dei risparmi nella gestione di futuri progetti analoghi nei contenuti, in cui si potrebbero utilizzare parti di precedenti progetti. In quest'ottica, un elemento fondamentale è la raccolta delle "lesson learned": tutto ciò che di positivo o negativo si è verificato nell'esecuzione del lavoro, lezioni, che vanno ad arricchire la cultura aziendale. Il Project Management è infatti una disciplina professionale ed organizzativa fortemente caratterizzata dall'esperienza, che si acquisisce solo lavorando, sbagliando e riflettendo, che cresce e si evolve in qualsiasi organizzazione. Queste ultime difatti possono acquisire esperienza solo monitorando costantemente i propri processi produttivi per creare vantaggi competitivi che li distingueranno dai concorrenti.

Quindi, come menzionato precedentemente, la guida PMBOK individua diverse aree di conoscenza che un project manager deve padroneggiare, come l'integrazione del progetto, la gestione dell'ambito, del tempo, dei costi, della qualità, delle risorse, delle comunicazioni, dei rischi, degli approvvigionamenti e degli stakeholder. Ogni area rappresenta un elemento chiave per il coordinamento e la gestione efficace di tutte le attività del progetto. Recentemente, la guida PMBOK ha subito un cambiamento significativo, spostando l'enfasi da un approccio puramente basato sui processi a uno più centrato sui principi e sui domini di performance. Questo cambiamento riflette l'esigenza di una maggiore adattabilità e personalizzazione nella gestione dei progetti, tenendo conto della crescente complessità e variabilità degli ambienti in cui i progetti vengono portati avanti.

Un altro standard internazionale rilevante è il PRINCE2 (Projects IN Controlled Environments), una metodologia sviluppata da Axelos, una joint venture tra il governo britannico e la società Capita ampiamente utilizzata in Europa. PRINCE2 è basata su sette principi, sette temi e sette processi che guidano il project manager attraverso tutte le fasi del progetto, focalizzandosi sulla gestione per eccezioni e sulla responsabilizzazione dei gruppi di lavoro. Questa metodologia enfatizza la chiarezza dei ruoli e delle responsabilità, nonché la necessità di adattare la gestione del progetto al contesto specifico.

Le metodologie Agile rappresentano un approccio alternativo e complementare, particolarmente adatto a contesti caratterizzati da elevata incertezza e cambiamenti frequenti. La metodologia Agile (o sviluppo Agile del software, in inglese Agile Software Development, abbreviato in ASD) è un approccio all'ingegneria del software che comprende una serie di metodi di sviluppo nati nei primi anni 2000. Questi metodi si basano su principi condivisi che derivano direttamente o indirettamente dal "Manifesto per lo Sviluppo Agile del Software" (Manifesto for Agile Software Development), pubblicato nel 2001 da Kent Beck, Robert C. Martin, Martin Fowler e altri. Questo manifesto ha definito i valori e i principi fondamentali alla base dello sviluppo Agile. Tra queste, Scrum, Kanban e Extreme Programming (XP) sono ampiamente adottate. L'approccio Agile si basa su iterazioni brevi e incrementali, coinvolgendo attivamente il cliente e promuovendo la collaborazione all'interno del gruppo di lavoro. Questo permette di adattarsi rapidamente ai cambiamenti nei requisiti e di fornire valore aggiunto in tempi ridotti.

1.3. Competenze chiave del Project Manager

Il ruolo del project manager non si limita all'applicazione di competenze tecniche per la gestione dei progetti; esso richiede anche una solida capacità di leadership e una visione strategica. Il PMI Talent Triangle sottolinea ad esempio, l'importanza di un equilibrio tra competenze tecniche di gestione dei progetti (Technical Project Management), capacità di leadership per guidare e motivare la squadra di lavoro e conoscenza strategica e di business (Strategic and Business Management) per allineare il progetto agli obiettivi aziendali. Secondo Harold Kerzner, il project manager moderno deve essere un facilitatore, promotore di innovazione e agente di cambiamento, in grado di navigare tra le dinamiche organizzative e influenzare positivamente gli stakeholder a ogni livello. Anche la letteratura accademica evidenzia l'importanza delle soft skills per il successo del project manager. Infatti, Müller e Turner hanno evidenziato come la comunicazione efficace, la gestione dei conflitti e la leadership in situazioni complesse siano competenze chiave per il successo dei progetti. In un contesto sempre più globale, è fondamentale che il project manager sia culturalmente competente, capace di gestire gruppi di lavoro distribuiti in diverse parti del mondo, e che possa sfruttare a proprio vantaggio nuove modalità di lavoro come lo smartworking, sviluppando una solida comprensione delle dinamiche interculturali e delle migliori pratiche per la gestione remota.

Queste considerazioni portano alla conclusione che il project manager sia una figura poliedrica che deve saper combinare competenze tecniche, capacità di leadership e una visione strategica. Metodologie e guide di riferimento riconosciute a livello internazionale, come quelle descritte in precedenza offrono strumenti e contesti fondamentali per affrontare le sfide della gestione dei progetti, sia nella pratica che nella teoria. Il project manager deve possedere una vasta gamma di competenze che possono essere suddivise in due grandi categorie: competenze tecniche e competenze trasversali (o soft skills). Tra le competenze tecniche, rientrano la capacità di pianificare e gestire le risorse, la gestione del rischio, che implica identificare e rispondere a potenziali problemi prima che diventino critici, e il monitoraggio continuo del progetto per verificarne lo stato e assicurarsi che gli obiettivi vengano raggiunti. Queste attività richiedono l'utilizzo di strumenti di gestione dei progetti, come software dedicati, come ad esempio Microsoft Project o Jira, i quali permettono di avere una visione chiara e sempre aggiornata dell'andamento del lavoro. Tra le competenze trasversali, invece, spiccano la leadership, utile per motivare i colleghi e creare un ambiente di lavoro positivo e collaborativo, la comunicazione efficace, essenziale per mantenere tutti allineati e per gestire le aspettative degli stakeholder, e la gestione del tempo, che implica saper allocare correttamente il tempo tra diverse attività per bilanciare gli obiettivi quotidiani e strategici. Anche la capacità di problem solving e la negoziazione sono fondamentali: il project manager deve saper risolvere problemi in modo efficace e negoziare con stakeholder e team per trovare compromessi che non compromettano la qualità del progetto.

Dunque, il project manager moderno non è solo un esperto delle metodologie di gestione dei progetti, ma anche un leader in grado di motivare le persone, risolvere problemi e affrontare situazioni impreviste con creatività e prontezza. Di fatto, la combinazione tra competenze tecniche e trasversali rende questa figura professionale indispensabile per il successo di qualsiasi progetto, in qualsivoglia settore economico.

2. Algoritmi di differenziazione semantica

2.1. Introduzione

Gli algoritmi di differenziazione semantica sono uno degli strumenti più avanzati per quanto riguarda l'analisi del codice e il confronto di diverse versioni di documenti o programmi. La necessità di distinguere tra modifiche testuali e modifiche che influenzano effettivamente la logica del codice ha causato un miglioramento continuo delle tecniche superando il confronto classico di linee di codice, tipico del diffing testuale. Il confronto classico, come ad esempio il comando diff, opera a livello di linee di testo indicando quali righe sono state aggiunte, eliminate o modificate. Tuttavia, questi strumenti non colgono la semantica delle modifiche, ovvero il reale significato che le modifiche hanno sulle logiche o nel comportamento dell'applicazione, infatti, un cambiamento può alterare profondamente il comportamento di un processo anche senza molte modifiche a livello di singole righe di testo, oppure potrebbe accedere il viceversa: un aggiornamento strutturale del codice sorgente potrebbe alterare in maniera minima la logica funzionale dell'applicazione. Di fatto, gli algoritmi di semantic diffing puntano a risolvere questo ostacolo fornendo una visione più accurata dell'informazione e cioè comprendendo il contesto e significato del codice. Tale metodo è estremamente importante nel contesto della manutenzione del codice, in cui è fondamentale comprendere se e come le modifiche possano influenzare il comportamento di una applicazione. Di seguito saranno esposte le sfide, campi di applicazione in cui questa materia di studio viene utilizzata ed esempi pratici delle applicazioni nel mondo lavorativo ed accademico.

2.2. Funzionamento degli algoritmi di differenziazione semantica

L'obiettivo degli algoritmi di differenziazione semantica è dunque quello di determinare il significato funzionale delle modifiche e di analizzare come esse impattino il comportamento generale del software. Il semantic diffing utilizza approcci strutturali e concettuali per identificare i cambiamenti. Lo strumento principale per questi algoritmi è l'analisi dell'Abstract Syntax Tree, una rappresentazione strutturata del codice sorgente, in cui ciascun nodo rappresenta un costrutto sintattico specifico (ad esempio variabili, cicli, dichiarazioni di funzioni), infatti, comparando gli alberi sintattici tra due versioni di un programma, è possibile rilevare cambiamenti a livello strutturale piuttosto che semplicemente testuale. Ciò implica che modifiche equivalenti a livello funzionale, ma realizzate con sintassi differenti, possono essere identificate come semanticamente equivalenti, migliorando significativamente la comprensione delle modifiche apportate. Ad esempio, il rinominare delle variabili o la ristrutturazione di cicli "for" in cicli "while" possono essere modifiche significative a livello sintattico ma semanticamente equivalenti, quindi, grazie a strumenti di semantic diffing, queste modifiche vengono rilevate come cambiamenti formali senza un impatto funzionale. Oltre all'analisi dell'AST, gli algoritmi utilizzano metodi di analisi del flusso di controllo e analisi del flusso dei dati per comprendere come le informazioni si propagano all'interno di un programma e per determinare se le modifiche influenzano il comportamento effettivo dell'applicazione. L'analisi del flusso di controllo modella l'esecuzione di un programma tramite grafi di controllo verificando se le modifiche alterano i percorsi di esecuzione. Mentre, l'analisi del flusso dei dati, invece, permette di tracciare la propagazione e l'uso delle variabili per verificare se cambiamenti nelle assegnazioni o nelle dipendenze possano alterare il comportamento del software. Le analisi sopra descritte consentono di comprendere se una modifica pur non alterando direttamente le righe di codice, abbia un impatto indiretto su altre parti del sistema, evidenziando potenziali problemi o regressioni.

2.3. Campi di applicazione

Gli algoritmi di differenziazione semantica hanno applicazione in numerosi settori soprattutto dove è essenziale comprendere l'impatto reale delle modifiche a livello "comportamentale". Nel settore del software gli algoritmi permettono ai programmatori di individuare le differenze che influenzano direttamente il

comportamento del programma evitando di considerare modifiche che potrebbero essere non rilevanti come quelle relative alla formattazione del codice. Un esempio significativo è durante il processo di fusione (merging) di branch paralleli: gli algoritmi di semantic diffing possono ridurre i conflitti derivanti da modifiche contemporanee che influiscono sul comportamento del codice. Il merge di branch distinti è una fase critica nello sviluppo collaborativo, in cui più sviluppatori lavorano su diverse parti di un progetto in parallelo, infatti, i conflitti durante una fase di merge possono portare a perdita di tempo, errori e malintesi. Proprio per questo il semantic diffing aiuta a identificare situazioni in cui due modifiche apparentemente diverse non influenzano il comportamento globale evitando conflitti non necessari e gestendo in maniera più efficiente il merge di grandi progetti. Ciò comporta la riduzione delle risoluzioni manuali nei conflitti durante un merge e garantisce che le modifiche semantiche rilevanti vengano gestite correttamente. Il semantic diffing non entra in gioco solo nello sviluppo software, nel contesto della documentazione gli algoritmi possono essere utilizzati per confrontare documenti, articoli o manuali tecnici con lo scopo di identificare modifiche che hanno un impatto sul significato complessivo del testo. Ad esempio, nei documenti legali è fondamentale comprendere quali modifiche alterino il significato delle clausole. Inoltre, nel contesto accademico gli algoritmi di differenziazione semantica possono essere utilizzati per confrontare versioni successive di articoli o per analizzare revisioni tra diverse stesure di una ricerca: questo tipo di analisi permette di capire come il contenuto e le conclusioni di un articolo siano evolute nel tempo, aiutando i revisori e gli editori a focalizzarsi sui cambiamenti che influenzano il contributo scientifico complessivo.

2.4. Lavori rilevanti

Di seguito viene fornita una breve descrizione di prodotti e progetti rilevanti, utilizzati in ambito lavorativo, presentati in lavori accademici o disponibili come progetti open-source in rete che affrontano temi riguardanti gli algoritmi di differenziazione semantica.

2.4.1. Graphtage

Graphtage è una utility che è possibile utilizzare da riga di comando o come libreria che è possibile importare in Python. Viene adottata per il confronto semantico e l'unione di strutture ad albero come file JSON, JSON5, XML, HTML, YAML e TOML. Il suo nome è la combinazione tra “graph” e “graftage” (ovvero la pratica orticola di unire due alberi insieme in modo che crescano come uno solo). Graphtage permette di vedere cosa c'è di diverso tra due file in modo rapido e semplice ma non semplicemente come un prodotto che rileva le differenze linea su linea, infatti il tool è semanticamente “consapevole”, il che gli consente di mappare le differenze tra strutture non ordinate come dizionari di JSON e tag di elementi XML; tuttavia, può anche confrontare file di due formati diversi. I formati di file ad albero stanno diventando sempre più comuni come mezzo per trasmettere e archiviare dati, basti pensare ai tanti servizi REST API che vengono sviluppati ogni giorno in giro per il mondo, di conseguenza uno strumento come Graphtage può essere molto utile nel confronto e nell’analisi delle sopra citate strutture ad albero. Una nota a margine per ciò che concerne il lavoro che presenterò nelle pagine successive. In principio, durante lo sviluppo dell’idea, avevo pensato di trasformare le strutture dati che descrivevano i progetti in formato ad albero, in particolare in JSON, per poi integrare quest’ultimi con Graphtage ed analizzare le differenze. Purtroppo, il tool non si è rivelato adatto, in quanto i risultati risultavano poco leggibili, detto ciò, la utility è davvero molto interessante per comprendere il significato di algoritmo di differenziazione semantica e per confrontare strutture dati ad albero, a patto che non contengano informazioni molto complesse come quelle che possono essere contenute in un progetto, inteso dal punto di vista gestionale.

2.4.2. ChangeDistiller

ChangeDistiller è uno strumento di analisi e differenziazione del codice sorgente per le applicazioni Java, utilizza una tecnica di differenziazione incentrata sugli alberi sintattici astratti (AST) che rappresentano il codice sorgente in forma strutturata ad albero. Consente ai membri del team di sviluppo di scorrere le modifiche apportate tra le versioni precedenti e attuali del codice Java. È particolarmente utile quando diverse persone del gruppo lavorano contemporaneamente sullo stesso codice e per i grandi progetti in cui i file di codice Java cambiano frequentemente. Praticamente, l'algoritmo crea uno “script di modifica” che descrive le operazioni necessarie per trasformare un AST più vecchio in uno più recente usando operazioni di base come l'inserimento, l'eliminazione, lo spostamento e l'aggiornamento dei nodi AST. Per interpretare queste modifiche l’algoritmo utilizza una tassonomia delle modifiche al codice sorgente che descrive i tipi di cambiamento a partire dalle operazioni sull'AST. Questa tassonomia traduce lo script di modifica in cambiamenti concreti al codice e definisce il “livello di significatività” di ciascuna modifica, ossia l'impatto che potrebbe avere sulle altre parti del codice e se potrebbe influire sulla funzionalità generale. Ogni tipo di modifica viene classificato in uno dei seguenti livelli: nessuno, basso, medio, alto o cruciale e grazie alle informazioni fornite dagli AST, ChangeDistiller permette di ottenere dettagli precisi su ogni modifica incluse le specifiche entità del codice coinvolte e la loro posizione rendendo possibile una comprensione dettagliata delle modifiche effettuate al codice sorgente. ChangeDistiller è progettato per il linguaggio Java e mette a disposizione un'API comoda per estrarre e analizzare in dettaglio le modifiche del codice.

