UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Triennale in Informatica

TESI DI LAUREA

Integrazione di ChatGPT per Migliorare la Documentazione degli Ingegneri del Software

RELATORE

Prof. Fabio Palomba

Dott. Vincenzo De Martino

Università degli Studi di Salerno

CANDIDATO

Carmine Iemmino

Matricola: 0512109893

Anno Accademico 2022-2023

Questa tesi è stata realizzata nel

L'universo è mutamento: la nostra vita è come la creano i nostri pensieri.

-Marco Aurelio

Abstract

Questa tesi discute l'applicazione e integrazione di ChatGPT nelle attività di software engineering, in particolare la stesura di documentazione. Studi simili nello stato dell'arte si focalizzano solo su alcune specifiche task di software engineering e non considerano la scrittura di documentazione nella sua totalità. L'obiettivo della tesi consiste nel valutare il supporto che ChatGPT possa fornire nella creazione di documentazione, dalla raccolta dei requisiti fino al testing, e l'impegno necessario per ottenere le risposte volute dal chatbot. Ciò verrà fatto immettendo richieste a ChatGPT, validando le soluzione proposte confrontando la documentazione risultante con un progetto software, Rojina Review. I risultati ottenuti evidenziano le grandi capacità del chatbot nel creare la documentazione di Ingegneria del Software, diminuendo notevolmente il tempo necessario per svolgere alcune attività; tuttavia non è ancora abbastanza performante per sostituire completamente un ingegnere del software.

Indice

Elenco delle figure

CAPITOLO 1

Introduzione

In questo capitolo viene fatta una presentazione del lavoro di tesi svolto, discutendo dell'ambito di applicazione, lo scopo e i risultati ottenuti.

1.1 Contesto applicativo

Per la realizzazione di documentazione corretta e di alta qualità nell'ingegneria del software sono necessari processi lunghi e complessi. Quest'ultimi richiedono una grande esperienza nel campo informatico e con una corretta documentazione sarà più semplice realizzare un software di qualità. Per facilitare e velocizzare questi processi è possibile applicare tecniche di intelligenza artificiale per agevolare tutte le fasi dello sviluppo software come: raccolta e analisi dei requisiti, in cui i bisogni del cliente che ha commissionato il prodotto vengono esplicitati e studiati; system design, in cui viene decisa la decomposizione in sistemi più piccoli del prodotto e l'architettura software da adottare; object design, in cui vengono delineate le interfacce delle classi che comporranno il sistema e gli elementi di riuso; testing, in cui viene verificato il corretto funzionamento del prodotto. Uno strumento di intelligenza artificiale altamente popolare in questo momento è ChatGPT, chatbot sviluppato da OpenAI basato sui large language models (LLMs) GPT, capaci di comprendere richieste

complesse e fornire risposte accurate. Il bot si sta rivelando formidabile in numerosi ambiti e con esso sta emergendo la figura professionale del prompt engineer, persona esperta nell'interagire e sviluppare richieste apposite per ottenere risposte efficaci e quanto migliori possibili dai LLMs come GPT. Un prompt engineer deve avere conoscenza delle capacità e delle limitazioni dello specifico large language model con cui ha a che fare, interpretando inoltre i risultati ottenuti. La loro esperienza acquisita potrebbe anche rivelare nuovi casi d'uso dei modelli, non facilmente individuabili da persone non esperte. Questa figura acquisirà un ruolo molto importante nei prossimi anni, se non già prossimi mesi.

1.2 Motivazioni e Obiettivi

Le motivazioni e obiettivi di questo lavoro di tesi risiedono nel testare l'efficacia di ChatGPT come assistente virtuale nelle attività di software engineering. In particolare si è interessati a valutare il grado di aiuto che il bot possa fornire agli sviluppatori e l'impegno necessario per utilizzarlo al meglio e ottenere risposte accettabili, usando come riferimento un progetto software già esistente, Rojina Review, sito web di notizie e recensioni su videogiochi sviluppato durante il corso di laurea. Lo studio verrà attuato tramite un'analisi empirica in cui verranno prese in considerazione la qualità delle risposte generate, l'aderenza agli standard del software engineering e il numero di prompts e/o correzioni manuali necessarie per ottenere una risposta riportabile nei documenti dello sviluppo software. L'operato di ChatGPT verrà quindi valutato in maniera critica, evidenziando eventuali punti di forza e limitazioni, delineando al contempo gli approcci migliori per interagire con il bot.

1.3 Risultati ottenuti

ChatGPT si è rivelato uno strumento molto utile nell'agevolare la stesura di documentazione software[1], permettendo di replicare quasi per intero il progetto di riferimento Rojina Review, impiegando indicativamente lo stesso tempo ma necessitando solo di una persona umana invece che tre. A causa degli errori in cui potrebbe

[1]Qui è disponibile la documentazione completa generata grazie all'aiuto di ChatGPT.

occasionalmente incorrere non è ancora capace di sostituire per intero la figura del software engineer, necessitando supervisione e a volte di correzioni manuali che non saprebbe attuare da solo anche con prompts mirati. L'impegno richiesto per utilizzare il bot nelle attività di software engineering si è rivelato essere generalmente medio, diminuendo man mano che si acquisisce esperienza usandolo, comprendendo i limiti della sua comprensione dei prompts e generazione di risposte.

1.4 Struttura della tesi

La seguente tesi è strutturata in questo modo: nel capitolo 2 è presentato lo stato dell'arte in ambito di intelligenza artificiale applicata al software engineering, con particolare focus sull'uso di large language models e chatbots per agevolare lo sviluppo software. La metodologia usata è discussa nel capitolo 3: essa descrive gli approcci adottati per utilizzare ChatGPT nelle diverse fasi di software engineering e alcuni esempi di prompts utilizzati, introducendo inoltre le domande di ricerca dello studio. Nel capitolo 4 vengono esplicitati i risultati derivanti dalla metodologia applicata: le repliche ai prompts date dal bot vengono discusse e viene data una risposta alle domande di ricerca. Infine è presente il capitolo 5, in cui vengono trattate le conclusioni e sviluppi futuri.

CAPITOLO 2

Background e stato dell'arte

In questo capitolo viene fornita una panoramica dello stato dell'arte in ambito di intelligenza artificiale applicata all'ingegneria del software, soffermandoci in particolare sull'uso dei large language models.

2.1 Intelligenza artificiale per l'ingegneria del software

L'intelligenza artificiale è una tecnologia che sta entrando con forza nella vita di tutti noi: dall'assistenza sanitaria, alla finanza, dalla manifattura al trasporto, dall'agricoltura all'educazione, dalla sicurezza ai servizi legali fino ad arrivare anche all'intrattenimento. Ogni ambito esistente sta traendo profitto e vantaggi dal suo utilizzo[1]. Un particolare ambito di interesse in cui può essere utilizzata l'intelligenza artificiale è l'ingegneria e lo sviluppo di software dando vita alla branca di ricerca chiamata AI4SE (Artificial Intelligence for Software Engineering). Questo settore si propone di fornire tool basati su AI per supportare tutte le fasi dell'ingegneria del software, a partire dall'analisi e raccolta dei requisiti fino al testing, manutenzione ed evoluzione[2].
Perini et al.[3] hanno affrontato il problema dell'assegnare la priorità giusta ai requisiti software: bassa, media o alta sono aggettivi fondamentali da associare ai requisiti,

che definiscono la strategia per lo sviluppo del software in esame. Il loro studio descrive un metodo di prioritizzazione dei requisiti chiamato Case-Based Ranking (CBRank) che offre un ordinamento dei requisiti basato sui continui feedback degli stakeholders. In particolare viene anche usata una componente di machine learning per ridurre il numero di feedback e informazioni necessarie ad ottenere un ordinamento corretto, aumentando la velocità con cui avviene il processo di prioritizzazione. Il metodo è stato testato variando il numero di requisiti da prioritizzare e il numero di feedback raccolti. Grazie al modulo di machine learning capace di approssimare il ranking dei requisiti dai feedback raccolti, CBRank è stato capace di prioritizzare oltre 100 requisiti. Limitazioni a questo studio sono la gestione delle dipendenze esistenti fra requisiti e l'aggiornamento del ranking quando nuovi requisiti vengono aggiunti o esistenti rimossi. Oltre ad avere un grado di priorità, i requisiti possono appartenere a diverse categorie. Esempi di categorie sono funzionali, usabilità, affidabilità, perfomance, supportabilità, etc. Un problema che nasce quindi, soprattutto in progetti di grandi dimensioni, è classificare i requisiti e porli nella giusta categoria. Uno studio in merito è stato fatto da Navarro et al.[4] che hanno usato tecniche di Deep Learning come soluzione. In particolare sono state usate le reti CNN, addestrate sul dataset PROMISE, contenente requisiti funzionali e 11 altre categorie di requisiti non funzionali. In base ai risultati ottenuti di precision e recall (rispettivamente di 0.80 e 0.785), si è concluso che il deep learning possa essere effettivamente utilizzato per ridurre notevolmente il tempo speso a classificare i requisiti. Uno studio simile di classificazione dei requisti è stato fatto da Rahman et al.[5] in cui nuovamente il dataset PROMISE è stato utilizzato ma con una diversa rete neurale, recurrent neural network (RNN) invece che convolutional neural network (CNN). In questo lavoro è stata evidenziato il predominio delle RNN rispetto alle CNN nel task di classificazione, con una precision e recall superiore (rispettivamente 0.973 e 0.967) in confronto allo studio effettuato da Navarro et al.[4]. Le limitazioni evidenziate da entrambi gli studi sono la mancanza di grandi dataset pubblici in ambito di machine learning applicato al software engineering. Un altro aspetto cruciale nella raccolta dei requisti è la presenza di faults al loro interno, come ad esempio ambiguità, omissione e inconsistenza di informazioni, affermazioni non corrette, etc. Il processo di individuazione e correzione dei faults può essere altamente time-consuming, in pro-

porzione alla grandezza del sistema in esame. Il problema è trattato da Singh et al.[6] in cui viene proposto un approccio orientato al machine learning per individuare e classificare i faults in maniera automatica. Sono stati testati nello studio diversi tipi di classificatori: bayesiani, support vector, alberi di decisione ed ensemble. I risultati fanno emergere una generale superiorità degli ensemble con una accuracy di 0.89. In particolare i classificatori ensemble sono stati performanti nel individuare fault che riguardassero ambiguità, affermazioni scorrette e informazioni inconsistenti. Gli altri classificatori individuali sono stati più accurati nell'individuare fault che riguardassero omissioni invece. Le limitazioni dello studio si sintetizzano nell'avere requisiti scritti in linguaggio naturale corretto senza errori di battitura e il basso numero di alcuni tipi di fault nei dati di addestramento: entrambi questi problemi abbassano le performance dei classificatori. Sviluppi futuri dello studio riguardano quindi la raccolta di ulteriori dati per addestrare più efficacemente i modelli di machine learning. Zhao e Huang[7] affrontano invece il problema del code functional similarity, più complesso rispetto al semplice code similarity che si focalizza solo sulla sintattica del codice e non sulla sua semantica e le funzioni che realizza. Misurare questa metrica tra diversi pezzi di codice renderebbe più semplice il refactoring e il riuso. Una soluzione proposta dagli autori dello studio è un tool chiamato DeepSim, basato su una feed-forward neural network. A differenza degli altri tool di code similarity esistenti, le features della rete neurale non sono ricavate a partire dai token in cui è stato scomposto il codice, ma bensì dal flusso di controllo e flusso di dati: questo più alto livello di astrazione permette di comprendere meglio la semantica del codice. Il tool è stato testato con due dataset, uno di questi è BigCloneBench. La metrica F-1 risultante per quest'ultimo è stata molta alta, con un punteggio di 0.98. Le limitazioni all'approccio sono dovute alla grandezza del dataset: BigCloneBench copre solo 10 funzionalità. Il codice può essere inoltre analizzato per individuare la presenza di design patterns[1] all'interno di esso, andando a costituire una parte importante del processo di reverse engineering, migliorando la manutenibilità e documentazione del codice sorgente. Questo problema di individuazione è trattato da Dwivedi et al.[8] il quale approccio di soluzione si basa sulle RNN. In particolare è stato usato un apprendimento supervisionato per individuare i patterns Abstract Factory e Adap-

[1] Design pattern: template di soluzione ad un problema ricorrente

ter. I dati di addestramento del classificatore sono metriche object-oriented come efficienza, complessità, riusabilità, etc. Il testing del modello sono risultati in una precision e recall pari a 1.00. Un lavoro simile è stato portato avanti da Thaller et al.[9] in cui invece delle RNN, sono usate le CNN e i design pattern considerati sono Singleton, Template, Composite e Decorator. La metrica F-1 risultante è rispettivamente 0.76, 0.58, 0.82 e 0.72. Limitazioni evidenziate in questo studio riguardano la qualità del dataset che è formato prevalentemente da progetti software che utilizzano vecchie versioni di Java, in cui alcuni design pattern, come Singleton, presentavano implementazioni molto diverse rispetto a quelle presenti oggi.

2.2 Large language models e chatbots per l'ingegneria del software

Il software engineering è pieno di dati non strutturati sotto forma di testo in linguaggio naturale che possono essere elaborati da tool e tecniche di NLP per semplificare lo sviluppo e la stesura della documentazione del software. Il Natural Language Processing (abbreviato in "NLP" e traducibile in "Elaborazione del Linguaggio Naturale") è una branca dell'intelligenza artificiale che permette ai computer di riconoscere, comprendere e generare il linguaggio umano. La prima capacità è denominata "Speech Recognition", la seconda "Natural Language Understanding", la terza "Natural Language Generation". Gli assistenti vocali, come Alexa e Siri, sono un esempio di tecnologia che utilizza queste tre abilità dell'NLP: in primo luogo l'assistente vocale riconosce il linguaggio parlato dell'utente, in seguito effettua le elaborazioni necessarie per comprendere il significato di quanto detto e in ultimo genera una risposta comprensibile dall'utente, in linguaggio naturale. Altri strumenti in cui viene utilizzato l'NLP sono i motori di ricerca, la traduzione di testi, sintesi di documenti, controllo grammaticale, sentiment analysis e chatbots[10]. L'NLP può avvalersi di diversi tipi di reti neurali di deep learning. Le migliori sono le RNN e seq2seq, e i transformers, stato dell'arte in ambito di NLP e che permettono l'esistenza dei large language models (LLM), capaci di fagocitare quantità abnormi di dati testuali e derivarne relazioni e dipendenze. Grazie al continuo aumento di

potenza computazionale disponibile e la conseguente possibilità di aumentare la dimensione dei dataset di addestramento e il numero di parametri, questi modelli sono migliorati molto negli ultimi 5 anni, attirando notevole interesse[11]. La maggior parte degli LLM è integrata in chatbots, software che simulano la conversazione umana, permettendo di interfacciarci ai dispositivi digitali come se stessimo parlando con una persona reale[12]. Gli esempi più famosi di chatbot che sfruttano LLM sono Bard e ChatGPT. Il primo è sviluppato da Google e basato sul large language model LaMDA (Language Models for Dialogue Applications)[13]. Il secondo invece è sviluppato da OpenAI e basato sui large language models GPT (Generative Pre-training Transformer). L'enorme dataset di addestramento utilizzato (570 GB nel caso di GPT-3), comprendente di libri, articoli e siti web, permette virtualmente al chatbot di rispondere a qualsiasi richiesta che preveda un output in formato testuale[14]. L'uso di ChatGPT come assistente allo sviluppo software è trattato da Tian et al.[15] in cui vengono testate le abilità del bot nella generazione, riparazione e sintesi di codice su problemi di programmazione comuni come ricerca sequenziale, ordinamento etc. Nello studio è inoltre effettuato un confronto con altri due LLMs, Codex e CodeGen. I risultati fanno emergere il dominio di ChatGPT rispetto a quest'ultimi ma anche la sua difficoltà nell'affrontare problemi mai visti prima. È inoltre emerso l'impatto negativo di lunghi prompts sulle sue capacità di deduzione delle informazioni. Una limitazione allo studio riguarda la casualità delle risposte date: uno stesso prompt/input potrebbe generare risposte diverse. Questa è una caratteristica comune a tutti gli LLM e non solo a ChatGPT. Questa problematica viene però mitigata dallo studio inserendo più volte lo stesso prompt e definendo una metrica basata sulla media di 5 risposte. Uno studio simile, ma con più task di software engineering analizzate, è stato fatto da Sridhara et al.[16] in cui le risposte date da ChatGPT sono state confrontate con risposte date da software engineer esperti. Nei risultati emersi il bot si è rivelato molto utile nella code clone detection e nella sintesi di log e codice sorgente, senza necessitare correzioni. In attività come la creazione di messaggi di commit e review di codice vi è bisogno di controllare il suo output ed eventualmente apportare correzioni. Ahmad et al.[17] hanno invece valutato le performance di ChatGPT in ambito di system design, usando un caso di studio reale di un software basato su un'architettura a servizi chiamato CampusBike.

Dai risultati è emersa la sua capacità di impersonare un architetto software, che però ha comunque bisogno di supervisione e guida verso le decisioni per essere usato nel design di un'architettura. Ross et al.[18] hanno creato invece il proprio chatbot dotato di LLM, definendo il prototipo "the Programmer's Assistant". Quest'ultimo è stato creato con l'intento di sviluppare uno strumento conversazionale che avesse consapevolezza delle precedenti interazioni avute, invece che fornire una risposta esclusivamente in base all'input corrente. Il tool è stato implementato tramite Codex, LLM addestrato appositamente su codice sorgente, e poi inserito nel contesto di un ambiente di sviluppo. Il prototipo è stato valutato tramite la partecipazione di 42 persone, con differenti livelli di programmazione. I partecipanti, nei loro feedback, hanno sottolineato la capacità del tool di produrre codice, spiegarlo e di rispondere a domande generali riguardanti la programmazione; però, come sottolineato dallo studio, the Progammer's Assistant non genera sempre codice perfetto o risposte corrette.

CAPITOLO 3

Metodologia

In questo capitolo viene descritta la metodologia applicata, evidenziata in figura 3.1, il cui obiettivo è stato valutare l'operato di ChatGPT nelle tipiche attività di software engineering. Il bot, nell'ultimo periodo, ha attirato l'interesse di molte persone, rivelandosi utile in molti task come creazione di contenuti, supporto all'educazione, scrittura di e-mail, etc.[19] Da ciò è derivata la curiosità di testare le sue capacità in ambito di software engineering. È stato usato come riferimento e confronto Rojina Review, un sito web di recensioni e notizie su videogiochi con e-commerce, sviluppato da me e alcuni miei compagni di corso[1] per gli esami di Tecnologie Software per il Web e Ingegneria del Software.
Le domande di ricerca poste sono:

RQ$_1$. *Che grado di aiuto può fornire ChatGPT nelle attività di software engineering?*

La prima domanda di ricerca riguarda quanto aiuto ChatGPT possa fornire nelle tipiche attività di software engineering presenti nelle diverse fasi di raccolta e analisi dei requisiti, system design, object design e testing. Esempi di attività sono la stesura dei requisiti funzionali e non funzionali, scrittura di use cases, generazione di diagrammi, decomposizione in sottosistemi, definizione di interfacce e test cases, etc.

[1] Andrea Vitolo, Carlo Colizzi e Carmine Pio Nardo

RQ$_2$. *Quanto sforzo è richiesto per arrivare ad una soluzione accettabile?*

La seconda domanda di ricerca riguarda quanto impegno sia necessario da parte del prompt engineer[2], in termini di numero di prompts e tempo impiegato, per poter arrivare ad inserire le risposte date da ChatGPT nella documentazione.
Per interagire con il bot sono state prese le seguenti convenzioni:

- È stata usata la versione gratuita del bot, basata su un modello GPT-3.5, meno potente rispetto al modello a pagamento GPT-4.
- Tutti i prompts sono stati immessi nella medesima chat.[3]
- Come primo prompt della chat, è stato chiesto esplicitamente a ChatGPT di impersonare un software engineer.
- L'interazione con il bot è avvenuta in italiano.
- Le eventuali correzioni da apportare alle risposte del bot sono state demandate in primis al bot stesso. In caso di correzioni veloci oppure nel caso in cui il bot ha difficoltà e/o impiega troppo tempo la risposta è stata modificata e inserita manualmente.