3. Impostazione del problema: sfide attuali e proposta di soluzione

Come affermato nell'introduzione, la gestione efficace dei progetti è una componente imprescindibile per il successo in svariati settori, dall'ingegneria del software, all'edilizia, fino alla ricerca scientifica. Uno degli aspetti più complessi nella gestione dei progetti è la tracciabilità e la ricostruzione delle modifiche progettuali nel corso del tempo od anche il confronto tra progetti simili fra loro per far in modo di evidenziare le differenze. La capacità di comprendere come un progetto è cambiato è fondamentale per garantire la coerenza, l'efficienza e la qualità dei risultati finali.

La ricostruzione delle modifiche progettuali presenta diverse sfide intrinseche:

1. Complessità crescente nei progetti: al giorno d'oggi i progetti tendono ad essere altamente complessi, coinvolgendo numerose attività interdipendenti e un vasto assortimento di risorse umane e materiali. Questa complessità rende difficile monitorare ogni cambiamento e comprendere il suo impatto sull'intero progetto.
2. Mancanza di tracciabilità: spesso, le modifiche ai progetti vengono registrate in modo non uniforme o frammentario, utilizzando diversi strumenti che non comunicano tra loro. Questa mancanza di tracciabilità centralizzata impedisce una visione a tutto tondo delle modifiche apportate.
3. Errore umano e comunicazione inefficiente: la comunicazione fra i gruppi di lavoro può essere soggetta a incomprensioni e/o omissioni, portando ad errori nella documentazione delle modifiche. Questo può causare confusione e ritardi nelle fasi successive del progetto.
4. Valutazione dell'impatto delle modifiche: senza strumenti adeguati, è difficile valutare l'impatto delle modifiche sulle tempistiche, sui costi e sulla qualità del progetto. Ciò può portare a decisioni poco informate e potenzialmente dannose per il successo del progetto.

Alla luce di questi temi, emerge la necessità di sviluppare strumenti e metodologie che consentano di automatizzare la tracciabilità: implementando o adottando sistemi che si occupino della registrazione delle modifiche, garantendo una documentazione accurata e aggiornata in tempo reale. Inoltre, è utile facilitare l'analisi delle differenze, fornendo mezzi per confrontare efficacemente diverse versioni di un progetto, o più progetti differenti, identificando rapidamente le differenze in termini di attività, risorse e budget.

Per affrontare queste esigenze, è stato sviluppato uno strumento che mira a fornire una soluzione per la gestione, l'analisi e le differenze fra progetti. Il sistema è progettato per:

1. modellare i progetti: utilizzando classi e oggetti che rappresentano i componenti fondamentali di un progetto - come attività, risorse e il progetto stesso -il sistema crea una rappresentazione coerente e organizzata delle informazioni progettuali.
2. automatizzare la tracciabilità delle modifiche: ogni modifica apportata a un progetto viene registrata attraverso metodi dedicati, garantendo che tutte le variazioni siano documentate in modo dettagliato e accessibile.
3. analizzare le differenze tra progetti: Il sistema implementa algoritmi di confronto, utilizzando metriche di similarità attraverso una classe dedicata. Questi algoritmi considerano diversi fattori, tra cui:
    - somiglianza dei nomi delle attività: valuta quanto i nomi delle attività siano simili, utilizzando tecniche di fuzzy matching.

- differenze nelle durate: analizza le variazioni nelle durate delle attività, identificando modifiche significative nelle tempistiche.
- confronto delle dipendenze: esamina le relazioni tra attività (predecessori e successori) per individuare cambiamenti nella struttura del progetto.
- analisi delle risorse: Confronta le risorse assegnate alle attività, sia in termini di quantità che di costi, per rilevare differenze che possono influire sul budget e sull'esecuzione del progetto.

4. generare report: Attraverso una classe apposita, il sistema è in grado di produrre report che evidenziano le differenze tra progetti, facilitando la comprensione delle modifiche e supportando la ricostruzione fra un progetto ed un altro.

L'adozione di strumenti come quello proposto rappresenta un'ulteriore opzione nella gestione moderna dei progetti. In un contesto in cui i progetti diventano sempre più complessi e le esigenze di tracciabilità e trasparenza aumentano, soluzioni innovative sono essenziali per mantenere la competitività e l'efficienza.La capacità di ricostruire in modo preciso le modifiche progettuali non solo migliora la gestione interna del progetto, ma ha anche implicazioni positive per gli stakeholder esterni.

3.1. Spiegazione del modello

3.2. Introduzione a Python

Per lo sviluppo della soluzione alle problematiche sopra esposte si è deciso di utilizzare il linguaggio di programmazione Python per modellare le entità coinvolte ed implementare gli algoritmi che consentono l'identificazione e la ricostruzione di differenze fra più progetti. La selezione del linguaggio di programmazione ha rappresentato una decisione molto importante nello sviluppo del mio progetto in quanto ha influito sulla produttività, sulla manutenzione del codice e sulla capacità di soddisfare i requisiti funzionali e non. Nel contesto del seguente lavoro di tesi, si è scelto di utilizzare Python come linguaggio di sviluppo, una decisione ravveduta da una serie di ragioni legate sia alle caratteristiche intrinseche del linguaggio sia alle specifiche esigenze del lavoro.

Python è un linguaggio di programmazione ad alto livello, interpretato, ovvero esegue direttamente il codice linea per linea quindi se vi sono errori nel codice del programma, ne interromperà l'esecuzione, in modo tale che i programmatori possono velocemente identificare gli errori nel codice. A differenza di altri linguaggi di programmazione, esso non utilizza le parantesi graffe ma bensì l'indentazione. Python è un linguaggio non fortemente tipizzato, infatti non è necessario dichiarare le tipologie di variabile nella scrittura del codice perché le determina nel tempo di esecuzione. Inoltre, è più vicino al linguaggio umano rispetto ad altri linguaggi di programmazione e questo implica che chi lo utilizza non deve preoccuparsi delle sue funzionalità sottostanti come l'architettura del calcolatore o la gestione della memoria. Uno dei tratti distintivi di Python è la sua estesa libreria standard, che mette a disposizione strumenti per gestire una vasta gamma di operazioni, dalla manipolazione di stringhe all'accesso al file system, dalla gestione delle reti alla programmazione concorrente. Inoltre, la comunità Python ha creato un ampio ecosistema di librerie e framework supplementari, accessibili attraverso il Python Package Index (PyPI), che ampliano ulteriormente le potenzialità del linguaggio. Il linguaggio supporta paradigmi di programmazione multipli, tra cui la programmazione procedurale, funzionale e orientata agli oggetti, offrendo una notevole flessibilità agli sviluppatori.

È stato creato da Guido van Rossum, un programmatore informatico olandese che lavorava al Centrum Wiskunde & Informatica (CWI), in principio era un progetto a cui van Rossum si dedicava per diletto durante il periodo natalizio. Il nome del linguaggio deriva allo show televisivo della BBC Monty Python's Flying Circus

perché l'inventore ne era grande appassionato. Egli ha rilasciato la prima versione del codice Python (versione 0.9.0) nel 1991 ed includeva già delle ottime caratteristiche, come ad esempio alcuni tipi di dati e funzioni per la gestione degli errori. Successivamente è stato rilasciato Python 1.0 nel 1994 con nuove funzioni per elaborare facilmente liste di dati e mappe. Python 2.0 è stato rilasciato nell'ottobre del 2000 con nuove caratteristiche utili ai programmatori, come il supporto ai caratteri Unicode. Nel dicembre 2008 è stato Python 3.0 che includeva nuove funzioni di stampa, altri supporti per la divisione numerica e la gestione degli errori. Inoltre, nel corso del tempo, fino ai giorni nostri sono state rilasciate numerose versioni minor del linguaggio.

La decisione di utilizzare Python per lo sviluppo di questo lavoro è stata guidata da diversi fattori chiave. In primo luogo, la semplicità e la leggibilità del codice offerta da Python sono state determinanti essendo il linguaggio rinomato per la sua sintassi pulita e minimalista, che rende il codice facile da leggere e comprendere. Questa caratteristica è particolarmente importante in un lavoro che ha richiesto la modellazione e la manipolazione di strutture dati complesse come attività, risorse e relazioni tra esse. La leggibilità del codice anche persone che non sono prettamente dei programmatori o degli ingegneri del software. Un altro elemento è il supporto al paradigma di programmazione orientata agli oggetti (OOP), basandosi il progetto su una struttura fortemente orientata agli oggetti, con classi che rappresentano entità come progetto, attività, risorse ecc. quindi offrendo un robusto supporto consente l'implementazione di classi, ereditarietà, polimorfismo e altri concetti di OOP in modo diretto e intuitivo. Ciò ha permesso quindi di modellare le componenti del lavoro in modo naturale e coerente, facilitando l'estensibilità e la modularità del codice. La rapidità nella prototipazione offerta dal linguaggio è stata opportuna durante la scrittura delle classi; infatti, grazie alla sua natura interpretata ha consentito consente uno sviluppo rapido e iterativo. Questa agilità ha permesso di implementare funzionalità, testare soluzioni alternative e adattare il codice alle nuove esigenze emerse durante il processo di sviluppo. Per di più, la vasta comunità e l'ecosistema di librerie di Python ha reso possibile l'utilizzo di librerie specifiche, come thefuzz per il calcolo della similarità tra stringhe, facilitando l'implementazione di funzionalità avanzate come le metriche di confronto tra attività, di conseguenza l'accesso a queste risorse hanno accelerato lo sviluppo e migliorato la qualità complessiva del software. Inoltre, la portabilità e compatibilità multipiattaforma di Python garantiscono che il software sviluppato possa essere eseguito su tutti i principali sistemi operativi, tra cui Windows, macOS e Linux. Questa portabilità aumenta la flessibilità nell'implementazione e nella distribuzione del progetto, permettendo di raggiungere un'ampia gamma di utenti senza la necessità di apportare modifiche al codice sorgente.

In sintesi, la scelta di Python è stata motivata dalla combinazione di questi fattori, che hanno contribuito a soddisfare le esigenze specifiche del lavoro. La semplicità e la leggibilità del codice, il supporto avanzato per la programmazione orientata agli oggetti, la rapidità di sviluppo, l'ampia comunità di supporto e la portabilità multipiattaforma, la facilità di integrazione con altri sistemi e la gestione efficiente delle eccezioni hanno reso Python il linguaggio ideale per realizzare una soluzione alle problematiche esposte.

3.3. Solid best practice

Durante lo sviluppo del codice ho adottato l'approccio SOLID con l'obbiettivo di assicurare qualità, manutenibilità e flessibilità; la scelta di seguire questi principi è stata motivata dalla necessità di realizzare un progetto modulare e facilmente adattabile a nuove esigenze permettendo che ogni componente fosse chiaramente definito e indipendente e favorendo così l'evoluzione del software senza comprometterne il funzionamento generale. Di seguito una breve presentazione, con qualche esempio, dei principi SOLID che sono stati applicati all'applicazione che verrà presentata successivamente.

I principi SOLID rappresentano un insieme di linee guida fondamentali per la progettazione di software orientato agli oggetti. Formulati da Robert C. Martin, questi principi mirano a migliorare la qualità del codice, rendendolo più flessibile, mantenibile e facilmente estendibile. L'acronimo SOLID si riferisce a cinque principi distinti: il Principio di Responsabilità Unica (Single Responsibility Principle, SRP), il Principio Aperto/Chiuso (Open/Closed Principle, OCP), il Principio di Sostituzione di Liskov (Liskov Substitution Principle, LSP), il Principio di Segregazione delle Interfacce (Interface Segregation Principle, ISP) e il Principio di Inversione delle Dipendenze (Dependency Inversion Principle, DIP). Di seguito, verranno esaminati in dettaglio ciascuno di questi principi.

3.3.1. Single Responsability Principle

Il Principio di Responsabilità Unica (SRP) afferma che ogni classe dovrebbe avere una sola ragione per cambiare, ovvero dovrebbe essere responsabile di una sola parte della funzionalità del sistema. Questo principio riduce la complessità del codice e rende le modifiche più semplici, poiché ogni classe è isolata in termini di compiti e non interferisce con le altre. Una progettazione che segue l'SRP evita la creazione di classi monolitiche che inglobano molteplici responsabilità, promuovendo invece una suddivisione logica e coerente. Ad esempio, in Python, se si ha una classe ReportGenerator che si occupa sia della generazione dei dati sia della formattazione del report, è possibile separare queste due responsabilità in due classi distinte: DataFetcher per il recupero dei dati e ReportFormatter per la formattazione, mantenendo così una singola responsabilità per ciascuna classe.

3.3.2. Open/Closed Principle

Il Principio Aperto/Chiuso (OCP) stabilisce che le entità software (classi, moduli, funzioni) devono essere aperte per estensione ma chiuse per modifica. Ciò significa che il comportamento di un modulo dovrebbe poter essere esteso senza dover modificare il suo codice sorgente, favorendo l'uso dell'ereditarietà e dei pattern di progettazione per introdurre nuove funzionalità. Ad esempio, in Python, si potrebbe avere una classe base Notification e due classi derivate EmailNotification e SMSNotification. Se si vuole aggiungere un nuovo tipo di notifica, come PushNotification, si può semplicemente creare una nuova classe derivata senza modificare il codice esistente.

3.3.3. Liskov Substitution Principle

Il Principio di Sostituzione di Liskov (LSP) è un principio derivante dai concetti di ereditarietà e polimorfismo. Esso sostiene che le classi derivate devono poter essere utilizzate al posto delle classi base senza alterare il corretto funzionamento del programma. In altre parole, i tipi derivati devono essere sostituibili ai tipi base, e qualsiasi istanza della classe base deve comportarsi allo stesso modo quando sostituita con un'istanza di una classe derivata. Ad esempio, se si ha una classe base Bird con un metodo fly(), una classe derivata Penguin che non può volare violerebbe il principio LSP, poiché un pinguino non dovrebbe essere sostituibile ad un uccello che può volare. In questo caso, sarebbe meglio ridefinire la gerarchia per evitare queste sostituzioni improprie.

3.3.4. Interface Segregation Principle

Il Principio di Segregazione delle Interfacce (ISP) è incentrato sulla creazione di interfacce specifiche per ogni client piuttosto che su interfacce generiche e troppo estese. Esso suggerisce che nessun client dovrebbe dipendere da metodi che non utilizza. In pratica, questo principio si traduce nella necessità di evitare interfacce "grasse", che contengono metodi non pertinenti a tutte le classi che le implementano, a favore di interfacce più mirate e coerenti con le reali esigenze di utilizzo. In Python, questo può essere realizzato utilizzando più classi base astratte invece di una singola classe che contiene tutti i metodi. Ad esempio, invece di avere una classe Device con metodi print() e scan(), si potrebbero creare due interfacce Printer e Scanner, che saranno implementate solo dalle classi che ne hanno effettivamente bisogno.

3.3.5 Dependency Inversion Principle

Infine, il Principio di Inversione delle Dipendenze (DIP) enfatizza l'importanza di dipendere da astrazioni e non da implementazioni concrete. Secondo questo principio, i moduli di alto livello non devono dipendere da quelli di basso livello, bensì entrambi devono dipendere da astrazioni. Le astrazioni non devono dipendere dai dettagli, ma i dettagli devono dipendere dalle astrazioni. Questo principio permette di ottenere una maggiore flessibilità, facilitando il riutilizzo del codice e rendendo il sistema più facile da testare. In Python, questo può essere realizzato utilizzando classi astratte e iniezione di dipendenze. Ad esempio, se si ha una classe OrderProcessor che dipende da un PaymentService, sarebbe meglio passare un'interfaccia PaymentServiceInterface al costruttore di OrderProcessor, piuttosto che una classe concreta. In questo modo, sarà possibile sostituire facilmente l'implementazione del servizio di pagamento senza modificare la logica di OrderProcessor.