[2]Prompt engineer: Persona che si occupa di definire i prompts da immettere al bot
[3]Per visionare la chat avuta con ChatGPT cliccare qui

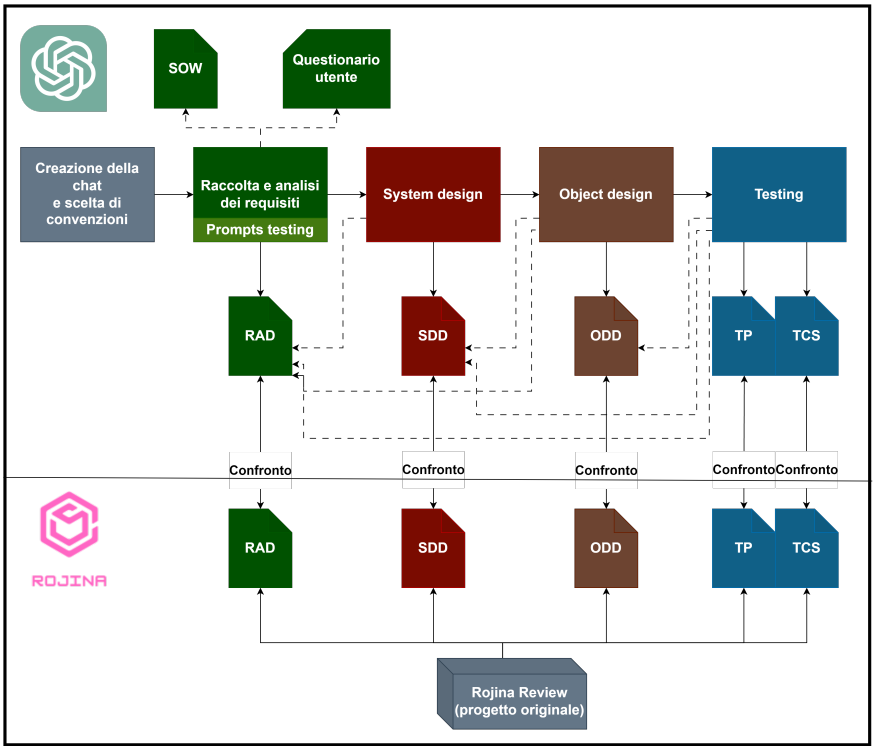

Figura 3.1: Pipeline di lavoro

3.1 Raccolta e analisi dei requisiti

Lo sviluppo è cominciato dando come prompt lo statement of work definito durante il progetto del corso di Ingegneria del Software, rimuovendo le macro funzionalità e altri dettagli non utili al bot. Allo statement sono poi state aggiunte, alla fine, le richieste "definisci scopo, ambito e macro funzionalità del sistema", come evidenziato in figura 3.2. Scopo e ambito servono a dare una descrizione introduttiva del sistema. In seguito gli si è richiesto di derivare i requisiti funzionali, con identificativi e suddivisi per attore, dalle macro funzionalità definite in precedenza e di scrivere i requisiti non funzionali secondo lo schema FURPS. La raccolta dei requisiti è stata poi arricchita richiedendogli di scrivere scenari ed use cases: in particolare si è fatto riferimento a quelli già scritti nel progetto originale di Rojina Review, in modo da poter effettuare un confronto diretto con ChatGPT. In seguito è iniziata l'analisi dei requisiti, chiedendo al bot di individuare gli oggetti entity del sistema. Fatto ciò si è passati alla definizione dei diagrammi, in particolare: class, sequence e statechart diagrams. Sono stati usati due approcci per la loro generazione:

- Esplicitando semplicemente la richiesta al bot, lasciando a lui la decisione su come generare i diagrammi
- Esplicitando, oltre alla richiesta, l'uso di codice PlantUML[4], immesso poi in draw.io[5] per generare i diagrammi.

Anche qui, come per scenari ed use cases, i sequence e statechart diagrams fatti generare riguardano funzionalità ed oggetti di cui sono stati creati diagrammi nel progetto originale per agevolare il confronto.

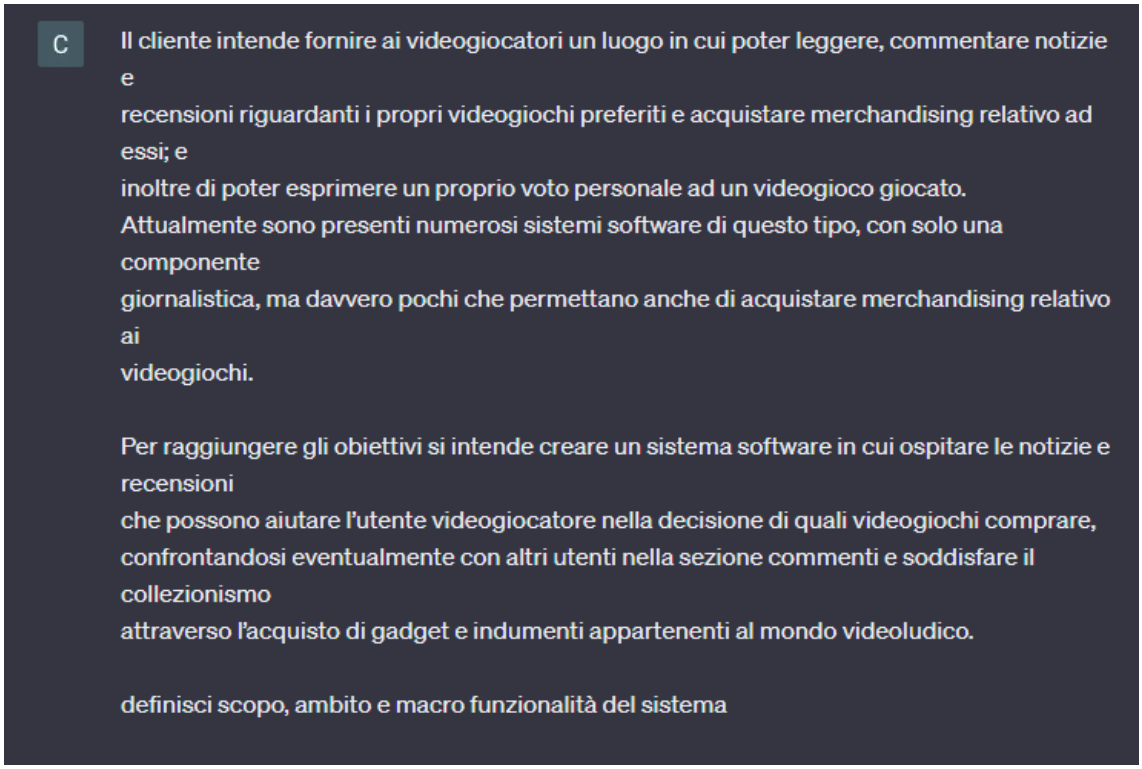

Figura 3.2: Prompt usato per richiedere scopo, ambito e macro funzionalità del sistema

3.1.1 Prompts testing

I prompts, dato che sono definiti da una persona umana, potrebbero soffrire di estrema soggettività: persone diverse scrivono in maniera diversa e di conseguenza anche le risposte ottenute da ChatGPT potrebbero variare molto.

[4]PlantUML: Linguaggio che permette di descrivere diagrammi UML, traducibile poi in immagini attraverso appositi tool

[5]draw.io: Strumento software per la creazione di diagrammi UML, ER e flowcharts

È stato dunque definito un questionario, il cui scopo principale è stato raccogliere prompts da persone che avessero esperienza in ambito di software engineering per poi confrontare le risposte ottenute dal bot in base ai diversi stili di scrittura. Il questionario è stato impostato in modo da simulare lo stesso approccio e metodologia usati in questa tesi: essendo consapevoli di trovarsi nella medesima chat con il bot, senza la necessità di dover ogni volta esplicitare tutte le informazioni e seguendo lo stesso ordine sequenziale di prompts. Le domande presenti sono in totale 11, di cui: 3 chiuse che fanno riferimento alle informazioni della persona che sta compilando il questionario, ovvero conoscenze di software engineering, titolo di studio e posizione lavorativa; 8 aperte che fanno riferimento alla raccolta e analisi dei requisiti: macro funzionalità, requisiti funzionali e non funzionali, scenari ed use cases, class, sequence e statechart diagrams.
Hanno partecipato al sondaggio 12 persone con diversi livelli di conoscenza di ingegneria del software e gradi di istruzione. Per ogni sottomissione è stata creata una chat apposita con il bot, inserendo e confrontando i prompts raccolti.

3.2 System design

Lo sviluppo è proseguito con la fase successiva di design del sistema, facendo generare i design goals a partire dai requisiti non funzionali del sistema. Si è tentato sia di farglieli ricordare sfruttando la sua context window[6], in quanto definiti in precedenza nella medesima chat, che fornendoglieli esplicitamente nel prompt. Nei prompts per generare i design goals è stato inoltre esplicitamente richiesto di formattarli tramite tabella con rank, identificativo, descrizione, categoria e requisiti non funzionali di origine. Per la decomposizione in sottosistemi, dapprima si è richiesto al bot che informazioni debba avere il prompt da sottomettergli per generare una buona suddivisione del sistema. Poi sono stati usati prompts specifici per fargli memorizzare requisiti funzionali, non funzionali e design goals. Infine è stato usato il prompt suggerito dal bot per generare la decomposizione in sottosistemi, come

[6]Context window: Il numero di diversi tokens che il bot può elaborare contemporaneamente. Ciò permette di creare dipendenze e riferimenti con prompts e risposte precedenti, dando nuove risposte più contestualizzate

evidenziato in figura 3.3. Da quest'ultima è stato poi richiesto al bot di generare, tramite PlantUML, il component diagram del sistema. Per quanto riguarda l'architettura del sistema, è stato dato un prompt in cui si richiedeva al bot consigli su quale utilizzare per poi richiedergli, sempre in PlantUML, l'architectural diagram. È stato in seguito richiesto di definire il mapping hardware/software del sistema e di generare il deployment diagram corrispondente. Si è passati quindi alla gestione dei dati persistenti, chiedendogli dapprima quale strumento utilizzare e in seguito facendogli generare il dizionario dei dati del sistema in base agli oggetti entity forniti. Per il dizionario dei dati gli è stato richiesto esplicitamente, in più prompts, di organizzare le informazioni in tabelle, una per ogni entità con descrizione, campi, vincoli. Sono inoltre stati dati prompts al bot per avere la sua opinione su alcune scelte di design intraprese in merito di alcune entità del sistema. Si è proseguiti quindi chiedendo al bot il tipo di controllo globale del software da adottare e di definire le condizioni limite del sistema tramite i boundary use cases. In particolare per quest'ultimi, nel prompt è stato inserito un esempio di boundary use case (preso dal progetto originale) in formato markdown[7], al fine di dare al bot un riferimento per la struttura dello uso case. Il markdown è stato ottenuto tramite il sito Table to Markdown, che converte tabelle (in questo caso la tabella in cui era formattato il boundary use case) in testo markdown da poter dare in pasto a ChatGPT. Sono stati poi definiti i servizi dei sottosistemi, chiedendo al bot, un prompt alla volta, di generare una tabella per ogni sottosistema con nome del servizio e relativa descrizione. Ultima attività di questa fase è stata definire la matrice del controllo degli accessi. È stato dato un prompt iniziale in cui si definivano righe e colonne della matrice, per poi riempirla iterativamente con prompts successivi, in cui ognuno conteneva il markdown dei servizi di uno specifico sottosistema definiti in precedenza.

[7]Markdown: linguaggio di markup che può essere convertito in HTML o altri formati tramite un tool apposito

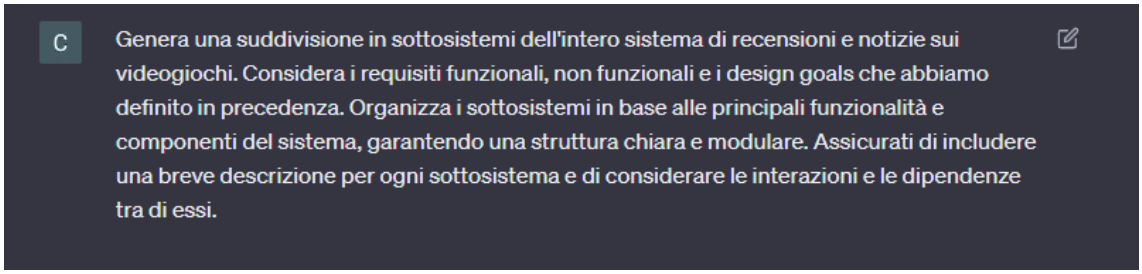

Figura 3.3: Prompt usato per richiedere la decomposizione in sottosistemi

3.3 Object design

La stesura di documentazione si è conclusa con quest'ultima fase. Inizialmente sono stati immessi prompts per generare le sezioni descrittive del documento di object design come object design goals e linee guida per la scrittura di codice. In seguito ci si è focalizzati con la suddivisione in packages del sistema, dandogli come prompt la lista di sottosistemi e le tecnologie da usare per lo sviluppo web, come evidenziato in figura 3.4. Successivamente sono stati dati altri prompts in cui sono stati richiesti i diagrammi dei packages, sia singoli che una vista generale più ad alto livello, sempre tramite PlantUML. Si è poi passati alla definizione delle interfacce di classe, in particolare quelle relative ai sottosistemi: sono stati dati diversi prompts, uno per sottosistema, in cui in base a quest'ultimo e ai suoi servizi è stato richiesto di definire l'interfaccia con metodi, descrizioni, parametri di input e di ritorno, pre e post condizioni sottoforma di tabella. In ultimo sono stati richiesti consigli al bot in merito a quale design pattern applicare nel sistema con relativo diagramma di rappresentazione e una descrizione dei componenti terzi usati dal sistema. Come specificato anche nella sezione precedente, ci si è soffermati sulla generazione di documentazione e non è stato generato codice.

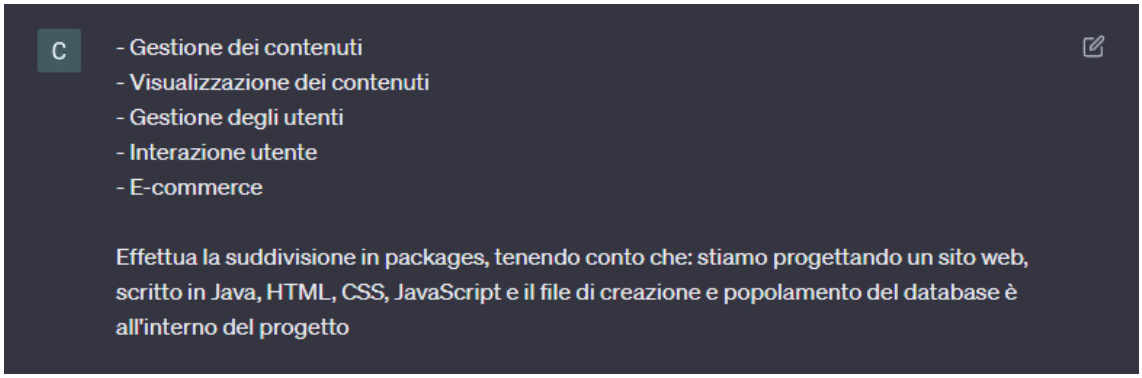

Figura 3.4: Prompt usato per richiedere la suddivisione in packages

3.4 Testing

Seguendo lo stesso ordine di fasi svolte durante il progetto di Ingegneria del Software, prima di focalizzarsi sull'object design si sono pianificate le attività di testing. Sono stati dunque usati diversi prompts per far generare le sezioni descrittive del documento di test plan come pass/fail criteria, approccio, materiale di testing, etc. Le funzionalità da testare sono state invece definite manualmente, facendo riferimento a quelle testate nel progetto originale. Si è poi passati ai category partitions, utilizzando tre diversi approcci per definirlo:

- Un approccio che utilizza uno use case generato dal bot stesso
- Un approccio che utilizza uno use case preso dal progetto originale
- Un approccio che utilizza un requisito funzionale

In ogni caso è stato richiesto al bot di formattare le risposte in tabelle con parametri, categorie e condizioni e di definire le combinazioni/test cases tramite tabelle con test case ID, test frame ed esiti. Occasionalmente nei prompts sono stati inseriti markdown di category partition da cui derivare i test cases. In seguito si è passati alla fase di specificazione dei test cases. Ciò è stato fatto dando diversi prompts, in cui ognuno conteneva il markdown del category partition e relativi test cases definiti in precedenza come evidenziato in figura 3.5, richiedendo al bot di specificare i test cases usando valori di esempio reali e formattandoli tramite tabella con test case ID, test frame, valori di input ed esiti. Visto che ci si è soffermati sulla generazione di documentazione e non di codice, i test pianificati e specificati non sono stati eseguiti.

C

Parametro	Categoria	Condizione	Esito
Nickname	Lunghezza [LN]	Lunghezza <= 20 AND Lunghezza >= 4	LN_OK
		Lunghezza > 20 OR Lunghezza < 4	Errore
	Caratteri speciali [CSN]	Caratteri speciali non presenti	CSN_OK
		Caratteri speciali presenti	Errore
Email	Formato [FMT]	Formato email valido (es. example@example.com)	FMT_OK
		Formato email non valido (es. example.com)	Errore
Password	Lunghezza [LP]	Lunghezza <= 20 AND Lunghezza >=8	LP_OK
		Lunghezza > 20 OR Lunghezza < 8	Errore
	Caratteri speciali [CSP]	Caratteri speciali presenti	CSP_OK
		Caratteri speciali non presenti	Errore

Test Case ID	Test Frame	Esito
TC_1.1_1	LN_ERR, CSN_OK, FMT_OK, LP_OK, CSP_OK	Errore: Lunghezza nickname non valida
TC_1.1_2	LN_OK, CSN_OK, FMT_ERR, LP_OK, CSP_OK	Errore: Formato email non valido
TC_1.1_3	LN_OK, CSN_OK, FMT_OK, LP_ERR, CSP_OK	Errore: Lunghezza password non valida
TC_1.1_4	LN_OK, CSN_OK, FMT_OK, LP_OK, CSP_ERR	Errore: Caratteri speciali mancanti nella password
TC_1.1_5	LN_OK, CSN_ERR, FMT_OK, LP_OK, CSP_OK	Errore: Caratteri speciali non validi nel nickname
TC_1.1_6	LN_OK, CSN_OK, FMT_OK, LP_OK, CSP_OK	Corretto: Tutti i parametri sono validi

Questi sono category partition e relativi test cases di Registrazione Videogiocatore

Figura 3.5: Prompt usato per richiedere la specifica dei test cases di un category partition espresso tramite markdown

CAPITOLO 4

Risultati

In questo capitolo vengono riportati e discussi i risultati ottenuti tramite i prompts immessi a ChatGPT. Verrà in primis discusso l'esito del questionario tramite il testing dei prompts raccolti e in seguito, con le informazioni ottenute interagendo con il bot, si risponderà alle domande di ricerca poste nella metodologia. [1]

4.1 Prompts testing

Al questionario hanno partecipato 12 persone che studiano in ambito informatico, suddivise per grado di istruzione in questo modo: 8 studenti triennali, 2 laureati triennali e 2 dottorandi. Per quanto riguarda la posizione lavorativa invece: 6 non lavorano, 3 lavorano in ambito aziendale, 2 in ambito accademico e 1 in ambito startup. In generale non si è riscontrata una correlazione tra lo stile di scrittura e il grado di istruzione-posizione lavorativa: ciò può essere dovuto alla grandezza del campione in esame, troppo piccolo per trovare correlazione e/o all'impostazione del questionario, con domande sì aperte ma altamente guidate. Di seguito una sintesi dei risultati per ogni domanda presente nel questionario:

[1] Per visionare i documenti prodotti con l'ausilio di ChatGPT, cliccare qui

- **Macro funzionalità**: in generale non ci sono state differenze tangibili nella generazione delle macro funzionalità del sistema anche usando prompt simili (stesso livello di dettaglio) ma formulazioni diverse. Unica accortezza è nell'usare verbi come "estrai", "deriva", che in alcuni casi hanno portato il bot a fare un semplice parsing dello statement of work dato in input, non aggiungendo di fatto nuove macro funzionalità. Sono stati inoltre riscontrati alcuni requisiti non funzionali nell'elenco di macro funzionalità. In aggiunta è stato testato un prompt minimale "get functions" che ha restituito un elenco soddisfacente di macro funzionalità.

- **Requisiti funzionali**: si è verificato che il risultato migliore si ottiene dicendo esplicitamente al bot di derivare i requisiti funzionali dalle macro funzionalità o di specificare "a basso livello": facendo altrimenti, il bot riscrive le stesse macro funzionalità usando parole diverse ma senza effettivamente andare nel dettaglio, come dovrebbe invece fare un requisito funzionale. Il bot è inoltre capace di assegnare una buona priorità ai requisiti funzionali. Saltuariamente nell'elenco dei requisiti funzionali sono presenti alcuni requisiti non funzionali.

- **Requisiti non funzionali**: in generale qualsiasi prompt restituisce una lista soddisfacente di requisiti non funzionali. I risultati migliori sono stati però ottenuti dicendo esplicitamente al bot di suddividerli secondo lo schema FURPS o in generale invitandolo a scendere nel dettaglio. È stato inoltre notato che, se i requisiti funzionali sono stati scritti derivandoli direttamente dalle macro funzionalità (e quindi scritti più in dettaglio), anche i requisiti non funzionali seguono lo stesso stile di scrittura più preciso.