I principi SOLID possono essere applicati anche al linguaggio di programmazione Python. Sebbene Python non sia un linguaggio strettamente orientato agli oggetti come altri linguaggi (ad esempio Java o C#), esso supporta pienamente i concetti di programmazione orientata agli oggetti e quindi si presta perfettamente all'implementazione dei principi SOLID. L'approccio dinamico e flessibile di Python permette di seguire l'SRP, facilitando la suddivisione delle responsabilità tra diverse classi e funzioni. Il principio OCP può essere rispettato utilizzando la composizione e l'ereditarietà per estendere le funzionalità senza modificare il codice esistente. Il principio LSP è applicabile creando classi che seguono rigorosamente il comportamento definito dalle classi base, mentre il principio ISP può essere seguito evitando di creare interfacce che contengono metodi non necessari per tutte le implementazioni. Infine, il principio DIP può essere realizzato in Python attraverso l'uso di astrazioni, come interfacce (simulate attraverso classi base astratte) e l'iniezione di dipendenze, che rendono il codice più modulare e testabile.

L'applicazione dei principi SOLID conduce alla creazione di un software che è più stabile, mantenibile e di qualità elevata. Essi rappresentano un insieme di best practice che supportano gli sviluppatori nella creazione di codice modulare, favorendo l'adattabilità del software ai cambiamenti e riducendo i costi di manutenzione a lungo termine. Pur non essendo sempre obbligatorio seguirli alla lettera, i principi SOLID forniscono un quadro di riferimento prezioso per lo sviluppo di soluzioni software che siano robusti e scalabili nel tempo.

3.4. Architettura

Oltre al linguaggio di programmazione e al principio SOLID per attuare al meglio i canoni della programmazione orientata agli oggetti, ho usufruito di GitHub per conservare il codice sorgente in maniera ordinata, mostrare e ricevere riscontri da coloro che mi hanno seguito durante il percorso di tesi.

GitHub è un servizio web e cloud-based che aiuta gli sviluppatori ad archiviare e gestire il loro codice, a tracciare e controllare le modifiche. Per capire esattamente cos'è GitHub, è necessario, seppur in maniera introduttiva, presentare due principi collegati: il controllo delle versioni e Git. Il controllo delle versioni aiuta gli sviluppatori a tracciare e gestire le modifiche al codice di un progetto software. Man mano che un progetto software cresce, il controllo delle versioni diventa essenziale. Esso permette agli sviluppatori di lavorare in sicurezza attraverso il branching (ramificazione) e il merging (fusione). Con il branching, uno sviluppatore duplica parte del codice sorgente (chiamato repository). Lo sviluppatore può quindi apportare in modo sicuro modifiche a quella parte del codice senza influenzare il resto del progetto. Una volta che lo sviluppatore fa funzionare correttamente la sua parte di codice, può fondere quel codice nel codice sorgente principale e renderlo ufficiale. Tutte queste modifiche vengono poi monitorate e, se necessario, possono essere ripristinate.

Mentre Git è uno specifico sistema di controllo versioni open-source creato da Linus Torvalds nel 2005. In particolare, Git è un sistema di controllo versioni distribuito, il che significa che l'intero codice base e la cronologia sono disponibili sul computer di ogni sviluppatore, il che permette di creare facilmente ramificazioni e fusioni.

GitHub è un'azienda a scopo di lucro che offre un servizio di hosting di repository Git basato su cloud. Praticamente, rende molto più facile per individui e gruppi di lavoro utilizzare Git per il controllo delle versioni e la collaborazione. L'interfaccia di GitHub è abbastanza facile da usare, quindi anche chi è alle prime armi possono sfruttare le funzionalità di Git. Infatti, senza GitHub, l'utilizzo di Git richiede generalmente una maggiore esperienza tecnica e l'utilizzo della linea di comando.

Fatta questa premessa, in appendice è presente il collegamento a cui è possibile accedere al codice sorgente del lavoro, notando, inoltre, come è stato utilizzato GitHub nello sviluppo di questo lavoro.

In modo da avere un quadro completo dell'intero progetto il seguente diagramma delle classi ne fornisce uno spaccato dell'architettura (Figura 3):

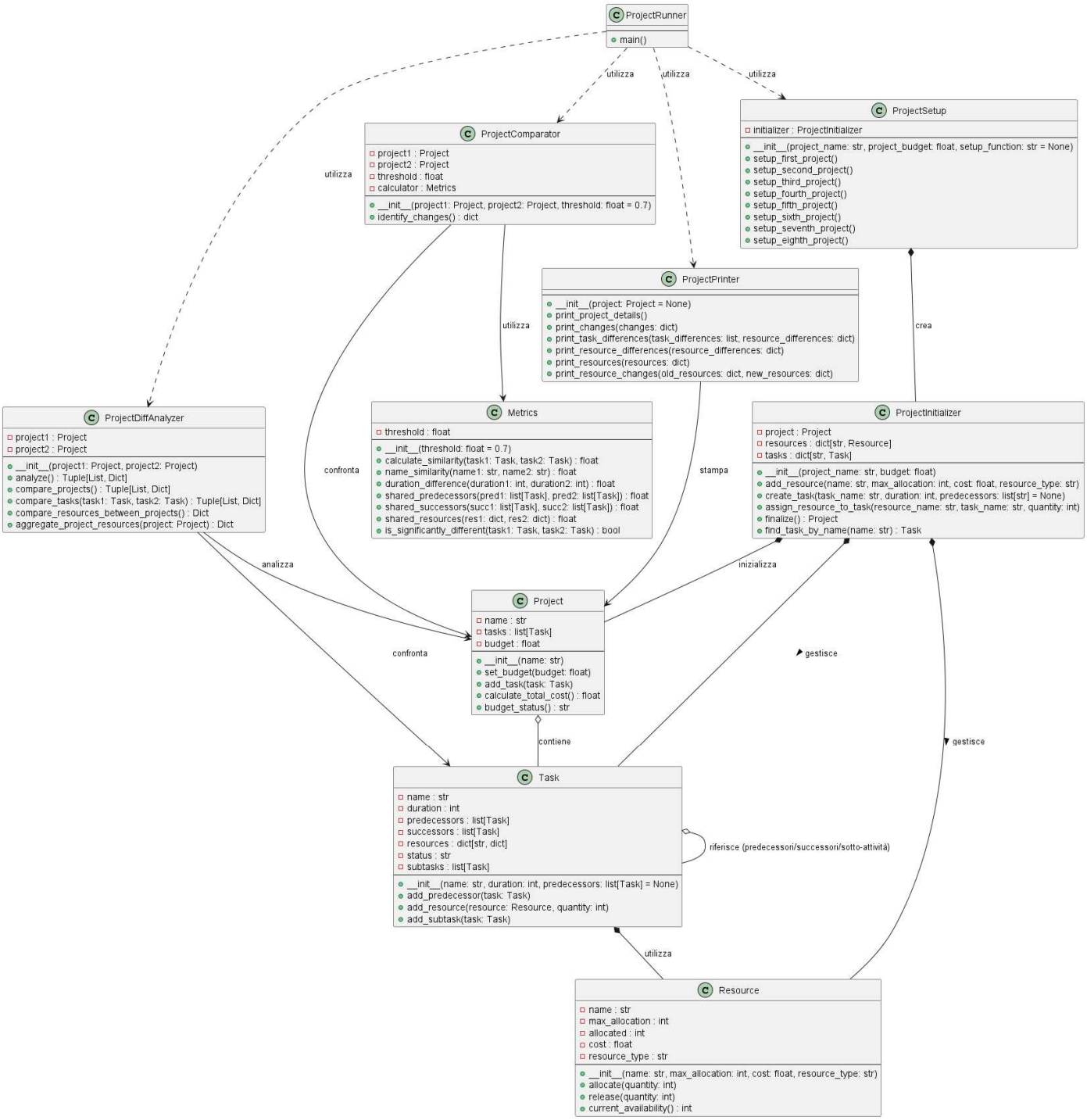
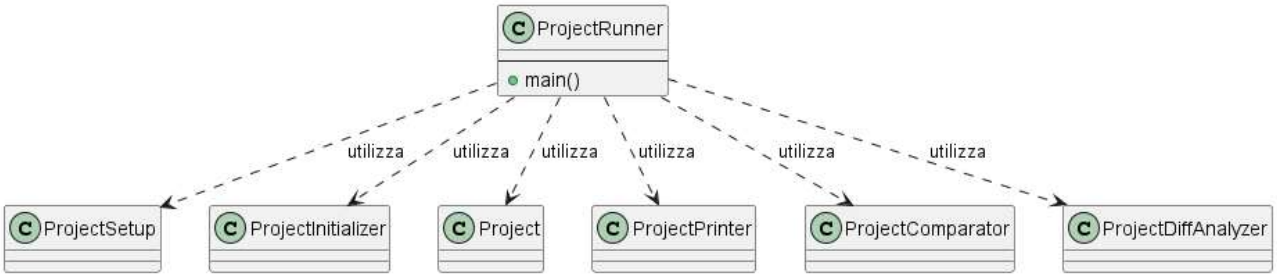


Figura 3: Architettura del progetto

Di seguito verrà presentata l'architettura delle classi che compongono il mio lavoro di tesi, spiegando i loro metodi e attributi, le responsabilità di ogni singola classe e le relazioni che intercorrono tra esse. Partendo dall'alto iniziamo con la classe di ProjectRunner.py

3.4.1. ProjectRunner.py



La classe è il punto di ingresso principale per l'esecuzione dell'applicazione in quanto coordina l'interazione tra le varie componenti del sistema, orchestrando la creazione, la stampa, il confronto e l'analisi dei progetti, ProjectRunner utilizza le altre classi del progetto per realizzare le funzionalità principali dell'applicazione. In particolare, il metodo main() rappresenta il punto di ingresso dell'applicazione e coordina diverse operazioni chiave. Fondamentalmente, si occupa della creazione dei progetti utilizzando la classe ProjectSetup per creare diverse istanze di progetti e per ognuno specifica il nome, il budget e il metodo di setup da utilizzare. Successivamente, il metodo procede alla finalizzazione dei progetti, infatti, dopo aver configurato un progetto con ProjectSetup, chiama il metodo finalize() sull'istanza di ProjectInitializer per ottenere l'oggetto Project completo. L'ultimo passaggio appena descritto è indispensabile per trasformare la configurazione iniziale in un progetto operativo.

Una volta ottenuti i progetti finalizzati, il metodo della classe esegue la stampa dei dettagli dei progetti.

Il metodo main() si occupa anche del confronto tra progetti. Utilizza ProjectComparator per confrontare coppie di progetti e identificare le modifiche tra di essi. Chiama il metodo identify_changes() per ottenere un dizionario delle modifiche, che viene poi passato a ProjectPrinter per la stampa. Questo processo consente di evidenziare le differenze significative tra le diverse versioni o tipologie di progetti.

Infine, impiega ProjectDiffAnalyzer per l'analisi delle differenze tra due progetti specifici, come il primo e il secondo progetto. Chiama il metodo analyze() per ottenere le differenze nelle attività e nelle risorse, che vengono poi stampate utilizzando ProjectPrinter. Questa analisi approfondita permette di comprendere in dettaglio le variazioni tra i progetti, supportando decisioni informate sulla gestione delle modifiche.

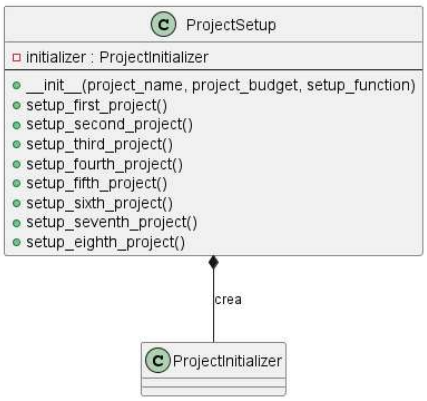
Di seguito un riassunto, tramite elenco puntato, per chiarire le principali operazioni della classe presa in esame:

1. Creazione e Stampa dei Progetti:
    - Crea progetti utilizzando ProjectSetup, specificando il nome, il budget e il metodo di setup per ciascuno.
    - Finalizza ciascun progetto chiamando finalize().
    - Istanzia ProjectPrinter per ciascun progetto e stampa i dettagli.

2. Confronto tra Coppie di Progetti:
    - Confronta varie coppie di progetti
    - Per ciascun confronto:
        - Istanzia ProjectComparator con i due progetti da confrontare.
        - Chiama identify_changes() per ottenere le modifiche.
        - Istanzia ProjectPrinter e chiama print_changes() per stampare le modifiche fra un progetto ed un altro.
3. Analisi delle Differenze e ricostruzione tra Progetti:
    - Analizza le differenze tra il primo e il secondo progetto.
    - Istanzia ProjectDiffAnalyzer con i due progetti.
    - Chiama analyze() per ottenere le differenze nelle attività e nelle risorse.
    - Utilizza ProjectPrinter per stampare le differenze dettagliate delle attività e delle risorse.

La progettazione di ProjectRunner come punto di ingresso centrale semplifica l'esecuzione dell'applicazione e facilita la manutenzione del codice, permettendo di aggiungere o modificare le operazioni principali senza influire sulle singole classi che gestiscono le specifiche funzionalità. Per eventuali lavori futuri, questa classe potrebbe essere il controller API dell'applicazione.

3.4.2. ProjectSetup.py



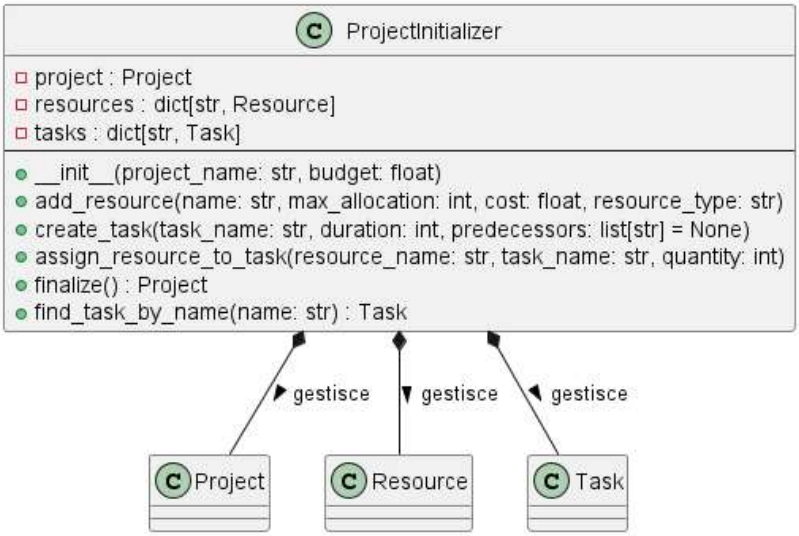
La classe ha la funzione di configurare ed inizializzare i progetti. Essa rappresenta un'interfaccia attraverso la quale è possibile definire progetti con specifiche attività e risorse, personalizzandoli in base alle esigenze particolari di ogni scenario. ProjectSetup è progettata per offrire flessibilità e modularità nella creazione dei progetti; attraverso il suo costruttore, la classe accetta parametri come il nome del progetto, il budget assegnato e un'opzione per specificare una funzione di setup personalizzata. Questo approccio consente di istanziare progetti con caratteristiche differenti senza la necessità di modificare la struttura interna della classe stessa.

Al momento della creazione di un'istanza di ProjectSetup, viene inizializzato un oggetto di ProjectInitializer, quest'ultimo è responsabile della costruzione effettiva del progetto, gestendo la creazione delle risorse, delle attività e delle relazioni tra esse. Di conseguenza, ProjectSetup agisce come un intermediario, orchestrando il processo di inizializzazione attraverso ProjectInitializer. La classe ProjectSetup è essenziale per la modularità

e l'estensibilità del sistema, agendo come un livello superiore di configurazione, essa separa la logica di definizione del progetto dalla logica di implementazione effettiva gestita da ProjectInitializer. Le scelte effettuate nei metodi di setup influenzano direttamente il comportamento e le caratteristiche delle istanze delle classi fondamentali del software. Questo design promuove una chiara separazione delle responsabilità, facilitando la manutenzione e l'espansione del sistema.

I vantaggi dell'approccio utilizzato sono molteplici, in prima istanza, la flessibilità rappresenta un elemento chiave, difatti, la possibilità di definire metodi di setup personalizzati permette di adattare rapidamente il sistema a nuove esigenze, senza dover alterare la struttura esistente. Ciò consente di rispondere agilmente a requisiti emergenti o cambiamenti nel contesto operativo, mantenendo al contempo la stabilità e la coerenza del sistema. In secondo luogo, la riutilizzabilità del codice è notevolmente migliorata. Separando la configurazione dalla logica di implementazione, il codice diventa più modulare e facilmente riutilizzabile. Questa modularità riduce la duplicazione, poiché componenti già sviluppate possono essere impiegate in diversi contesti senza necessità di riscrittura. Inoltre, facilita l'aggiornamento delle funzionalità, poiché le modifiche possono essere effettuate in un'unica posizione nel codice, propagandosi automaticamente a tutte le parti del sistema che ne fanno uso. Infine, l'approccio adottato semplifica significativamente la manutenzione del sistema. La chiara divisione delle responsabilità tra le varie componenti rende più agevole la comprensione dell'architettura complessiva e delle interazioni tra i diversi moduli. Di conseguenza, le modifiche possono essere apportate in modo isolato, senza impatti indesiderati su altre componenti, ciò riduce il rischio di introdurre errori durante gli interventi di manutenzione e migliora l'efficienza del processo di sviluppo, consentendo di mantenere un alto livello di qualità e affidabilità del software.

3.4.3. ProjectInitializer.py



È responsabile dell'inizializzazione e della configurazione dettagliata, infatti agisce come un costruttore che si occupa di creare l'istanza del progetto (Project) e di popolarlo con le risorse (Resource) e le attività (Task) necessarie. La classe fornisce metodi per aggiungere risorse, creare attività, assegnare risorse alle attività e finalizzare il progetto per l'utilizzo successivo. Il costruttore della classe ProjectInitializer accetta il nome del progetto e il budget come parametri. All'interno del costruttore, viene creata un'istanza della classe Project con il nome fornito, e il budget viene impostato tramite il metodo set_budget. Inoltre, vengono inizializzati due dizionari: self.resources, per memorizzare le risorse disponibili, e self.tasks, per tenere traccia delle attività create.

Oltre alle funzioni precedentemente menzionati, la classe ha a disposizione find_task_by_name, questo metodo di supporto permette di recuperare un'attività dal dizionario self.tasks utilizzando il suo nome ed è

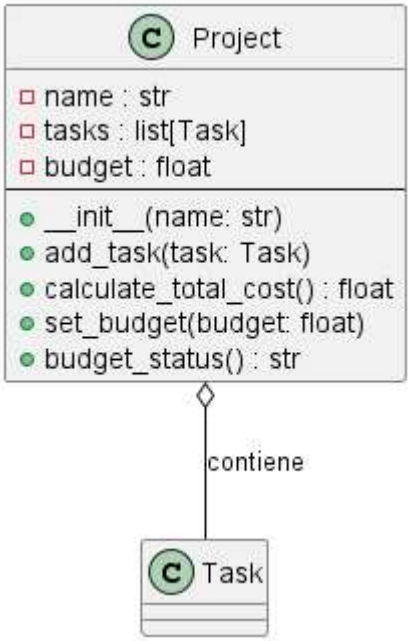
particolarmente utile quando si devono stabilire dipendenze tra le attività, assicurando che le attività predecessori esistano e siano correttamente referenziate.

ProjectInitializer svolge il ruolo di costruttore dei progetti, gestendo in modo organico e strutturato la creazione delle componenti fondamentali del progetto stesso, come attività e risorse. Grazie ai suoi metodi, permette di:

- Definire le risorse disponibili, con le loro caratteristiche.
- Creare le attività che compongono il progetto, specificando le durate e le dipendenze.
- Assegnare le risorse alle attività in modo controllato, gestendo l'allocazione delle quantità.
- Finalizzare il progetto, rendendolo pronto per ulteriori elaborazioni o analisi.

La sua progettazione orientata agli oggetti e la separazione delle responsabilità facilitano la manutenzione del codice e l'estensibilità del sistema, infatti, si concentra sulle operazioni di basso livello per la creazione del progetto, lasciando ad altre classi il compito di definire le configurazioni (ProjectSetup) o di analizzare e stampare i dati (ProjectPrinter, ProjectDiffAnalyzer). Adottando l'approccio descritto si hanno vantaggi di modularità, riutilizzabilità e manutenibilità rendendo inoltre il codice più facile da comprendere.

3.4.4. Project.py



La classe Project esprime il fulcro centrale del sistema servendo da contenitore per tutte le informazioni relative a un progetto specifico ed incapsulando non solo i dettagli generali del progetto, come il nome e il budget assegnato, ma anche l'insieme delle attività (Task) che lo compongono e le risorse coinvolte attraverso queste attività. La classe Project è fondamentale per la gestione coerente e organizzata dei dati del progetto, fornendo metodi che consentono di interagire efficacemente con le sue componenti.

Di seguito, una descrizione delle funzioni principali:

1. Costruttore (__init__(self, name))

Il costruttore della classe Project inizializza un'istanza del progetto con il nome specificato. All'atto della creazione, il budget viene inizialmente impostato a zero e viene inizializzata una lista vuota per contenere le

attività del progetto. Questo approccio permette di creare un progetto in modo incrementale, aggiungendo successivamente il budget e le attività necessarie.

2. set_budget(self, budget)

Questo metodo consente di impostare il budget totale assegnato al progetto. Il budget rappresenta la somma massima di risorse finanziarie disponibili per l'esecuzione del progetto e viene utilizzato come parametro di riferimento per confrontare i costi stimati derivanti dalle attività pianificate.

3. add_task(self, task)

Permette di aggiungere un'istanza di Task alla lista delle attività del progetto.

4. calculate_total_cost(self) -> float

Calcola il costo totale stimato del progetto, sommando i costi associati alle risorse di ciascuna attività. Il calcolo avviene iterando su tutte le attività del progetto e, per ciascuna di esse, moltiplicando la quantità di ogni risorsa utilizzata per il costo unitario della risorsa stessa. Il risultato è una stima del costo complessivo necessario per completare tutte le attività pianificate, tenendo conto delle risorse coinvolte.

5. budget_status(self) -> str

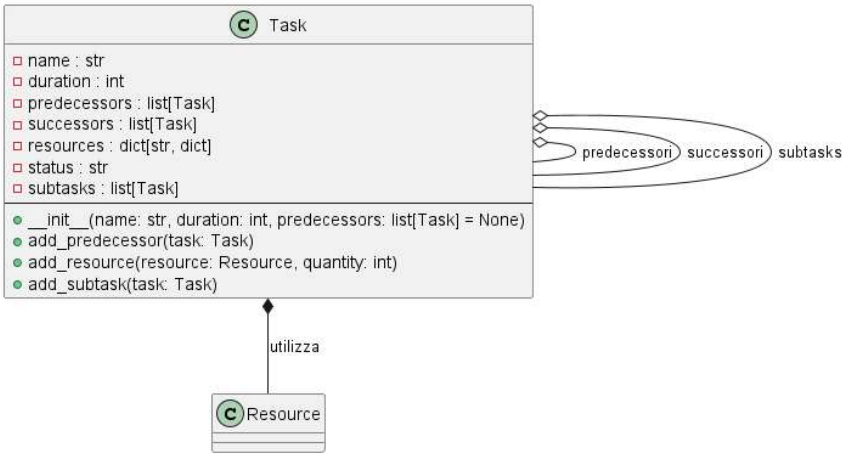
La funzione budget_status fornisce un'analisi dello stato finanziario del progetto rispetto al budget assegnato. Utilizza il metodo calculate_total_cost per ottenere il costo totale stimato e calcola il budget residuo sottraendo il costo totale dal budget assegnato. Restituisce una stringa che riassume il budget assegnato, il costo totale stimato e il budget residuo, offrendo una visione immediata della situazione finanziaria del progetto, informazione essenziale per i manager perché permette di identificare tempestivamente eventuali sforamenti di budget e di adottare le misure correttive necessarie.

Nel sistema complessivo la classe rappresenta uno dei fondamenti per l'intero sistema sviluppato riproducendo le informazioni che definiscono un progetto, come il nome, il budget e le attività da svolgere con le rispettive risorse assegnate. Fornisce metodi che permettono di:

- Gestire le Attività
- Monitorare i Costi
- Valutare lo Stato Finanziario

Essi consentono di mantenere il controllo sulle attività pianificate, di monitorare l'utilizzo delle risorse e di garantire che il progetto rimanga entro i limiti finanziari stabiliti. La possibilità di calcolare il costo totale e di valutare lo stato del budget in modo rapido e accurato è fondamentale per evitare sorprese finanziarie e per assicurare il successo del progetto. La progettazione orientata agli oggetti consente di soddisfare le esigenze in vari settori industriali, contribuendo al raggiungimento degli obiettivi organizzativi.

3.4.5. Task.py



Evidenzia le attività individuali che compongono un progetto. Ogni istanza di Task modella un'unità di lavoro specifica con attributi e metodi che permettono di definire le sue caratteristiche, gestire le dipendenze con altre attività e assegnare le risorse necessarie per il suo completamento.

La classe Task offre una serie di metodi che permettono di gestire in modo completo e flessibile le attività all'interno di un progetto. La seguente è una descrizione delle funzioni principali:

1. Costruttore (__init__(self, name, duration, predecessors=None))

Instanzia un'attività con il nome specificato, la durata e un elenco opzionale di predecessori. Se i predecessori non vengono specificati, l'attività viene considerata indipendente all'inizio. Gli attributi principali inizializzati nel costruttore includono:

- name (str): Il nome dell'attività, che funge da identificatore.
- duration (int): La durata prevista per il completamento dell'attività, espressa in giorni (o un'altra unità di tempo appropriata).
- predecessors (list[Task]): Una lista di attività che devono essere completate prima dell'inizio di questa attività.
- successors (list[Task]): Inizialmente vuota, verrà popolata con le attività che dipendono dal completamento di questa attività.
- resources (dict): Un dizionario che mappa il nome della risorsa ai dettagli dell'allocazione per questa attività.
- status (str): Lo stato corrente dell'attività, ad esempio "Non Iniziata", "In Corso", "Completata".
- subtasks (list[Task]): Una lista opzionale di sotto-attività che compongono l'attività corrente.

2. add_predecessor(self, task)

Permettere di aggiungere un'attività precedente a quella presa in considerazione alla lista dei predecessori ed aggiorna l'attività predecessore per includere l'attività corrente nella sua lista di successori. Questo meccanismo bidirezionale assicura che le dipendenze tra le attività siano gestite in modo coerente, facilitando la pianificazione e la sequenza delle operazioni.

3. add_resource(self, resource, quantity)

Consente di assegnare una risorsa specifica all'attività, specificando la quantità necessaria. Se la risorsa è già stata assegnata all'attività, il metodo aggiorna la quantità allocata, gestendo la differenza nell'allocazione della risorsa. Durante questo processo, il metodo invoca il metodo allocate dell'istanza Resource per assicurarsi che la quantità richiesta sia disponibile e che l'allocazione rispetti i limiti massimi.

4. add_subtask(self, task)

La funzione concede di aggiungere una sotto-attività all'attività corrente. Le sotto-attività sono istanze di Task che rappresentano attività più piccole o specifiche all'interno dell'attività principale. L'uso di sotto-attività permette di organizzare il lavoro in modo più dettagliato e strutturato, facilitando la gestione di attività complesse che possono essere suddivise in componenti più piccole.

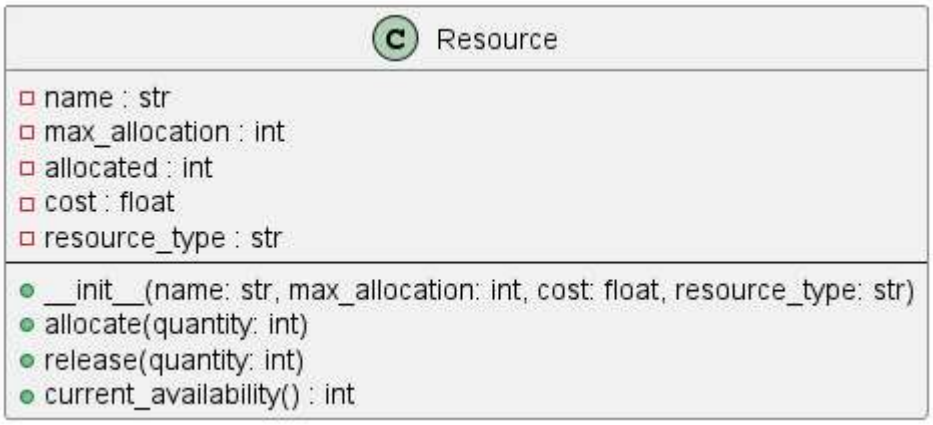
In aggiunta ai metodi descritti, la classe Task presenta molti attributi che definiscono lo stato e le caratteristiche dell'attività:

- name: Il nome dell'attività.
- duration: La durata prevista dell'attività.
- predecessors: La lista delle attività che devono essere completate prima di questa attività.
- successors: La lista delle attività che dipendono dal completamento di questa attività.
- resources: Un dizionario contenente le risorse assegnate all'attività e i dettagli dell'allocazione.
- status: Lo stato corrente dell'attività.
- subtasks: Una lista di sotto-attività associate all'attività corrente.

Riassumendo la classe Task è essenziale per la modellazione dettagliata delle attività di un progetto e permette di:

- Definire attività con caratteristiche specifiche, come nome, durata e risorse necessarie.
- Stabilire dipendenze tra le attività, gestendo predecessori e successori per una pianificazione efficace.
- Assegnare risorse alle attività, garantendo che le risorse siano allocate correttamente e che i limiti di disponibilità siano rispettati.
- Organizzare il lavoro attraverso sotto-attività, facilitando la gestione di attività complesse.

3.4.6. Resource.py



La classe Resource opera sulle risorse che possono essere allocate alle attività (Task) all'interno di un progetto (Project). Le risorse possono essere di vari tipi, come personale umano, materiali, attrezzature o altre risorse necessarie per il completamento delle attività pianificate e fornisce attributi e metodi che consentono di gestire efficacemente la disponibilità, l'allocazione e il rilascio delle risorse, garantendo che il progetto possa essere eseguito in modo efficiente e senza superare i limiti di disponibilità.

Offre una serie di funzioni che permettono di controllare lo stato della risorsa e gestire le operazioni di allocazione e rilascio:

1. Costruttore (__init__(self, name, max_allocation, cost, resource_type))

Istanzia una risorsa con i seguenti attributi:

- name (str): Il nome della risorsa, identificatore univoco.
- max_allocation (int): La quantità massima disponibile della risorsa per l'allocazione nel progetto.
- cost (float): Il costo per unità della risorsa. Questo valore è utilizzato per calcolare i costi associati alle attività che utilizzano la risorsa.
- resource_type (str): Il tipo di risorsa, ad esempio "Human", "Material", "Equipment". Questo attributo può essere utilizzato per categorizzare le risorse e applicare specifiche logiche di gestione.
- allocated (int): Inizialmente impostato a zero, rappresenta la quantità attualmente allocata della risorsa.

Il costruttore assicura che tutti gli attributi siano inizializzati correttamente, preparando la risorsa per le operazioni di allocazione successive.

2. allocate(self, quantity)

Questo metodo gestisce l'allocazione di una certa quantità della risorsa. Quando un'attività richiede una risorsa, il metodo allocate viene invocato per assegnare la quantità necessaria. Il metodo verifica se la quantità richiesta è disponibile, confrontando la somma dell'allocazione corrente e della quantità richiesta con la disponibilità massima (max_allocation). Se la quantità richiesta supera la disponibilità residua, viene sollevata un'eccezione e viene gestita l'impossibilità di allocare la risorsa. Se l'allocazione è possibile, il metodo aggiorna l'attributo allocated incrementandolo della quantità assegnata.