- **Scenario**: in generale non ci sono state differenze tangibili nella generazione degli scenari anche usando prompt simili (stesso livello di dettaglio) ma formulazioni diverse. A seconda delle esigenze, è possibile far generare al bot uno scenario descrittivo oppure uno scenario in cui vengono sottolineate in maniera marcata le interazioni tra utente e sistema.

- **Use case**: in generale non ci sono state differenze tangibili nella generazione degli use cases anche usando prompt simili (stesso livello di dettaglio) ma

formulazioni diverse. È stato però capace di creare un buono schema grafico di uno use case, delineando stati e transizioni appropriate. A volte negli use cases menziona requisiti funzionali che non sono stati proprio definiti all'interno della chat.

- **Class diagram**: non c'è stato verso, con nessun prompt, di generare un class diagram soddisfacente, corredato di gerarchie, relazioni e cardinalità giuste, anche dandogli in input una descrizione testuale di quest'ultime. Bisogna inoltre stare attenti a specificare che il class diagram debba essere "del sistema": dato che il prompt del class diagram è stato immesso subito dopo la generazione di uno use case, se non specificato il contesto, il bot potrebbe generare le classi relative esclusivamente allo use case. Le descrizioni testuali del diagramma che sono state ottenute sono però soddisfacenti.

- **Sequence diagram**: Un sequence diagram soddisfacente è ottenibile solamente tramite l'ausilio di PlantUML: il bot non è in nessun modo capace di disegnare in maniera accurata sfruttando solo i caratteri ASCII

- **Statechart diagram**: La situazione, rispetto ai sequence diagram, è leggermente migliore per quanto riguarda il disegno tramite caratteri ASCII: gli stati sono delineati in maniera corretta ma mancano le transizioni. Anche in questo caso risultati soddisfacenti sono ottenuti solo tramite l'ausilio di PlantUML.

In conclusione non ci sono differenze tra i risultati ottenuti con prompt con diversi stili di scrittura, lasciando all'utente la possibilità di scegliere se parlare al bot in maniera più "gentile" trattandolo come un essere umano, oppure fargli richieste esplicite e dirette trattandolo come una macchina. Differenze sono state invece riscontrate a seconda di quanto chiediamo al bot di essere dettagliato nella risposta: ciò ovviamente dipende da cosa vogliamo ottenere, in quanto a volte potrebbero essere preferibili risposte più sintetiche e concise piuttosto che dettagliate e discorsive. Per quanto concerne i diagrammi invece, si è assodato che data la sua natura di chatbot testuale ha davvero molta difficoltà a disegnare usando caratteri, fornendo però delle descrizioni testuali più che soddisfacenti dei diagrammi che prova a creare. [2]

[2] Accesso al questionario qui

4.2 RQ1: Che grado di aiuto può fornire ChatGPT nelle attività di software engineering?

Nelle sezioni successive verrà valutato l'aiuto e il supporto fornito da ChatGPT in ciascuna fase dello sviluppo software.

4.2.1 Raccolta e analisi dei requisiti

Non c'è stata una sostanziale differenza tra il dare direttamente come prompt iniziale della chat lo statement of work e invece di dirgli esplicitamente di impersonare di un software engineer prima di fargli generare gli artefatti: dato un quesito o una richiesta, il bot risponderà sempre al massimo delle sue capacità; nel caso specifico non è stata rilevata una differenza nello stile di scrittura. I cambiamenti, laddove sono presenti, sono frutto del non determinismo delle risposte del bot. A supporto di quanto detto, in un'altra chat, si è tentato di far impersonare a ChatGPT un software engineer non competente e ha risposto comunque al massimo delle sue potenzialità, impersonando di fatto, in maniera implicita, un software engineer competente. Solo dopo avergli fatto notare l'errore di incoerenza e spronato a scrivere incorrettamente, il bot ha effettivamente impersonato un software engineer non competente.
Nel generare i requisiti funzionali sono emerse le capacità del bot di soddisfare più richieste contemporaneamente: rimuovere requisiti, aggiungere e rinominare attori. I requisiti non funzionali ottenuti sono invece perlopiù generici ma è stato in grado di suddividerli secondo lo schema FURPS. Nella generazione degli scenari è venuta meno la sua abilità di dedurre informazioni dal contesto, confondendo le funzionalità utilizzabili dal videogiocatore e dal giornalista, aggiungendo inoltre un attore non richiesto. Grazie agli scenari generati in precedenza è riuscito a produrre uno use case model del sistema quasi completo, manchevole solo di alcuni use cases. La scrittura degli use cases è fatta in maniera corretta se esplicitata la funzionalità: usando nel prompt l'identificativo, definito da lui stesso in precedenza nello use case model, lo use case ottenuto è diverso da quello richiesto.
Grazie agli use cases generati in precedenza è stato capace di stilare una lista degli oggetti entity del sistema con relativa descrizione, dimostrandosi anche capace, con

ulteriori prompts, di aggiungere, eliminare e accorpare concetti. Non è stato però capace di rappresentare questi oggetti tramite un class diagram soddisfacente: la maggior parte delle relazioni è sbagliata e alcune sono mancanti. Risultati migliori, ma comunque non accettabili, sono stati ottenuti richiedendo al bot di usare PlantUML e di non disegnare tramite testo ASCII. I sequence e statechart diagrams ottenuti figure 4.1, 4.2, invece, sono soddisfacenti, permettendo al bot di sostituirsi effettivamente alla loro costruzione manuale. Evidenziata in figura 4.4 la rappresentazione ASCII di uno statechart diagram e in figura 4.3 lo statechart diagram preso dal progetto progetto originale.

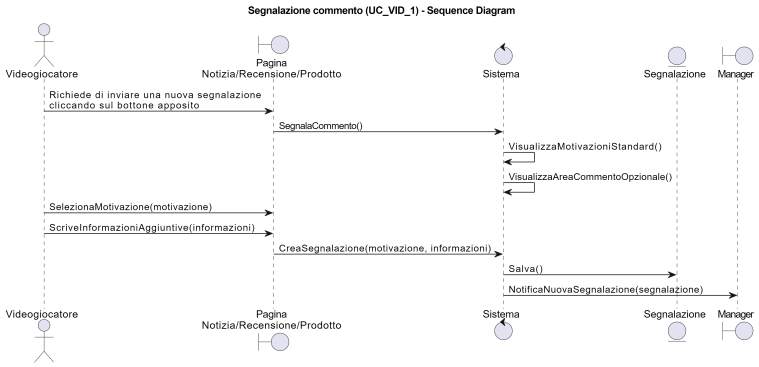

Figura 4.1: Sequence diagram dell'invio di una segnalazione generato da ChatGPT tramite PlantUML

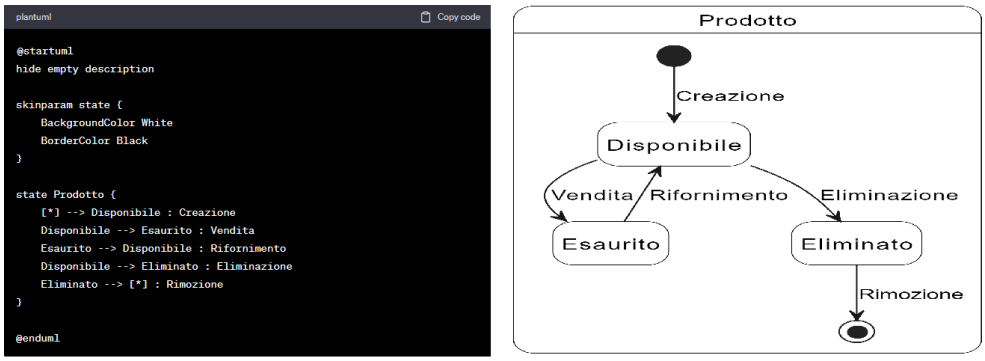

Figura 4.2: Statechart diagram di un oggetto prodotto generato da ChatGPT tramite l'ausilio di PlantUML

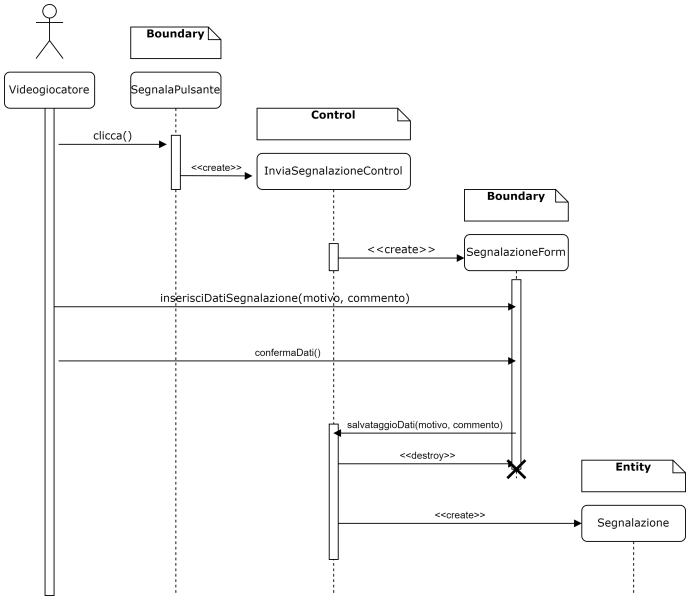

Figura 4.3: Sequence diagram dell'invio di una segnalazione del progetto originale

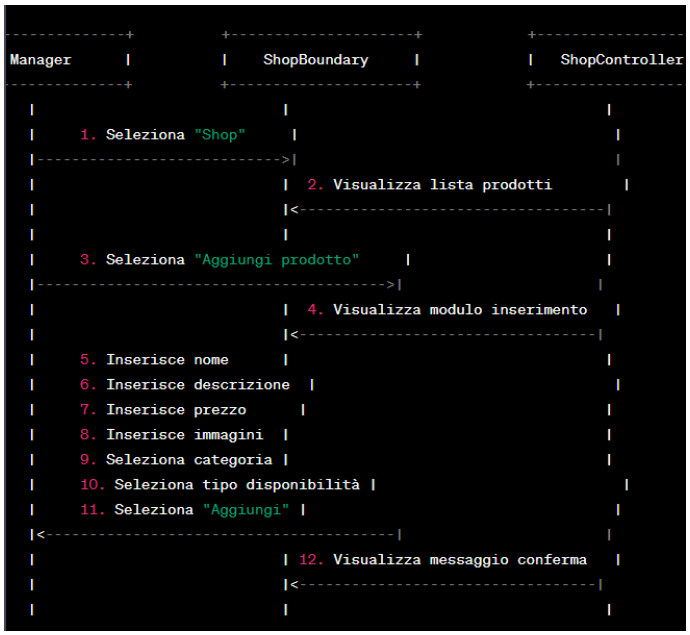

Figura 4.4: Sequence diagram dell'aggiunta di un prodotto generato da ChatGPT tramite ASCII

Take-away Message #1. ChatGPT si è rivelato molto utile nella raccolta dei requisiti, agevolando soprattutto la scrittura di use cases e scenari e fornendo un'ottima base di partenza per requisiti funzionali e non funzionali. Nell'analisi dei requisiti invece non è stato impeccabile a causa dell'impossibilità di generare un class diagram soddisfacente ma è stato comunque di aiuto per i sequence e statechart diagrams.

4.2.2 System design

Per far generare i design goals è stato necessario riscrivere nella chat i requisiti non funzionali prodotti precedentemente, in quanto ci sono stati molti prompts e risposte diverse di mezzo (scenari, use cases, diagrammi) che hanno reso difficile al bot ricordare gli ultimi requisiti non funzionali definiti. I design goals definiti sono stati formattati correttamente sotto forma di tabella e associati ai relativi requisiti non funzionali di origine. Data la complessità e l'eterogeneità di informazioni necessarie per effettuare una suddivisione dei sottosistemi efficiente, si è chiesto direttamente a ChatGPT che prompt sottomettergli per generare la suddetta suddivisione.[3] Con il prompt suggerito dal bot la risposta è stata buona ma non impeccabile: ha fatto un buon lavoro nella suddivisione dei sottosistemi, discreto nell'associazione tra requisiti e design goals con i sottosistemi ma ha totalmente omesso le dipendenze tra sottosistemi, sebbene specificate nel prompt. Sono state quindi necessarie correzioni e prompts extra. Il component diagram associato alla suddivisione in sottosistemi è stato generato in maniera corretta tramite PlantUML 4.5: tutti i sottosistemi sono presenti e le dipendenze rispettate. Unico problema è l'ordine grafico del diagramma con frecce troppo lunghe e/o che si intersecano, ma ciò è imputabile a PlantUML e non a ChatGPT. PlantUML inoltre, attraverso il codice, genera un'immagine che non è possibile modificare facilmente tramite strumenti come draw.io. La descrizione del mapping hardware/software del sistema è stata molto esauriente e dettagliata e il relativo deployment diagram è corretto. Tra i vari possibili modi di gestire i dati persistenti, il bot ha scelto quello corretto: i database relazionali. Il dizionario dei dati generato è a grandi linee corretto, fornendo una buona base su cui poi effettuare modifiche manuali: modifiche effettuate tramite prompts hanno portato all'introduzione di errori e cambi di formattazione delle tabelle. Evidenziata in figura 4.6 la tabella prodotto del progetto originale e in figura 4.7 la tabella prodotto generata da ChatGPT. Per quanto riguarda il controllo degli accessi, il bot ha con successo definito una matrice degli accessi vuota, con i sottosistemi rappresentati dalle righe e gli attori dalle colonne. La matrice è poi stata riempita quasi del tutto con prompts

[3] Quest'approccio può essere utile quando si ha difficoltà a formulare richieste al bot in maniera appropriata.

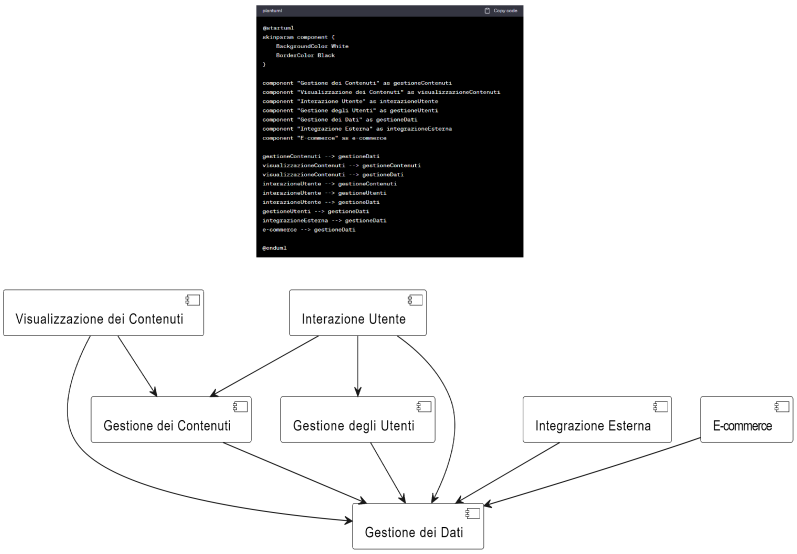

Figura 4.5: Component diagram del sistema generato da ChatGPT tramite PlantUML

Nome entità	*Prodotto*		
Descrizione	Rappresenta un prodotto acquistabile sulla piattaforma		
Nome campo	**Tipo**	**Vincolo di chiave**	**Altri vincoli**
id	int	PRIMARY KEY	NOT NULL AUTO_INCREMENT
nome	varchar(100)		NOT NULL UNIQUE
descrizione	text		NOT NULL
*immagine**	varchar(100)		NOT NULL
prezzo	float		NOT NULL
disponibilità	int		NOT NULL
*numeroVoti***	int		NOT NULL
*mediaVoto***	float		NOT NULL
nome_categoria	varchar(30)	FOREIGN KEY (Categoria)	NOT NULL

Figura 4.6: Tabella prodotto del progetto originale

successivi: sono state necessarie delle modifiche manuali ma la matrice ottenuta costituisce un'ottima base. Sono inoltre apparsi all'interno delle celle della tabella dei tag "
" che sono però stati prontamente rimossi attraverso un prompt. Il flusso di controllo è stato scelto e specificato dal bot in maniera corretta: il tipo event-driven è proprio l'opzione giusta in questo contesto. I boundary use cases generati rispettano la formattazione della tabella di esempio data in input e i contenuti presenti

Tabella: Prodotto

Campo	Tipo	Descrizione	Vincoli
ID	int	Identificatore univoco del prodotto	Chiave primaria
Nome	stringa	Nome del prodotto	-
Descrizione	stringa	Descrizione del prodotto	-
Prezzo	decimal	Prezzo del prodotto	-
Immagini	stringa	Elenco delle immagini del prodotto	-
ID_Categoria	int	Identificatore della categoria associata al prodotto	Chiave esterna (Categoria.ID)
Disponibilità	stringa	Tipo di disponibilità del prodotto (esaurito, disponibile, etc.)	-

Figura 4.7: Tabella prodotto generata da ChatGPT

sono coerenti con quanto richiesto, seppur generici: con un ulteriore prompt è stato specificata la presenza del web server Tomcat, risultando in boundary use cases più dettagliati. I servizi dei sottosistemi generati sono ottimi: il bot, per ogni sottosistema, ha correttamente individuato i servizi presenti nella descrizione e dedotto altri coerenti invece non specificati. ChatGPT si è inoltre dimostrato capace di soddisfare più richieste di aggiunta e modifica di servizi contemporaneamente. Evidenziati in figura 4.8 i servizi del sottosistema "Gestione dei contenuti".

Nome Servizio	Descrizione
Crea Notizia	Permette ai giornalisti di creare una nuova notizia, fornendo i dettagli come titolo, contenuto e categoria.
Modifica Notizia	Permette ai giornalisti di apportare modifiche a una notizia esistente, consentendo di aggiornare il suo contenuto.
Elimina Notizia	Permette ai giornalisti di eliminare una notizia dal sistema.
Crea Recensione	Permette ai giornalisti di creare una nuova recensione di un videogioco, inserendo informazioni dettagliate.
Modifica Recensione	Permette ai giornalisti di modificare una recensione esistente, consentendo di aggiornarne il contenuto.
Elimina Recensione	Permette ai giornalisti di eliminare una recensione dal sistema.
Inserisci Videogioco	Permette ai giornalisti di inserire un nuovo videogioco nel sistema, fornendo i dettagli come titolo, descrizione e genere.
Modifica Videogioco	Permette ai giornalisti di apportare modifiche a un videogioco esistente, consentendo di aggiornarne le informazioni.
Elimina Videogioco	Permette ai giornalisti di eliminare un videogioco dal sistema.

Figura 4.8: Servizi del sottosistema "Gestione dei contenuti" generati da ChatGPT

💬 **Take-away Message #2.** ChatGPT si è rivelato estremamente utile in questa fase di system design, anche più che nella raccolta e analisi dei requisiti. Il supporto è stato evidente nella creazione del dizionario dei dati, che per sistemi grandi come Rojina Review può impiegare molto tempo senza l'aiuto del bot; discorso simile per quanto riguarda i servizi dei sottosistemi. Le uniche difficoltà riscontrate sono state nella decomposizione in sottosistemi e nella definizione della matrice degli accessi, che hanno necessitato di molti prompts per ottenere delle buone basi di partenza a cui sono state effettuate modifiche manuali.