3. release(self, quantity)

Il metodo release gestisce il rilascio di una certa quantità precedentemente allocata della risorsa. Quando un'attività completa l'utilizzo di una risorsa o riduce la quantità necessaria, il metodo release viene utilizzato per aggiornare l'allocazione corrente. Il metodo verifica che la quantità da rilasciare non ecceda la quantità attualmente allocata. Successivamente, decrementa l'attributo allocated della quantità rilasciata, rendendo la risorsa disponibile per altre attività.

4. current_availability(self) -> int

Questo metodo restituisce la quantità di risorsa attualmente disponibile per l'allocazione. Calcola la disponibilità residua sottraendo la quantità allocata (allocated) dalla disponibilità massima (max_allocation). Questo valore è utile per verificare se ulteriori allocazioni possono essere soddisfatte e per pianificare l'utilizzo delle risorse nel progetto.

Oltre ai metodi descritti, la classe presenta diversi attributi che definiscono le caratteristiche della risorsa:

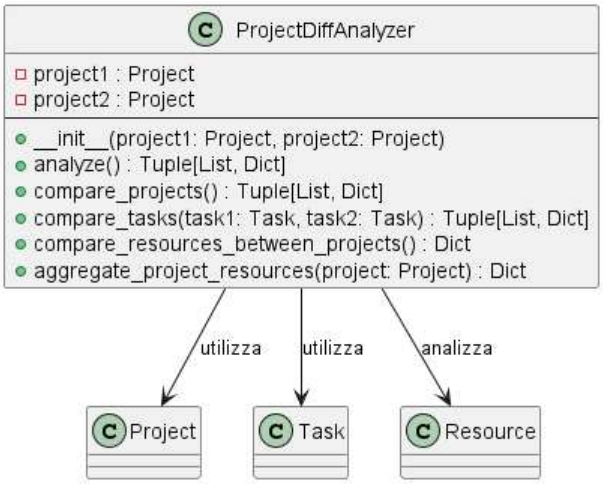
- name: Il nome della risorsa.
- max_allocation: La quantità massima disponibile della risorsa per il progetto.
- allocated: La quantità attualmente allocata della risorsa.
- cost: Il costo per unità della risorsa.
- resource_type: Il tipo di risorsa.

La classe Resource interagisce principalmente con la classe Task e, indirettamente, con la classe Project fornendo meccanismi per:

- Controllare la disponibilità delle risorse: attraverso gli attributi max_allocation e allocated, si tiene traccia della quantità totale disponibile e di quella già assegnata, permettendo di evitare sovra-allocazioni.
- Gestire l'allocazione e il rilascio: i metodi allocate e release consentono di allocare e rilasciare le risorse in modo controllato, aggiornando lo stato della risorsa e garantendo che le allocazioni rispettino i limiti imposti.
- Calcolare i costi associati: L'attributo cost permette di calcolare i costi associati all'utilizzo della risorsa nelle attività, contribuendo al calcolo del costo totale del progetto.
- Categorizzare le risorse: l'attributo resource_type consente di classificare le risorse, facilitando la gestione e l'applicazione di logiche specifiche per le diverse tipologie di risorse

In conclusione, offre la flessibilità necessaria per applicare logiche specifiche a diversi gruppi di risorse, adattandosi alle esigenze particolari di vari settori e progetti.

3.4.7. ProjectDiffAnalyzer.py



La classe ProjectDiffAnalyzer è progettata per confrontare due istanze di progetti (Project) e identificare le differenze tra di esse, analizzando sia le attività (Task) che le risorse (Resource) coinvolte nei progetti e fornendo un'analisi dettagliata delle variazioni che possono esistere tra due versioni di un progetto o tra progetti differenti.

La classe offre una serie di metodi che permettono di eseguire un confronto approfondito tra due progetti, le funzioni sviluppate nella classe sono:

1. Costruttore (__init__(self, project1, project2))

Accetta due istanze di Project come parametri, rappresentanti i due progetti da confrontare. L'inizializzazione prepara l'analizzatore per eseguire l'analisi delle differenze tra i due progetti specificati.

2. analyze(self) -> Tuple[List, Dict]

Questo metodo è il punto di ingresso principale per l'analisi delle differenze tra i due progetti restituendo una tupla contenente una lista e un dizionario:

- La lista contiene le differenze a livello di attività, come attività aggiunte, rimosse o modificate.
- Il dizionario contiene le differenze nelle risorse, come variazioni nelle quantità o nei costi delle risorse assegnate alle attività.

Il metodo analyze coordina l'utilizzo di altri metodi interni per eseguire un confronto dettagliato sia delle attività che delle risorse.

3. compare_projects(self) -> Tuple[List, Dict]

Esegue il confronto tra le due istanze di Project, analizzando le attività contenute in ciascun progetto ed identificando attività presenti in un progetto ma non nell'altro, nonché attività comuni con differenze significative. Il risultato è una lista di differenze rilevate tra i progetti.

4. compare_tasks(self, task1, task2) -> Tuple[List, Dict]

Questo metodo confronta due istanze di Task, per individuare scostamenti specifici fra le due attività analizzando attributi come il nome, la durata, i predecessori, i successori e le risorse assegnate. Restituisce una tupla contenente:

- Una lista di differenze rilevate negli attributi dell'attività.

- Un dizionario con dettagli sulle risorse che differiscono tra le due attività.

5. compare_resources_between_projects(self) -> Dict

Aggrega le risorse utilizzate in ciascun progetto confrontando e identificando le quantità totali e i costi delle risorse tra i due progetti, nonché differenze nelle quantità o nei costi delle risorse comuni. Restituisce un dizionario contenente le differenze rilevate nelle risorse.

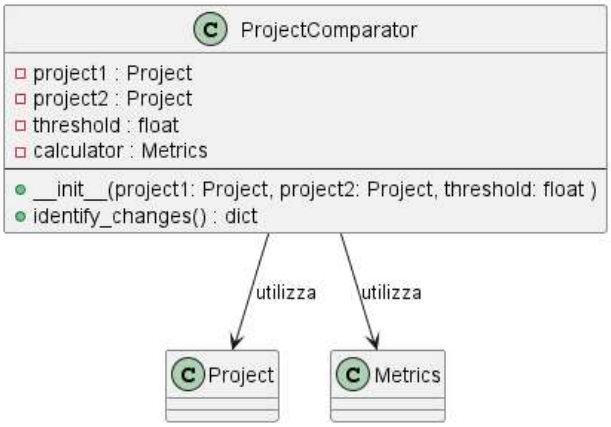
6. aggregate_project_resources(self, project) -> Dict

È un metodo di supporto aggrega le risorse di un progetto specifico. Itera attraverso tutte le attività del progetto e accumula le quantità e i costi delle risorse assegnate. Restituisce un dizionario che mappa i nomi delle risorse ai loro totali aggregati di quantità e costi. Questo risultato è utilizzato successivamente per il confronto tra i progetti.

Riassumendo, le principali attività svolte dalla classe presa in esame sono:

- l'identificazione delle differenze tra progetti
- supporto al processo decisionale
- facilitazione nella gestione delle modifiche

3.4.8. ProjectComparator.py



Questa classe utilizza metriche di similarità che vengono implementate nella classe Metrics per determinare se le attività (Task) in due progetti sono simili o se presentano differenze sostanziali. L'obiettivo principale di ProjectComparator è di supportare l'analisi delle modifiche progettuali, facilitando la comprensione di come un progetto è evoluto e la ricostruzione che permette il passaggio tra un progetto e l'altro.

1. Costruttore (__init__(self, project1, project2, threshold))

Il costruttore della classe ProjectComparator accetta come parametri due istanze di Project, le quali rappresentano i progetti da confrontare. Inoltre, accetta un parametro threshold, che definisce la soglia di similarità oltre la quale due attività sono considerate simili ed inoltre, inizializza un'istanza della classe Metrics, utilizzata per calcolare la similarità tra le attività.

Gli attributi utilizzati dal costruttore sono i seguenti:

- project1: primo progetto
- project2: secondo progetto da confrontare.
- threshold: soglia di similarità
- calculator: istanza della classe Metrics utilizzata per calcolare le metriche di similarità.

2. identify_changes(self) -> Dict

Questo metodo identifica i cambiamenti tra i due progetti, restituendo un dizionario che contiene:

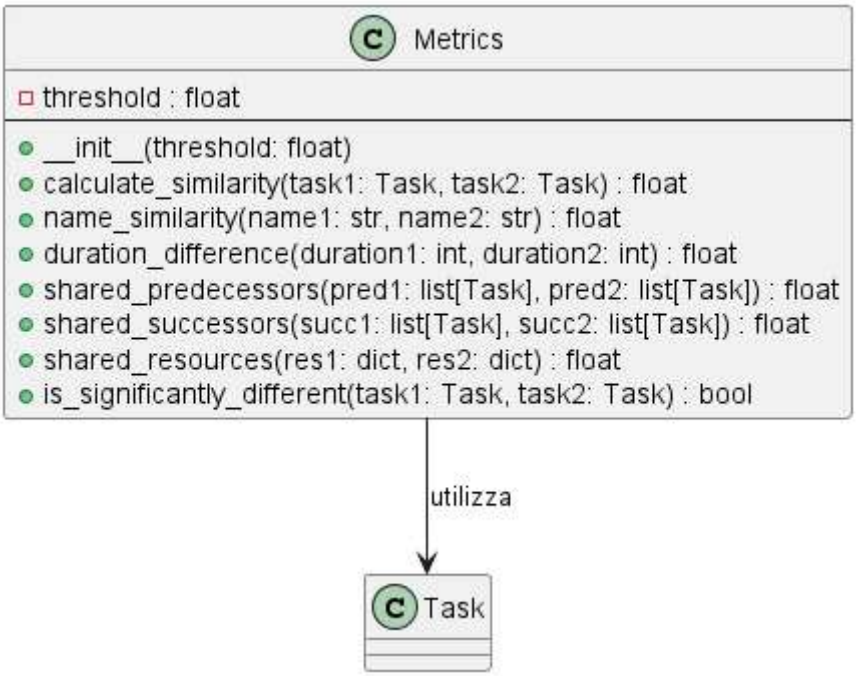
- added_tasks: attività presenti in project2 ma non in project1.
- removed_tasks: attività presenti in project1 ma non in project2.
- modified_tasks: attività che sono presenti in entrambi i progetti ma che presentano differenze significative.
- unchanged_tasks: attività che sono simili in entrambi i progetti, secondo la soglia di similarità definita.

Il metodo funziona nel seguente modo:

- Crea insiemi dei nomi delle attività per entrambi i progetti.
- Identifica le attività con lo stesso nome e le confronta utilizzando le metriche di similarità.
- Determina se le attività sono significativamente diverse basandosi sul punteggio di similarità e sulla soglia.
- Classifica le attività aggiunte, rimosse, modificate o invariate.

La classe ProjectComparator serve per il confronto automatizzato e l'identificazione dei cambiamenti tra progetti, attraverso l'utilizzo di metriche di similarità, consente di determinare rapidamente e accuratamente quali attività sono rimaste invariate e quali hanno subito modifiche significative. L'utilizzo di una soglia di similarità configurabile rende la classe flessibile e adattabile a diversi contesti applicativi, in cui il grado di sensibilità ai cambiamenti può variare.

3.4.9. Metrics.py



Progettata per calcolare la similarità tra due istanze di attività, la classe implementa diverse metriche che considerano vari aspetti delle attività, come il nome, la durata, i predecessori, i successori e le risorse assegnate. L'obiettivo principale di Metrics è fornire un indicatore che possa essere utilizzato per determinare se due attività sono significativamente diverse o abbastanza simili da essere considerate equivalenti nel contesto dell'analisi dei progetti.

I seguenti sono i metodi implementati nella classe Metrics:

1. Costruttore (__init__(self, threshold=0.7))

Il costruttore accetta un parametro che definisce la soglia di similarità oltre la quale due attività sono considerate simili. Questo valore predefinito nel lavoro di tesi è impostato a 0.7, ma può essere modificato in base alle esigenze specifiche del contesto applicativo.

2. calculate_similarity(self, task1, task2) -> float

Calcola un punteggio di similarità tra due attività, combinando diverse metriche individuali. Il punteggio risultante è un valore compreso tra 0 e 1, dove 1 indica una perfetta uguaglianza.

Il calcolo del punteggio di similarità avviene attraverso:

- Nome dell'attività: viene calcolata la similarità tra i nomi delle attività utilizzando una metrica di similarità delle stringhe, in particolare si utilizza il rapporto di similarità di Levenshtein.
- Durata: viene confrontata la durata delle due attività calcolando la differenza relativa.
- Predecessori e successori: viene valutata la sovrapposizione tra i predecessori e i successori delle due attività.
- Risorse assegnate: viene analizzata la similarità tra le risorse assegnate alle due attività, considerando sia i tipi di risorse che le quantità.

Il punteggio finale è ottenuto combinando queste metriche, possibilmente ponderandole in base all'importanza relativa di ciascun attributo.

3. name_similarity(self, name1, name2) -> float

Calcola la similarità tra i nomi di due attività. Utilizza l'algoritmo di fuzzy matching per determinare quanto i nomi siano simili. Il risultato è un valore compreso tra 0 e 1.

4. duration_difference(self, duration1, duration2) -> float

Calcola la differenza relativa tra le durate delle due attività. La formula può essere espressa come:

$$Differenza = 1 - \frac{|duration1 - duration2|}{max(duration1, duration2)}$$

Questo valore è compreso tra 0 e 1, dove 1 indica durate identiche.

5. shared_predecessors(self, predecessors1, predecessors2) -> float

Valuta la percentuale di predecessori condivisi tra le due attività. Calcola l'intersezione e l'unione degli insiemi di predecessori e determina il rapporto tra gli elementi comuni e il totale.

6. shared_successors(self, successors1, successors2) -> float

Simile al metodo precedente, ma applicato ai successori delle attività.

7. shared_resources(self, resources1, resources2) -> float

Analizza le risorse assegnate alle due attività, valutando la similarità in termini di tipi di risorse e quantità. Può considerare sia la presenza di risorse comuni che le differenze nelle quantità assegnate.

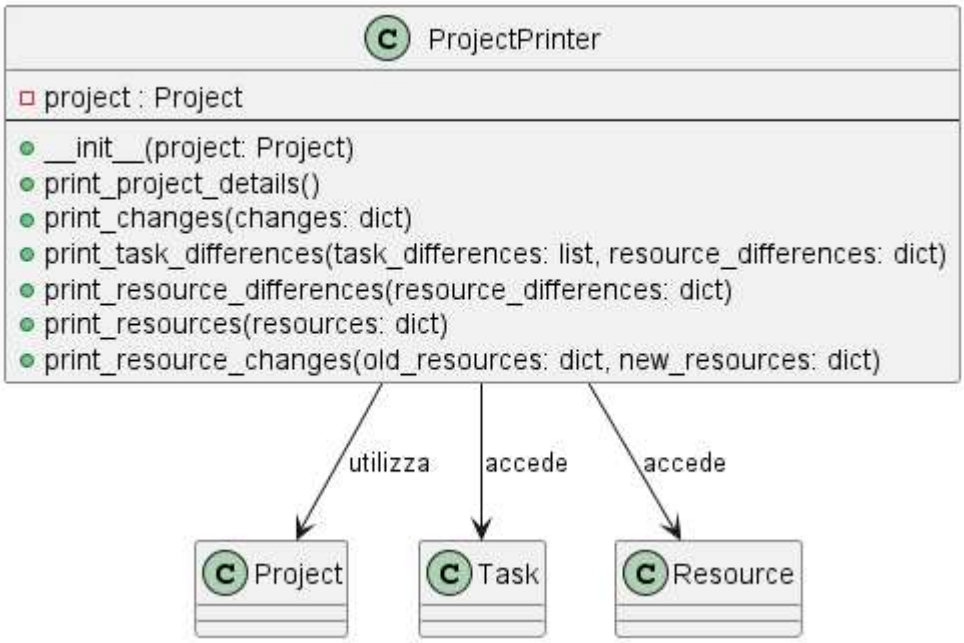
8. is_significantly_different(self, task1, task2) -> bool

Determina se due attività sono significativamente diverse in base al punteggio di similarità calcolato. Se il punteggio è inferiore alla soglia, il metodo restituisce True, indicando che le attività sono considerate diverse.

Chiaramente, l'attributo principale in questa classe è la soglia threshold (float), utilizzata per determinare se due attività sono significativamente diverse. La classe Metrics interagisce con le classi Task e ProjectComparator, inoltre, quest'ultima utilizza un'istanza di Metrics per confrontare le attività di due progetti e si affida ai metodi di Metrics per ottenere il punteggio di similarità tra coppie di attività e determinare se sono considerate simili o significativamente diverse.

In definitiva, Metrics è utile per quantificare la similarità tra attività mettendo a disposizione un insieme di metriche personalizzabili che consentono di valutare vari attributi delle attività, offrendo una visione dettagliata delle somiglianze e delle differenze esistenti. Inoltre, la progettazione modulare di Metrics permette di estendere o modificare facilmente le metriche utilizzate, adattandosi a diversi domini applicativi o esigenze specifiche.

3.4.10 ProjectPrinter.py



ProjectPrinter è una componente dedicata alla visualizzazione e alla presentazione delle informazioni relative a un progetto. Essa offre una serie di metodi che consentono di stampare dettagli del progetto, comprese le attività, le risorse, le dipendenze tra le attività, le differenze e la ricostruzione per passare da un progetto ad un altro. Questa classe risulta indispensabile per facilitare la comprensione dello stato del progetto e per comunicare efficacemente le informazioni agli utenti.

Presenta vari metodi per la presentazione delle informazioni permettendo una comunicazione chiara e strutturata dei dati del progetto.

Il metodo print_project_details(self) stampa i dettagli completi del progetto includendo informazioni quali il nome del progetto, il budget assegnato, il costo totale stimato, lo stato del budget, l'elenco delle attività del progetto, con dettagli su ciascuna attività come il nome, la durata, i predecessori, i successori e le risorse assegnate. Ciò consente di avere prospettiva completa del progetto facilitando la comprensione della sua struttura e delle sue componenti operative. Per i cambiamenti tra progetti, la funzione print_changes(self, changes) accetta come parametro un dizionario changes contenente informazioni sulle differenze tra due progetti stampando le attività aggiunte, rimosse, modificate e invariate e presentando dettagli per ciascuna categoria; print_task_differences(self, task_differences, resource_differences), invece, stampa le differenze specifiche tra due attività. Il parametro task_differences è una lista di differenze rilevate negli attributi dell'attività, mentre resource_differences è un dizionario contenente le differenze nelle risorse assegnate. Questo risulta particolarmente utile per evidenziare cambiamenti significativi tra versioni diverse della stessa attività, consentendo un'analisi dettagliata delle variazioni intervenute.

Per analizzare le differenze nelle risorse tra due progetti, il metodo print_resource_differences(self, resource_differences) stampa le informazioni sulle risorse aggiunte, rimosse o modificate, inclusi dettagli su quantità e costi semplificando la comprensione di come le risorse siano cambiate tra i progetti, contribuendo alla pianificazione e all'allocazione efficace delle risorse, mentre la funzione print_resources(self, resources) stampa l'elenco delle risorse disponibili nel progetto con le relative informazioni. Infine, print_resource_changes(self, old_resources, new_resources) confronta due insiemi di risorse, old_resources e new_resources, e stampa le differenze tra di essi evidenziando le risorse che sono state aggiunte, rimosse o che hanno subito modifiche nelle quantità o nei costi, aiutando a monitorare l'evoluzione delle risorse nel tempo.

Ricevendo un'istanza di Project, la classe utilizza i suoi attributi e metodi per accedere alle informazioni sul progetto, come l'elenco delle attività, il budget e il costo totale stimato. In aggiunta, accede alle istanze di Task contenute nel progetto per stampare i dettagli delle attività e le informazioni sulle risorse assegnate alle attività facilitando la comprensione dell'impiego delle risorse e contribuendo alla loro gestione efficiente.

In definitiva, la classe è l'unica responsabile della presentazione dei risultati degli algoritmi descritti in precedenza rispettando le linee guida dei principi SOLID.

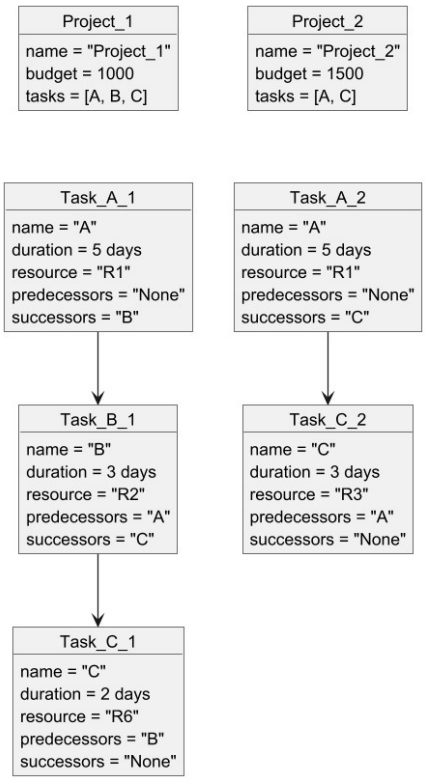
4. Risultati: esecuzioni interessanti dell’algoritmo

In questo capitolo vengono presentati e discussi alcuni esempi rilevanti di esecuzioni dell'algoritmo di confronto e differenziazione tra progetti. L'obiettivo è mostrare come l'algoritmo riesca a identificare le differenze tra due versioni di progetti, individuando aggiunte, rimozioni e modifiche nei task e nelle risorse. Ogni esempio sarà accompagnato da un object diagram dei progetti coinvolti e dall'output dell'algoritmo, fornendo così una rappresentazione visiva e una descrizione dettagliata di come la struttura del progetto evolve.

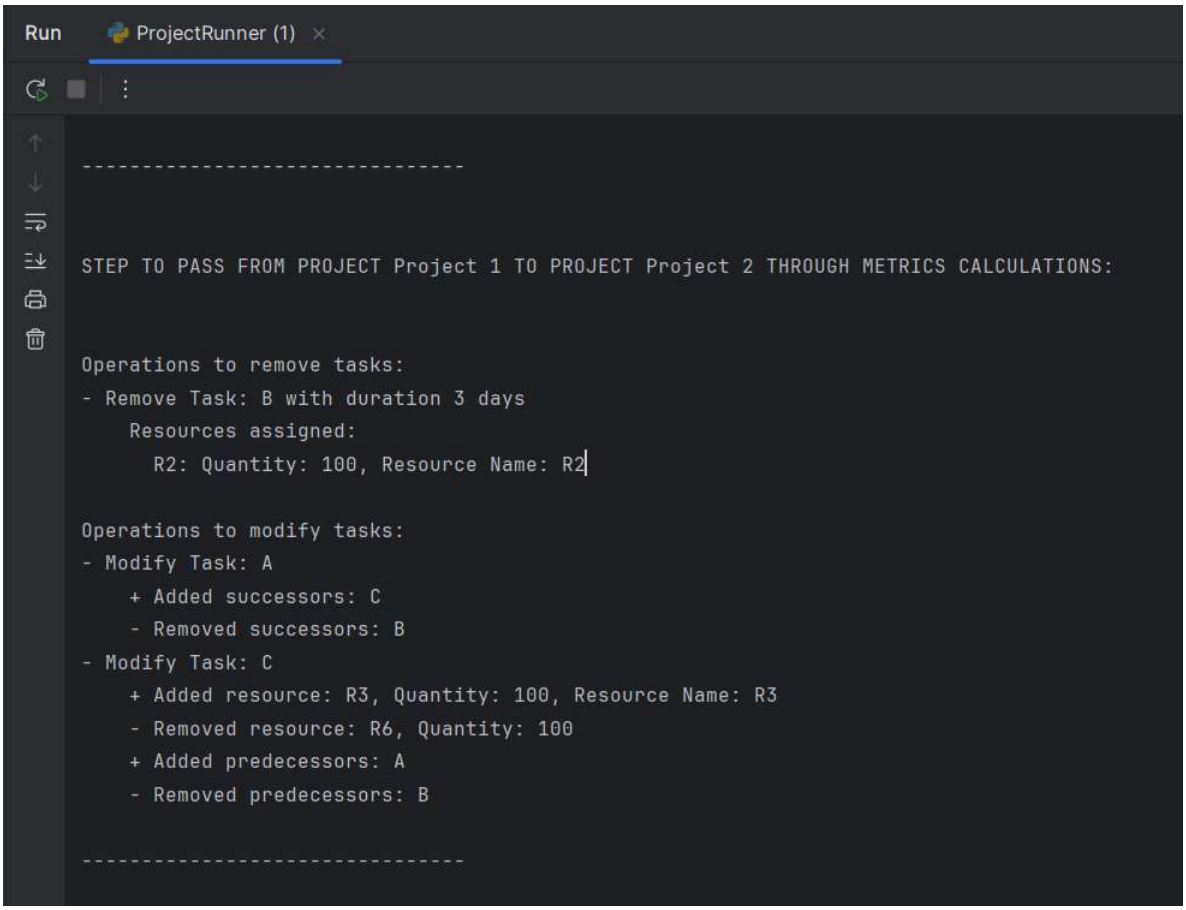
4.1. Esecuzione 1

In questo primo esempio si analizza la transizione da Project 1 a Project 2. Questa evoluzione mostra un cambiamento sostanziale nella struttura dei task, con la rimozione del task "B" e alcune modifiche importanti ai task "A" e "C". L'output mostra come l'algoritmo riesca a rilevare e descrivere ogni singolo cambiamento avvenuto nella transizione tra le due versioni.

Per meglio comprendere le differenze, vediamo prima l'object diagram dei due progetti. Project 1 è composto da tre task: "A", "B" e "C". In Project 2, il task "B" viene rimosso e i task rimanenti subiscono delle modifiche. Di seguito viene presentato l'object diagram di entrambi i progetti, che evidenzia la struttura e le relazioni tra le varie attività



L'output dell'algoritmo per il passaggio da Project 1 a Project 2 è il seguente:

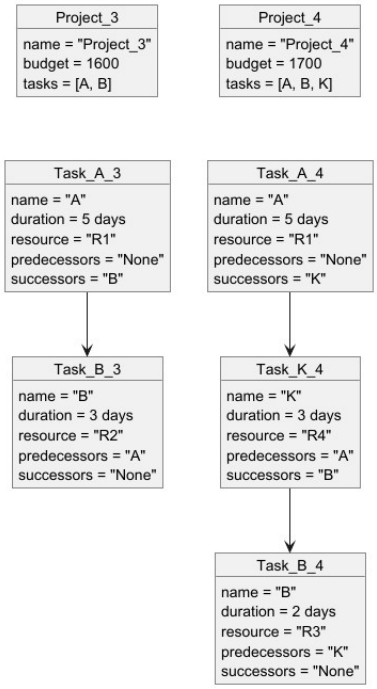


In questo esempio, l'algoritmo rileva che l'attività "B" viene completamente rimossa dal progetto. Questo comporta un impatto significativo sulla dipendenza tra le attività, con "A" che adesso ha come successore diretto "C", anziché "B". Inoltre, la modifica di "C" riguarda anche la risorsa associata, di conseguenza nel passaggio da Project 1 a Project 2 viene rimossa la risorsa "R6" e viene invece aggiunta la risorsa "R3". Questo può indicare un cambiamento nella tipologia di lavoro che il task "C" richiede, rappresentando una modifica significativa della natura del progetto.

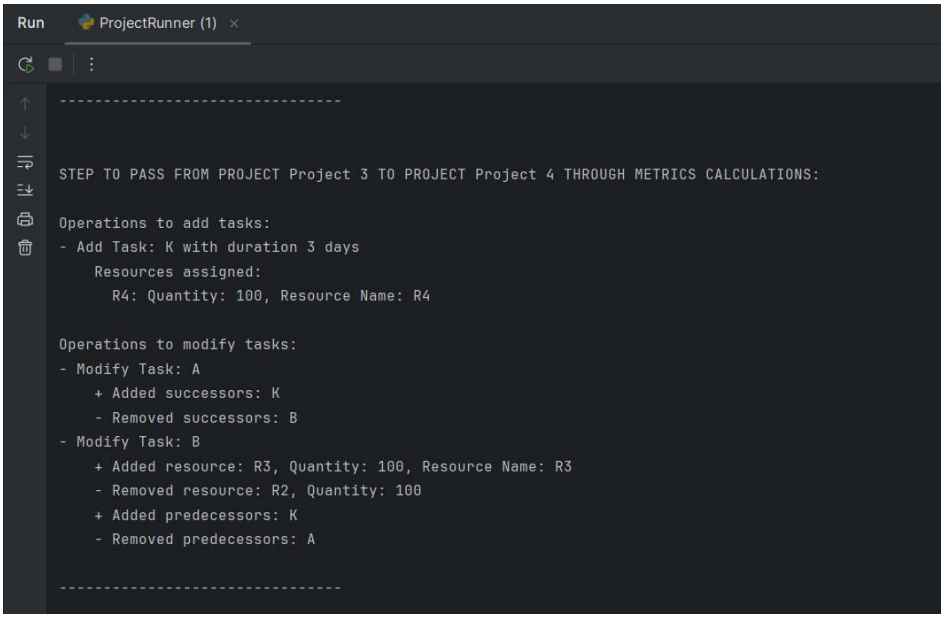
4.2. Esecuzione 2

Nel secondo esempio viene analizzata il passaggio da Project 3 a Project 4. La differenza principale è l'introduzione di un nuovo task "K" tra i task "A" e "B", che rappresenta quindi un cambiamento nella sequenza di attività del progetto, oltre al fatto che avviene una variazione nell'assegnazione delle risorse.

L'object diagram di Project 3 mostra due task principali: "A" e "B". In Project 4 invece, si aggiunge un nuovo task "K", modificando così la sequenza e la struttura del flusso di lavoro. Ecco la rappresentazione dei due progetti:



Il risultato dell'algoritmo per il passaggio da Project 3 a Project 4 è il seguente:

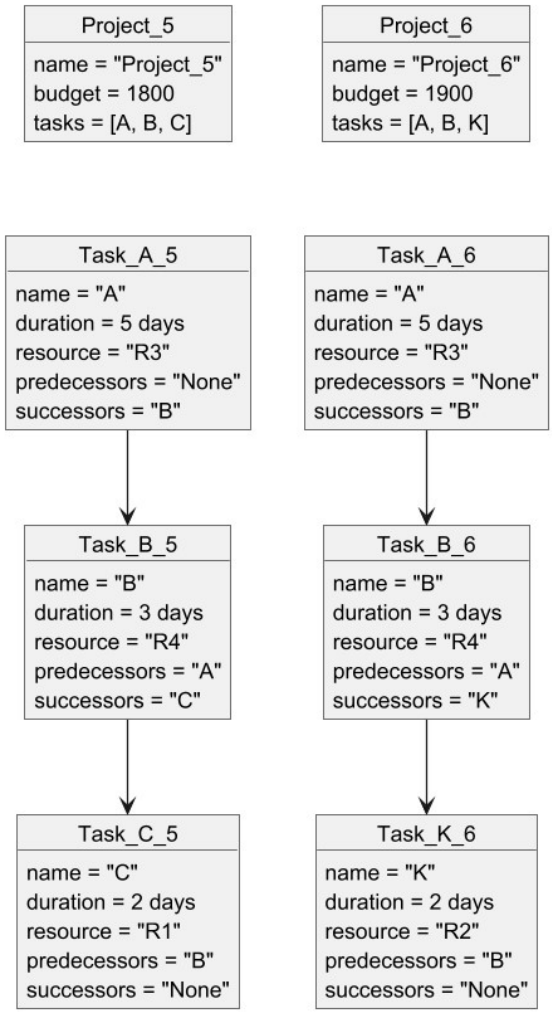


In questa esecuzione, l'algoritmo rileva l'aggiunta della nuova attività "K", posizionata tra "A" e "B". Questo nuovo task richiede una risorsa specifica, "R4", la quale rappresenta un elemento nuovo del progetto. La modifica del task "B" riflette il fatto che ora "B" dipende da "K" e non più direttamente da "A". Questo tipo di cambiamento è tipico nei progetti quando si aggiungono nuove fasi o attività intermedie, spesso per migliorare la qualità del risultato finale o per rispondere a nuove esigenze del progetto.