4.2.3 Object design

Data in input la decomposizione in sottosistemi, la suddivisione in packages è stata effettuata in maniera corretta dal bot con una descrizione adeguata per ogni pacchetto. Ulteriori packages sono stati aggiunti sotto richiesta con successo. Il primo codice PlantUML, fornito per generare il diagramma dei packages, è corretto semanticamente ma non sintatticamente: sono presenti delle stringhe che non permettono la compilazione. In particolare la presenza di "//", che è una stringa consentita ad esempio in Java per scrivere commenti, ma non in PlantUML che usa " ' ". Con l'ausilio di due ulteriori prompts l'errore è stato corretto. Inoltre, grazie alla context window del bot, ogni diagramma è ottenuto con più facilità, ovvero meno prompts, rispetto al diagramma precedente: in un certo senso, ChatGPT "impara" cosa voglio ottenere da lui e anche in quale specifico formato. Evidenziato in figura 4.9 il diagramma dei packages del progetto originale e in figura 4.10 il diagramma generato da ChatGPT. La definizione delle interfacce dei sottosistemi in linguaggio Java è fatta

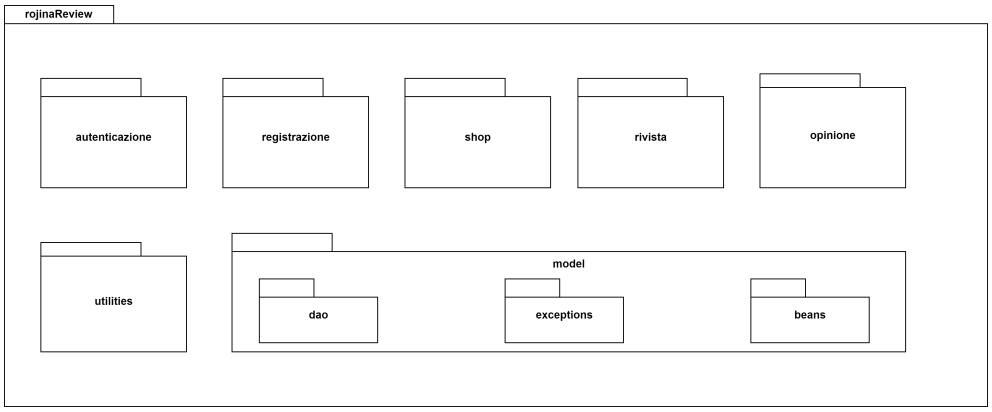

Figura 4.9: Diagramma dei packages del sistema del progetto originale

Figura 4.10: Diagramma dei packages del sistema generato da ChatGPT tramite PlantUML

in maniera ottima da parte del bot. Seppur con difficoltà, ChatGPT ha correttamente convertito le interfacce in linguaggio Java in interfacce più generiche sotto forma di tabelle con descrizione, parametri di input e ritorno, pre e post condizioni per

ogni metodo. Anche qui, come per la matrice degli accessi, sono stati inseriti dei tag "
" visibili non richiesti che stavolta è stato necessario rimuovere a mano. Per quanto riguarda i design pattern, in risposta alla richiesta su quale utilizzare, ChatGPT ha consigliato l'MVC. Ciò non è completamente corretto in quanto l'MVC non è un design pattern secondo la famosa gang of four[20] ma bensì un'architettura software three-tier che utilizza al suo interno il design pattern Observer. È stato in seguito proposto dal bot come design pattern Front Controller, adducendo però motivazioni non abbastanza convincenti ad adottarlo. È stato quindi proposto nel prompt il design pattern Facade, che ha generato una corretta spiegazione di quest'ultimo e calandolo nel contesto di Rojina Review. Inoltre è stato generato un diagramma corretto, tramite PlantUML, dell'applicazione del design pattern nel sistema 4.11.

💬 **Take-away Message #3.** ChatGPT si è rivelato molto utile nella fase di object design. Il bot ha dato grande aiuto e conseguente risparmio di tempo sulla definizione delle interfacce, sia in formato tabellare, riportate nel documento di object design, che in linguaggio Java per un futuro sviluppo di codice. Inoltre è stato di supporto nello spiegare dettagliatamente la teoria e l'applicazione del design pattern scelto, dimostrandosi capace anche di generare un diagramma corrispondente di esempio. Nulla di speciale per quanto riguarda i packages invece, la cui suddivisione e creazione di diagrammi avrebbe richiesto più o meno lo stesso tempo anche fatte manualmente senza ChatGPT.

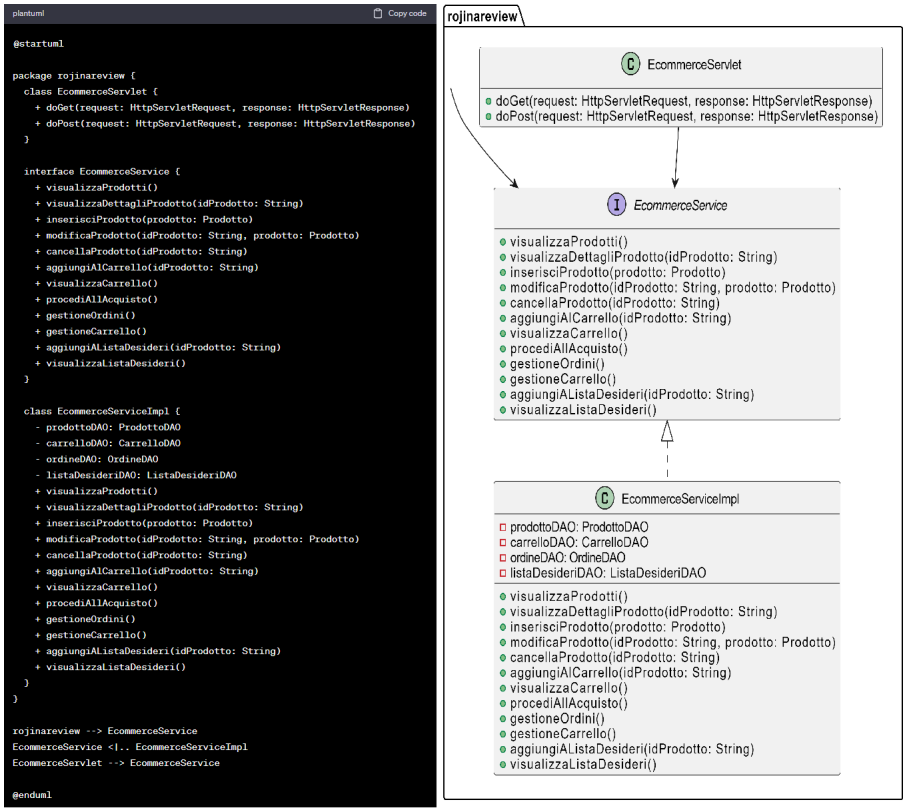

Figura 4.11: Diagramma di applicazione del design pattern Facade all'interno del sistema generato da ChatGPT tramite PlantUML

4.2.4 Testing

I primi category partition ottenuti sono insoddisfacenti: è stato necessario riportare tramite markdown un esempio di category partition corretto preso dal progetto originale per generarne una buona base di partenza: sono necessari ulteriori prompts e/o modifiche manuali per migliorare categorie, scelte e combinazioni/test cases. Anche senza use case, ma specificando solo la funzionalità, è capace di dedurre parametri e categorie. Evidenziato in figura 4.12 un category partition preso dal progetto originale e in figura 4.13 un category partition generato da ChatGPT. I test cases specifications, generati a partire dai markdown dei category partitions e relativi test cases, sono dopo diversi tentativi perlopiù corretti. Potrebbero esserci piccoli errori come non rispettare una scelta di un test frame: ad esempio, nel contesto della registrazione di un videogiocatore, per la scelta "Lunghezza nickname sbagliata" ha inserito un nickname di lunghezza valida. I test cases specifications seguenti, grazie alla context window, hanno necessitato di meno correzioni manua-

Parametro: Motivazione	
Nome Categoria	Scelta per la categoria
Scelta [SM]	1. Scelta == "Scegli motivazione" [error] 2. Scelta == "Altro" [PROPERTY SM_OK] [PROPERTY ALTRO] 3. Scelta == altri elementi in lista [PROPERTY SM_OK]

Parametro: Commento_Aggiuntivo	
Nome Categoria	Scelta per la categoria
Lunghezza Commento [LC]	1. IF ALTRO=true AND LC=0 [error] 2. IF ALTRO=false AND LC=0 [PROPERTY LC_OK] 3. LC>=1 AND LC<=256 [PROPERTY LC_OK] 4. LC > 256 [Errore]

Test Case ID	Test Frame	Esito
TC_ 2.2_1	SM1	Errato: Scegliere una motivazione
TC_ 2.2_2	SM2,LC1	Errato: Inserire un commento aggiuntivo per specificare meglio la segnalazione
TC_2.2_3	SM2,LC4	Errato: Commento troppo lungo
TC_2.2_4	SM2,LC3	Corretto
TC_2.2_5	SM3,LC2	Corretto
TC_2.2_6	SM3,LC3	Corretto

Figura 4.12: Category partition di "Segnalazione commento" del progetto originale

Parametro	Categoria	Condizione	Esito
Motivazione	Selezione [SEL]	Motivazione selezionata tra le opzioni standard	SEL_OK (Motivazione selezionata)
		Motivazione non selezionata	SEL_NA_ERR (Motivazione non selezionata)
		Motivazione selezionata "Altro" senza commento aggiuntivo	SEL_OT_ERR (Motivazione Altro senza commento)
Commento	Lunghezza [L]	Lunghezza <= 256 caratteri	L_OK (Lunghezza commento <= 256)
		Lunghezza > 256 caratteri	L_NA_ERR (Lunghezza commento > 256)

Test Case ID	Test Frame	Esito
TC_1	SEL_OK, L_OK, C_OK	OK (Motivazione selezionata, lunghezza commento <= 256, commento aggiuntivo vuoto)
TC_2	SEL_OT_OK, L_OK, C_OK	OK (Motivazione selezionata, lunghezza commento <= 256, commento aggiuntivo non vuoto)
TC_3	SEL_ERR, L_OK, C_OK	Errore (Motivazione non selezionata, lunghezza commento <= 256, commento aggiuntivo vuoto)
TC_4	SEL_OT_OK, L_ERR, C_OK	Errore (Motivazione selezionata, lunghezza commento > 256, commento aggiuntivo non vuoto)

Figura 4.13: Category partition di "Segnalazione commento" generato da ChatGPT

li. Evidenziato in figura 4.14 un singolo test case specification preso dal progetto originale e in figura 4.15 tutti i test cases specification formattati in una singola tabella. Entrambe le figure si riferiscono allo stessa funzionalità "Segnalazione commento".

Test case ID	TC_2.2_1	Test frame	SM1

Pre-condizione

Il videogiocatore si trova nella pagina di segnalazione

Flusso di eventi

1.Il videogiocatore nella segnalazione inserisce i seguenti dati in input

Input	Valore
Motivazione	Seleziona una motivazione
Commento_Aggiuntivo	

2.Il videogiocatore clicca sul pulsante Segnala

Oracolo

La segnalazione non viene inoltrata perché non è stata selezionata una motivazione valida

Figura 4.14: Test case specification di "Segnalazione commento" del progetto originale

Test Case ID	Test Frame	Valori di Input	Esito
TC_2.2_1	SEL_NA_ERR	Motivazione: Nessuna	Errore: la motivazione non è stata selezionata
TC_2.2_2	SEL_OT_ERR	Motivazione: Altro Commento: Nessun commento aggiuntivo	Errore: è stato selezionato "Altro" senza commento aggiuntivo
TC_2.2_3	SEL_OK, L_ERR	Motivazione: Bug Commento: "Lorem ipsum dolor sit amet, consectetur adipiscing elit"	Errore: il commento aggiuntivo è troppo lungo
TC_2.2_4	SEL_OK, L_NE_OK	Motivazione: Feedback Commento: "Il livello di difficoltà è troppo elevato"	Corretto: è stata selezionata una motivazione e un commento aggiuntivo
TC_2.2_5	SEL_OK, L_E_OK	Motivazione: Suggerimento Commento: Nessun commento aggiuntivo	Corretto: è stata selezionata una motivazione (che non è "Altro") e nessun commento aggiuntivo

Figura 4.15: Test cases specification di "Segnalazione commento" generato da ChatGPT

💬 **Take-away Message #4.** ChatGPT si è rivelato sufficiente nel test planning, ovvero nella definizione di category partitions e test cases. Svolge un buon lavoro nel passaggio da use case/funzionalità a category partition, necessitando solo di qualche correzione manuale. Pecca invece nella definizione di combinazioni, che potrebbero essere scorrette o mancanti, rendendo l'utilizzo di tool appositi per definire test cases più adatto. È stato invece quasi ottimo nella specifica dei test cases, facendo risparmiare molto tempo nella creazione di dati reali di esempio. Occasionalmente potrebbe sbagliare qualche parametro e va quindi corretto.