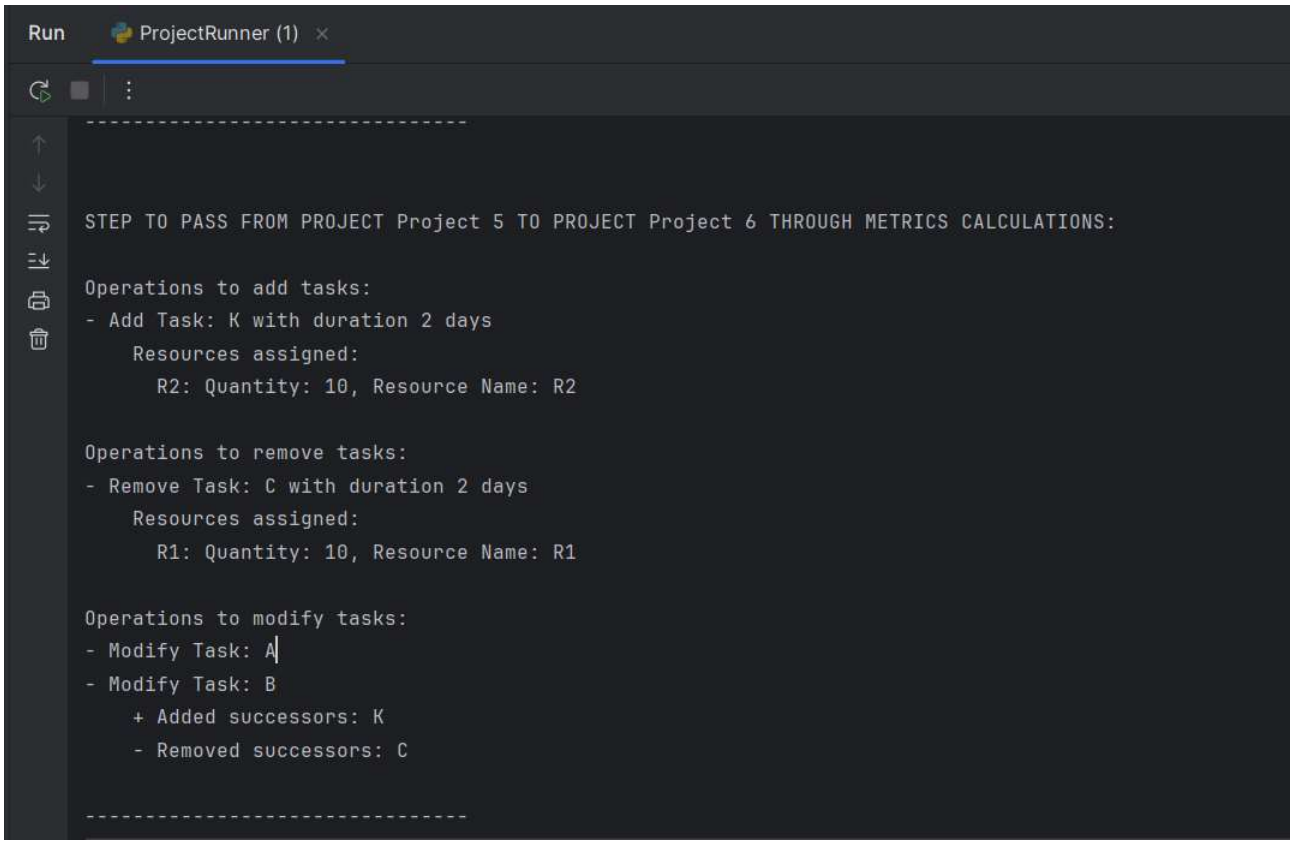
4.3. Esecuzione 3

Nel terzo esempio si analizza il passaggio da Project 5 a Project 6 che prevede sia l'aggiunta di un nuovo task "K" sia la rimozione del task "C". Le modifiche riguardano anche le dipendenze tra i task e la tipologia di risorse utilizzate.

L'object diagram di Project 5 include 3 attività: "A", "B" e "C". Nel passaggio a Project 6, il task "C" viene eliminato e un nuovo task "K" viene introdotto, come mostrato di seguito:



Il prodotto dell'algoritmo per il passaggio da Project 5 a Project 6 è il seguente:



L'algoritmo identifica la rimozione dell'attività "C" e l'aggiunta di "K". Questa sostituzione di task comporta anche una modifica delle risorse, con "K" che utilizza una risorsa differente rispetto a "C". Il cambiamento nelle dipendenze tra i task evidenzia un riadattamento della sequenza operativa: il task "B" non conduce più a "C", ma al nuovo task "K". Questo tipo di trasformazione può indicare un cambiamento nelle priorità del progetto o un adattamento a nuove specifiche esigenze, come la riduzione dei tempi o l'ottimizzazione dell'uso delle risorse.

Gli esempi sopra descritti dimostrano come l'algoritmo sia in grado di catturare le differenze strutturali e semantiche tra diverse versioni d progetti. Gli object diagram offrono una chiara rappresentazione visiva delle strutture, mentre il risultato dell'algoritmo fornisce una descrizione dettagliata dei cambiamenti e della logica che c'è dietro la rilevazione di essi. Questo approccio integrato permette non solo di comprendere le differenze, ma anche di valutare l'impatto di tali modifiche sul progetto complessivo, facilitando la gestione delle attività di project management.

4.4. Applicazioni pratiche del progetto sviluppato

Lo strumento si presta a svariate applicazioni pratiche del project management, sia in ambito accademico che lavorativo. Nel settore edile potrebbe essere utilizzato nel caso di un progetto di villette a schiera; infatti, la caratteristica intrinseca di un progetto di villette a schiera è di avere delle palazzine che differisco-no per pochi particolari l'una dall'altra. Con lo strumento sviluppato si potrebbero evidenziare queste differenze comprendendo, in maniera analitica, i passaggi necessari per passare da un progetto di un immobile all'altro ed evidenziando quelle che sono le differenze in termini di risorse, materiali e non, e costi. Al-tro settore in cui è possibile applicare la differenziazione semantica fra progetti è durante l'organizzazione di eventi, infatti, pensando a quegli eventi che si ripetono con cadenza periodica, come il Salone del Gusto o il Salone del Libro che si replicano ogni anno aggiungendo o rimuovendo particolari nella struttura dell'evento o dell'organizzazione. Ebbene, con lo strumento è possibile sottolineare le differenze ed an-che la variazione nei costi e nell'utilizzo delle risorse fra un'edizione e l'altra. Anche nel settore dell'Information Technology l'utilizzo di questo strumento offre numerosi vantaggi, per esempio, in un progetto di sviluppo software che prevede il rilascio di versioni successive di un prodotto in cui vengono introdotte nuove funzionalità e vengono risolti eventuali bug. Con questo strumento, è possibile confrontare le diverse versioni del progetto software evidenziando nel dettaglio quali funzionalità sono state aggiunte, modificate o rimosse e quali risorse sono state allocate per il completamento di ciascuna modifica. Un'altra possibile applicazione riguarda il settore della logistica e della gestione delle catene di fornitura. Pensiamo, ad esempio, a un'azienda che organizza il trasporto di beni su base periodica. Ogni piano logisti-co può variare a seconda delle esigenze specifiche del cliente, della quantità di beni da trasportare, dei mezzi utilizzati e delle destinazioni finali. Con lo strumento sviluppato, è possibile confrontare piani logistici simili evidenziando quali risorse aggiuntive siano state necessarie per soddisfare una richiesta particolare, o come l'ottimizzazione dei percorsi abbia influito sui costi complessivi di trasporto. In questo modo, l'azienda può ottimizzare i suoi processi operativi e ridurre al minimo i costi. Nel settore sanitario, questo strumento può trovare applicazione nel contesto della pianificazione ospedaliera, in particolare per la gestione delle risorse, ad esempio, la gestione dei reparti di terapia intensiva può differire da un periodo all'altro in base alla domanda, come durante picchi di epidemie o in periodi di bassa affluenza. Con questo strumento, i direttori sanitari possono analizzare le differenze nei piani di gestione dei reparti, identificando variazioni nei protocolli di cura, nei turni del personale o nei materiali medici necessari, ottimizzando così le risorse disponibili in modo più accurato. Un altro contesto di applicazione può essere individuato nell'ambito dell'insegnamento e della formazione professionale. Considerando un corso di formazione che viene adattato e aggiornato ogni anno, un programma di studi o un corso di specializzazione, l'analisi delle differenze semantiche permette di identificare chiaramente le modifiche introdotte nel syllabus, nei contenuti, nei moduli e nella loro durata. Questo consente ai coordinatori del corso di avere una visione precisa delle evoluzioni didattiche e di comunicare in modo trasparente le modifiche agli studenti e ai docenti consentendo, inoltre, di tracciare l'efficacia di ogni aggiornamento, mettendo in evidenza le risorse necessarie per ciascun cambiamento e valutando l'impatto sul processo di apprendimento. Infine, un'ulteriore applicazione pratica può essere riscontrata nei progetti di ricerca, che coinvolgono gruppi di lavoro interdisciplinari e internazionali. In questi ambiti, i progetti di ricerca sono spesso soggetti a revisioni e aggiornamenti e possono variare in base ai contributi dei diversi autori o in risposta alle scoperte emergenti. Con l'utilizzo di questo strumento, è possibile analizzare le differenze tra le varie versioni dei progetti, identificando quali componenti sono stati aggiunti o eliminati, e come le risorse della ricerca, sia umane che tecnologiche, siano state redistribuite. Questo tipo di analisi permette di documentare con precisione il percorso di sviluppo della ricerca, promuovendo la trasparenza e facilitando la cooperazione fra i soggetti che possono comprendere meglio l'evoluzione del progetto e allineare i propri contributi in modo più efficace.

Gli esempi sopra esposti dimostrano come il software implementato possa essere applicato in una vasta gamma di contesti, offrendo un valore aggiunto in termini di comprensione, ottimizzazione e gestione delle risorse, sia nei settori tradizionali che in quelli più innovativi.

5. Conclusioni

Il software sviluppato in questa tesi si propone come uno strumento per l'analisi delle differenze semantiche tra dati strutturati di progetti versatile che va dalla gestione operativa delle attività alla pianificazione, dall'evidenziare differenze tra progetti diversi alla ricostruzione dei passi necessari per passare da una struttura dati di un progetto ad un'altra. Il sistema sviluppato quindi si presenta come un tool potente per la gestione dei progetti e la sua struttura modulare e scalabile lo rende adatto a settori e contesti differenti. Grazie alla capacità di documentare dettagliatamente lo stato dei progetti e di fornire strumenti per l'analisi e il confronto delle diverse versioni, il sistema contribuisce a migliorare la trasparenza e la comunicazione tra le persone che lavorano al progetto e a facilitare il rapporto con gli stakeholder. La possibilità di integrare questo sistema anche in percorsi di formazione lo rende uno strumento prezioso non solo per le organizzazioni che vogliono migliorare i loro processi di gestione, ma anche per quelle realtà accademiche che desiderano offrire un'esperienza di apprendimento pratica e significativa. Come ogni progetto, anche il sistema presentato in questa tesi presenta alcune condizioni che potrebbero essere perfezionate in sviluppi futuri, infatti, l'algoritmo di comparazione, pur essendo efficace, potrebbe essere migliorato per garantire un'analisi ancor più accurata nel constatare le differenze tra le attività di progetto. Allo stato attuale l'algoritmo utilizza un metodo basato su indicatori come il nome dell'attività, la durata, i predecessori, i successori e le risorse. Sta di fatto che la qualità della comparazione potrebbe beneficiare di una maggiore considerazione del contesto semantico e delle interdipendenze tra le attività attraverso l'integrazione di tecniche di apprendimento automatico, come il machine learning, il quale potrebbe consentire al sistema di riconoscere in maniera più sofisticata le similitudini e le differenze tra le descrizioni delle attività portando a una valutazione più approfondita delle variazioni progettuali. Altri spunti di interesse potrebbero essere relativi alla costruzione di una interfaccia grafica che permetta di mostrare in maniera visuale i progetti caricati sul software sottoforma di reticolo, consentendo quindi di sottolineare le potenzialità del semantic diffing applicato ai progetti in maniera ancor più evidente. In quest'ultimo caso, il sistema sviluppato si comporterebbe come una componente di backend che fornisce le informazioni alla componente front-end, presupponendo di conseguenza anche la costruzione e l'esposizione di endpoint REST API che consentano la comunicazione tra le due componenti. Un ultimo sviluppo di notevole interesse potrebbe riguardare l'integrazione del sistema con altre piattaforme di gestione dei progetti, come i software di project management più comuni (ad esempio, Microsoft Project, Jira, Trello). Creando interfacce standardizzate per l'importazione e l'esportazione dei dati, il sistema potrebbe essere utilizzato come un componente di analisi avanzata integrato all'interno di una suite più ampia di strumenti aziendali. Questo migliorerebbe notevolmente la capacità del sistema di fornire un supporto concreto alle decisioni manageriali, rendendo le informazioni disponibili in modo fluido e immediato.

In definitiva, il lavoro di tesi prodotto rappresenta un primo passo importante verso una gestione dei progetti più informata, trasparente e ottimizzata, in grado di rispondere alle sfide crescenti del contesto competitivo attuale. Sebbene vi siano margini di miglioramento e numerosi aspetti che possono essere ulteriormente sviluppati, l'approccio proposto si è dimostrato efficace nel fornire un'analisi dettagliata e strutturata delle modifiche progettuali, migliorando la capacità di comprensione e gestione delle differenze tra versioni successive di un progetto o fra progetti differenti.

Bibliografia

- PMI (Project Management Institute), *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Sixth Edition*. Project Management Institute, Inc., 2017.
- PMI (Project Management Institute), *A Guide to the Project Management Body of Knowledge (PMBOK® Guide) – Seventh Edition*. Project Management Institute, Inc., 2021.
- H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. John Wiley & Sons.
- Schwalbe, K. (2016). *Information technology project management*. Cengage Learning.
- AXELOS, *Managing Successful Projects with PRINCE2® – 6th Edition*. TSO (The Stationery Office), 2017.
- PMI e Agile Alliance, *Guida alle Pratiche Agile (Agile Practice Guide)*. Project Management Institute, Inc., 2017.
- R. Turner e B. Lloyd-Walker, "Emotional intelligence (EI) capabilities training: can it develop EI in project teams?", *International Journal of Managing Projects in Business*, 2008.
- S. Horwitz, "Identifying the Semantic and Textual Differences Between Programs", in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, 1990.
- J.-R. Falleri, C. Morat, e X. Blanc, "Fine-Grained and Accurate Source Code Differencing", in *Proceedings of the 29th IEEE/ACM International Conference on Automated Software Engineering*, 2014.
- Dig, Danny, and Ralph Johnson. "The role of refactorings in API evolution." *21st IEEE International Conference on Software Maintenance (ICSM'05)*. IEEE, 2005.
- Nguyen, Hung Viet, et al. "Detecting semantic merge conflicts with variability-aware execution." *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. 2015.
- Martin, Robert C. "Clean architecture: *"A Craftsman's Guide to Software Structure and Design*
- *" Available online* (2017).
- Martin, Robert C. *Clean code: a handbook of agile software craftsmanship*. Pearson Education, 2009.
- Fowler, Martin. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

Appendice

Per una maggior comprensione del progetto sviluppato di seguito vengono elencate le parti di codice rilevanti:

Metrics.py

```
from thefuzz import fuzz


class Metrics:
    def __init__(self, threshold=0.7):
        self.threshold = threshold

    def calculate_similarity(self, task1, task2):
        cN = self.name_similarity(task1.name, task2.name)
        cD = self.duration_difference(task1.duration, task2.duration)
        cP = self.shared_predecessors(task1.predecessors, task2.predecessors)
        cS = self.shared_successors(task1.successors, task2.successors)
        cR = self.shared_resources(task1.resources, task2.resources)

        return cN * cD * cP * cS * cR

    @staticmethod
    def name_similarity(name1, name2):
        return fuzz.ratio(name1, name2) / 100

    @staticmethod
    def duration_difference(duration1, duration2):
        return 1 - abs(duration1 - duration2) / max(duration1, duration2)

    @staticmethod
    def shared_predecessors(predecessors1, predecessors2):
        common = set(predecessors1) & set(predecessors2)
        return len(common) / max(len(predecessors1), len(predecessors2), 1)

    @staticmethod
    def shared_successors(successors1, successors2):
        common = set(successors1) & set(successors2)
        return len(common) / max(len(successors1), len(successors2), 1)

    @staticmethod
    def shared_resources(resources1, resources2):
        common = set(resources1.keys()) & set(resources2.keys())
        return len(common) / max(len(resources1), len(resources2), 1)

    def is_significantly_different(self, task1, task2):
        return self.calculate_similarity(task1, task2) < self.threshold
```

ProjectComparator.py

```
from src.comparer.Metrics import Metrics


class ProjectComparator:
    def __init__(self, project1, project2, similarity_threshold=0.7):
        self.project1 = project1
        self.project2 = project2
        self.calculator = Metrics(threshold=similarity_threshold)

    def identify_changes(self):
        project1_task_names = [task.name for task in self.project1.tasks]
        project2_task_names = [task.name for task in self.project2.tasks]

        tasks_added = set(project2_task_names) - set(project1_task_names)
        tasks_removed = set(project1_task_names) - set(project2_task_names)

        changes = {
            'added': [],
            'removed': [],
            'modified': []
        }

        for task in self.project2.tasks:
            if task.name in tasks_added:
                changes['added'].append((task.name, task))

        for task in self.project1.tasks:
            if task.name in tasks_removed:
                changes['removed'].append((task.name, task))

        for task1 in self.project1.tasks:
            for task2 in self.project2.tasks:
                if task1.name == task2.name:
                    if self.calculator.is_significantly_different(task1, task2):
                        changes['modified'].append((task1.name, task1, task2))

        return changes
```

ProjectDiffAnalyzer.py

```
class ProjectDiffAnalyzer:
    def __init__(self, project1, project2):
        self.project1 = project1
        self.project2 = project2

    def analyze(self):
        task_differences, resource_differences = self.compare_projects()
        return task_differences, resource_differences

    def compare_projects(self):
        task_differences = []
        resource_differences = {"quantity_diffs": [], "cost_diffs": []}
        project1_tasks = {task.name: task for task in self.project1.tasks}
        project2_tasks = {task.name: task for task in self.project2.tasks}

        for task_name in project1_tasks:
            if task_name not in project2_tasks:
                task_differences.append(f"Task '{task_name}' is in Project 1 but
not in Project 2.")
        for task_name in project2_tasks:
            if task_name not in project1_tasks:
                task_differences.append(f"Task '{task_name}' is in Project 2 but
not in Project 1.")

        for task_name, task1 in project1_tasks.items():
            if task_name in project2_tasks:
                task2 = project2_tasks[task_name]
                task_diffs, res_diffs = self.compare_tasks(task1, task2)
                task_differences.extend(task_diffs)
                resource_differences["quantity_diffs"].extend(res_diffs["quan-
tity_diffs"])
                resource_differences["cost_diffs"].ex-
tend(res_diffs["cost_diffs"])

        return task_differences, resource_differences

    @staticmethod
    def compare_tasks(task1, task2):
        task_differences = []
        resource_differences = {"quantity_diffs": [], "cost_diffs": []}
        predecessors1 = {pred.name for pred in task1.predecessors}
        predecessors2 = {pred.name for pred in task2.predecessors}

        if predecessors1 != predecessors2:
            missing_in_p2 = predecessors1 - predecessors2
            missing_in_p1 = predecessors2 - predecessors1
            if missing_in_p2:
                task_differences.append(f"Task '{task1.name}' has predecessors
{missing_in_p2} missing in Project 2.")
            if missing_in_p1:
                task_differences.append(f"Task '{task2.name}' has predecessors
{missing_in_p1} missing in Project 1.")

        task1_resources = {res_name: details for res_name, details in task1.re-
sources.items()}
        task2_resources = {res_name: details for res_name, details in task2.re-
sources.items()}

        for res_name, details in task1_resources.items():
```

```
            if res_name not in task2_resources:
                resource_differences["quantity_diffs"].append(
                  f"Resource '{res_name}' is in '{task1.name}' in Project 1 but not in
Project 2.")
            else:

                if details["quantity"] != task2_resources[res_name]["quantity"]:
                    resource_differences["quantity_diffs"].append(
                        f"Resource '{res_name}' quantity differs for task '{task1.name}'
({details['quantity']} in "
                        f"Project 1 vs {task2_resources[res_name]['quantity']} in Project
2).")

                if details["resource"].cost != task2_resources[res_name]["re-
source"].cost:
                    resource_differences["cost_diffs"].append(
                        f"Resource '{res_name}' cost differs for task '{task1.name}'
({details['resource'].cost} in "
                        f"Project 1 vs {task2_resources[res_name]['resource'].cost} in
Project 2).")

        for res_name, details in task2_resources.items():
            if res_name not in task1_resources:
                resource_differences["quantity_diffs"].append(
                    f"Resource '{res_name}' is in '{task2.name}' in Project 2 but not in
Project 1."
                )

        return task_differences, resource_differences

    def compare_resources_between_projects(self):
        resource_differences = {"quantity_diffs": [], "cost_diffs": []}
        project1_resources = self.aggregate_project_resources(self.project1)
        project2_resources = self.aggregate_project_resources(self.project2)

        for res_name, res_details in project1_resources.items():
            if res_name not in project2_resources:
                resource_differences["quantity_diffs"].append(f"Resource '{res_name}' is
unique in Project 1.")
            else:
                if res_details["total_quantity"] != project2_resources[res_name]["to-
tal_quantity"]:
                    diff = abs(res_details["total_quantity"] - project2_re-
sources[res_name]["total_quantity"])
                    resource_differences["quantity_diffs"].append(
                        f"Resource '{res_name}' has a quantity difference of {diff}.")
                if res_details["cost"] != project2_resources[res_name]["cost"]:
                    resource_differences["cost_diffs"].append(
                        f"Resource '{res_name}' cost per unit differs: Project 1 =
{res_details['cost']}, Project 2 = "
                        f"{project2_resources[res_name]['cost']}.")

        for res_name in project2_resources.keys() - project1_resources.keys():
            resource_differences["quantity_diffs"].append(f"Resource '{res_name}' is
unique to Project 2.")

        return resource_differences

    @staticmethod
    def aggregate_project_resources(project):
        aggregated_resources = {}
        for task in project.tasks:
            for res_name, details in task.resources.items():
                if res_name not in aggregated_resources:
                    aggregated_resources[res_name] = {"total_quantity": 0, "cost": de-
tails["resource"].cost}
                aggregated_resources[res_name]["total_quantity"] += details["quantity"]
        return aggregated_resources
```

ProjectInitializer.py

```
from src.model.Project import Project
from src.model.Resource import Resource
from src.model.Task import Task


class ProjectInitializer:
    def __init__(self, project_name, budget):
        self.project = Project(project_name)
        self.project.set_budget(budget)
        self.resources = {}
        self.tasks = {}

    def add_resource(self, name, max_allocation, cost, resource_type):
        self.resources[name] = Resource(name, max_allocation, cost, re-
source_type)

    def create_task(self, task_name, duration, predecessors=None):
        new_task = Task(task_name, duration)
        self.tasks[task_name] = new_task
        if predecessors:
            for pred_name in predecessors:
                pred_task = self.find_task_by_name(pred_name)
                if pred_task:
                    new_task.add_predecessor(pred_task)

    def assign_resource_to_task(self, resource_name, task_name, quantity):
        resource = self.resources[resource_name]
        task = self.tasks.get(task_name)
        if task:
            task.add_resource(resource, quantity)

    def finalize(self):
        for task in self.tasks.values():
            self.project.add_task(task)
        return self.project

    def find_task_by_name(self, name):
        return self.tasks.get(name)
```

Project.py

```
class Project:
    def __init__(self, name):
        self.name = name
        self.tasks = []
        self.budget = 0

    def add_task(self, task):
        self.tasks.append(task)

    def calculate_total_cost(self):
        return sum(
            task.resources[res]["resource"].cost * task.resources[res]["quan-
tity"] for task in self.tasks for res in
            task.resources)

    def set_budget(self, budget):
        self.budget = budget

    def budget_status(self):
        total_cost = self.calculate_total_cost()
        return f"Budget: {self.budget}, Total Cost Estimation: {total_cost}, Re-
sidual budget: {self.budget - total_cost}"
```

Task.py

```
class Task:
    def __init__(self, name, duration, predecessors=None):
        self.name = name
        self.duration = duration
        self.predecessors = predecessors if predecessors is not None else []
        self.successors = []
        self.resources = {}
        self.status = "Not Started"
        self.subtasks = []

    def add_predecessor(self, task):
        if task not in self.predecessors:
            self.predecessors.append(task)
        if self not in task.successors:
            task.successors.append(self)

    def add_resource(self, resource, quantity):
        if resource.name in self.resources:
            current_allocation = self.resources[resource.name]["quantity"]
            resource.allocate(quantity - current_allocation)
            self.resources[resource.name]["quantity"] = quantity
        else:
            resource.allocate(quantity)
            self.resources[resource.name] = {"resource": resource, "quantity":
quantity}

    def add_subtask(self, task):
        self.subtasks.append(task)
```

Resource.py

```
class Resource:
    def __init__(self, name, max_allocation, cost, resource_type):
        self.name = name
        self.max_allocation = max_allocation  # max availability of the resource
        self.allocated = 0  # resource allocation
        self.cost = cost
        self.resource_type = resource_type  # resource type: material, human

    def allocate(self, quantity):
        if self.allocated + quantity > self.max_allocation:
            raise ValueError(
                f"Unable to allocate {quantity} units of '{self.name}' because
it would exceed the maximum allocation "
                f"of {self.max_allocation}. Currently allocated: {self.allo-
cated}.")

        self.allocated += quantity

    def release(self, quantity):
        self.allocated -= quantity
        if self.allocated < 0:
            self.allocated = 0

    def current_availability(self):
        return self.max_allocation - self.allocated
```

ProjectPrinter.py

```
class ProjectPrinter:
    def __init__(self, project=None):
        self.project = project

    def print_project_details(self):
        if not self.project:
            print("No project specified for printing details.")
            return

        print(f"Project Name: {self.project.name}")
        print(f"Budget Status: {self.project.budget_status()}")
        print("Tasks and Resources:")
        for task in self.project.tasks:
            predecessors_names = [pre.name for pre in task.predecessors]
            successors_names = [suc.name for suc in task.successors]

            print(f"\n  Task: {task.name}, Duration: {task.duration} days")
            print(f"    Predecessors: {', '.join(predecessors_names) if prede-
cessors_names else 'None'}")
            print(f"    Successors: {', '.join(successors_names) if succes-
sors_names else 'None'}")

            for res_name, details in task.resources.items():
                resource = details["resource"]
                quantity = details["quantity"]
                total_cost = resource.cost * quantity
                print(
                    f"    Resource: {res_name}, Quantity: {quantity}, Cost per
unit: {resource.cost},"
                    f" Total Cost: {total_cost}")
```

```
    @staticmethod
    def print_task_differences(task_differences, resource_differences):
        has_differences = (task_differences or
                           resource_differences["quantity_diffs"] or
                           resource_differences["cost_diffs"])

        if has_differences:
            print("Differences between Project 1 and Project 2:")
            if task_differences:
                print("\nDifferences in Tasks:")
                for diff in task_differences:
                    print(f"- {diff}")
            if resource_differences["quantity_diffs"]:
                print("\nDifferences in Resource Quantities for Corresponding
Tasks:")
                for diff in resource_differences["quantity_diffs"]:
                    print(f"- {diff}")
            if resource_differences["cost_diffs"]:
                print("\nDifferences in Resource Costs for Corresponding
Tasks:")
                for diff in resource_differences["cost_diffs"]:
                    print(f"- {diff}")
        else:
            print("No differences found between Project 1 and Project 2.")

    @staticmethod
    def print_resource_differences(resource_differences):
        has_differences = resource_differences["quantity_diffs"] or re-
source_differences["cost_diffs"]

        print("\nAggregate Resource Differences between Project 1 and Project
2:")
        if has_differences:
            if resource_differences["quantity_diffs"]:
                print("\nDifferences in Resource Quantities:")
                for diff in resource_differences["quantity_diffs"]:
                    print(f"- {diff}")
            if resource_differences["cost_diffs"]:
                print("\nDifferences in Resource Costs:")
                for diff in resource_differences["cost_diffs"]:
                    print(f"- {diff}")
        else:
            print("No significant resource differences found.")

    def print_changes(self, changes):
        if changes['added']:
            print("Operations to add tasks:")
            for task_name, task in changes['added']:
                print(f"- Add Task: {task_name} with duration {task.duration}
days")
                self.print_resources(task.resources)

        if changes['removed']:
            print("\nOperations to remove tasks:")
            for task_name, task in changes['removed']:
                print(f"- Remove Task: {task_name} with duration {task.duration}
days")
                self.print_resources(task.resources)

        if changes['modified']:
```

```
            print("\nOperations to modify tasks:")
            for task_name, old_task, new_task in changes['modified']:
                print(f"- Modify Task: {task_name}")
                self.print_resource_changes(old_task.resources, new_task.re-
sources)

                relations = [("predecessors", old_task.predecessors,
new_task.predecessors),
                             ("successors", old_task.successors, new_task.suc-
cessors)]

                for relation_type, old_relations, new_relations in relations:
                    old_set = set([rel.name for rel in old_relations])
                    new_set = set([rel.name for rel in new_relations])

                    if old_set != new_set:
                        added_relations = new_set - old_set
                        removed_relations = old_set - new_set

                        if added_relations:
                            print(f"    + Added {relation_type}: {',
'.join(added_relations)}")
                        if removed_relations:
                            print(f"    - Removed {relation_type}: {',
'.join(removed_relations)}")

    @staticmethod
    def print_resources(resources):
        if resources:
            print("    Resources assigned:")
            for res_name, details in resources.items():
                print(f"      {res_name}: Quantity: {details['quantity']}, Re-
source Name: {details['resource'].name}")
        else:
            print("    No resources assigned.")

    @staticmethod
    def print_resource_changes(old_resources, new_resources):
        for res_name, details in new_resources.items():
            if res_name not in old_resources:
                print(
                    f"    + Added resource: {res_name}, Quantity: {de-
tails['quantity']}, Resource Name: "
                    f"{details['resource'].name}")
            elif old_resources[res_name]['quantity'] != details['quantity']:
                print(
                    f"    * Changed resource {res_name} from Quantity: {old_re-
sources[res_name]['quantity']}"
                    f" to Quantity: {details['quantity']}")

        for res_name in old_resources:
            if res_name not in new_resources:
                print(f"    - Removed resource: {res_name}, Quantity: {old_re-
sources[res_name]['quantity']}")
```