4.3 RQ2: Quanto sforzo è richiesto per arrivare ad una soluzione accettabile?

Complessivamente lo sforzo richiesto per arrivare ad una soluzione accettabile è medio: può variare da basso ad alto a seconda della complessità dell'attività in questione, in maniera direttamente proporzionale al numero di prompts e correzioni necessarie. Ad esempio una decomposizione in sottosistemi con relativi servizi formattati tramite tabella nella fase di system design richiede più sforzo rispetto alla stesura degli scenari nella raccolta dei requisiti. Da sottolineare che l'esperienza acquisita interagendo con il bot fa una grande differenza nello sforzo necessario per arrivare ad una soluzione accettabile: capire e settare in modo appropriato la sua context window diventa fondamentale così come comprendere le sue limitazioni nell'affrontare task troppo difficili che necessitano di un approccio divide et impera. Ritengo di particolare importanza la figura del prompt engineer che verrà a crearsi nei prossimi anni, capace, attraverso prompts studiati e mirati, di ottenere il massimo da ChatGPT e indirizzarlo nella direzione voluta.
Nelle sezioni successive verrà quantificato e discusso nel dettaglio l'impegno richiesto da parte del prompt engineer per ottenere da ChatGPT una soluzione accettabile.
Sono stati definiti tre livelli di effort:

- **basso**, la risposta desiderata viene ottenuta con un massimo di 2 prompts, quest'ultimi non necessitano di essere articolati e la risposta non necessita di correzioni ma può essere tranquillamente copiata e incollata. Il prompt engineer può anche non avere conoscenze di software engineering. Si ricorda però che il bot può occasionalmente sbagliare ed è sempre utile, se non anche necessario, avere un esperto del campo che supervisiona il suo operato.

- **medio**, la risposta desiderata viene ottenuta con un numero di prompts compreso tra 3 e 5, quest'ultimi sono più o meno articolati e la risposta potrebbe necessitare qualche correzione manuale laddove sia più complicato far modificare la risposta al bot stesso. Il prompt engineer deve avere conoscenze di base di software engineering.

- **alto**, la risposta desiderata viene ottenuta con un minimo di 5 prompts, quest'ultimi necessitano di essere molto articolati e la risposta necessita di correzioni manuali che il bot non saprebbe attuare. Il prompt engineer deve avere ottime conoscenze di software engineering.

4.3.1 Raccolta e analisi dei requisiti

- **Macro funzionalità**: basso, la prima risposta del bot è stata più che sufficiente.
- **Requisiti funzionali**: basso, le poche correzioni impartite al bot sono dovute alla componente soggettiva con la quale un software engineer (in questo caso ChatGPT) può interpretare uno statement of work: non avendogli dato in input le macro funzionalità presenti nel SOW, i requisiti funzionali generati si discostavano dal progetto originale.
- **Requisiti non funzionali**: basso, sebbene non sia stato capace di generare requisiti non funzionali specifici, quelli ottenuti non hanno richiesto molto impegno da parte del prompt engineer.
- **Scenari**: medio, per ottenere uno scenario in linea con il contesto dei requisiti funzionali generati, c'è stato il bisogno di essere più precisi con i prompt.
- **Use case model**: medio, ci sono voluti diversi prompt per ottenere una lista completa di use cases.
- **Use cases**: medio, non è possibile usare gli identificativi definiti precedentemente nella chat e bisogna essere specifici in modo da ottenere anche i flussi di evento alternativi.
- **Oggetti entity**: medio, più prompts sono stati necessari per ottenere una lista completa degli oggetti entity del sistema.
- **Sequence diagram**: alto, per ottenere un sequence diagram soddisfacente (ma non perfetto) vi è bisogno di molti prompt e di essere molto specifici.
- **Statechart diagram**: medio, con un numero non eccessivo di prompts si sono ottenuti risultati soddisfacenti.

Il class diagram non è stato preso in considerazione in quanto non ottenibile in maniera soddisfacente tramite ChatGPT.

4.3.2 System design

- **Design goals**: medio, per ottenere una lista soddisfacente di design goals ordinati secondo priorità, con RNF di origine ed identificativi, eventualmente anche in formato tabellare, sono necessari prompts relativamente articolati.
- **Suddivisione in sottosistemi**: alto, sono stati necessari più prompts per ottenere la suddivisione finale, tra memorizzazioni di RF, RNF e DG, richiesta effettiva e correzioni impartite al bot (come le dipendenze) e correzioni effettuate manualmente.
- **Component diagram**: medio-basso, è necessario specificare i sottosistemi e le loro dipendenze, per poi richiedere al bot il testo PlantUML da convertire in immagine tramite strumenti come draw.io.
- **Hardware/software mapping**: medio-basso, è bastato un solo prompt, seppur più o meno articolato, per generare una risposta dettagliata e soddisfacente a cui non è stata necessaria alcuna manipolazione.
- **Deployment diagram**: medio-basso, grazie alla descrizione dettagliata fornita precedentemente dal bot sul mapping hardware/software, è stato immediato ottenere il deployment diagram corrispondente. Unico problema si è presentato con l'inclusione di url non più esistenti nel codice PlantUML, conseguenza dei dati non aggiornati su cui è addestrato il bot. Questa difficoltà potrebbe presentarsi per qualsiasi tipo di diagramma.
- **Dizionario dei dati**: alto, sono stati necessari molti prompts (8) per acquisire una buona base di partenza che comunque ha necessitato molte modifiche che sarebbero state difficili da ottenere dal bot: in ottica di risparmio tempo per un software engineer si è quindi optato per effettuare manipolazioni manuali. Esempi sono aggiungere la clausola "NOT NULL" ad alcuni attributi; aggiungere attributi mancati dal bot (ad es. in Videogiocatore il suo nickname);

modificare alcuni attributi (ad es. in Prodotto modificare la disponibilità da tipo stringa a tipo intero).

- **Controllo degli accessi**: alto, sono stati necessari numerosi prompts (circa 15) per costruire l'intera matrice. Si è rivelato molto utile nella costruzione delle prime 3 righe/sottosistemi della tabella con poche manipolazioni manuali. Le restanti 2 righe/sottosistemi sono state definite invece quasi del tutto manualmente.

- **Controllo globale del software**: basso, è stato necessario un singolo prompt, seppur articolato, per ottenere una risposta giusta e soddisfacente che non ha avuto bisogno di modifiche manuali.

- **Condizioni limite**: medio-basso, è stato necessario in media 1 prompt per ogni boundary use case, fatto eccezione per il primo che ne ha necessitati 2: uno per indicare la formattazione desiderata tramite markdown (estratto tramite Table to Markdown), l'altro per specificare meglio il contesto in cui scrivere gli use cases.

- **Servizi dei sottosistemi**: alto, sono stati necessari in media 3 prompts per ogni sottosistema. La maggior parte delle manipolazioni necessarie, soprattutto quelle più consistenti, sono state ottenute grazie al bot. Piccole modifiche sono state fatte manualmente in modo da risparmiare tempo. Unica eccezione è stata l'ultimo sottosistema, e-commerce, che ha necessitato di forti correzioni manuali che il bot non è riuscito ad elargire con un prompt apposito.

4.3.3 Object design

- **Packages**: medio-basso, la suddivisione in packages corretta è stata ottenuta con facilità grazie a soli 2 prompts. Leggermente più difficoltà si è riscontrata nel generare diagrammi tramite PlantUML dei singoli package. I primi 3 package hanno richiesto in media 3 prompts ciascuno e gli ultimi 2 invece, grazie alla context window del bot, hanno richiesto solo 1 prompt ciascuno.

- **Class interfaces**: medio-alto, le interfacce in linguaggio Java sono state ottenute con facilità, grazie ad 1 solo prompt. È stato più difficile invece trasformare queste interfacce Java in interfacce generiche sotto forma di tabella al fine di inserirle nel documento di object design: ci sono voluti quasi una decina di prompts iniziali di try and error per ottenere il contesto operativo giusto in cui poi far generare le interfacce corrette con una media di 2 prompts ciascuna.

- **Design pattern**: basso, se già si è consapevoli del design pattern che si vuole adottare, tramite anche un 1 solo prompt è possibile ottenere una descrizione soddisfacente del pattern calato nel contesto del sistema. Tramite un ulteriore prompt è possibile inoltre ottenere un diagramma dell'applicazione del design pattern all'interno del sistema.

4.3.4 Testing

- **Category partition**: alto, per ogni category partition sono necessari più prompts, una media di circa 10 che varia a seconda della complessità dello use case. È difficile ottenere un category partition perfetto, il più delle volte sono richieste manipolazioni manuali. Le combinazioni sono un'incognita: a volte vengono scritte correttamente, altre volte no: di conseguenza vanno sempre controllate ed eventualmente corrette.

- **Test case specification**: medio-basso, sono stati necessari dei prompts iniziali per configurare il contesto operativo del bot. Una volta fatto ciò, ogni test case specification, per ogni funzionalità, è stato ottenuto con un singolo prompt contenente il markdown del category partition e dei test cases. Le risposte sono perlopiù corrette e solo saltuariamente potrebbero richiedere piccole modifiche, più facilmente adoperabili in maniera manuale piuttosto che scomodando il bot, che potrebbe anche sbagliare la correzione o perdere traccia del contesto operativo.

💬 **Take-away Message #5.** Lo sforzo richiesto per ottenere una soluzione accettabile da ChatGPT è variabile a seconda della specifica attività di software engineering e dall'esperienza del prompt engineer

CAPITOLO 5

Conclusioni e sviluppi futuri

Il lavoro svolto in questa tesi ha proposto un ulteriore metodo per utilizzare l'intelligenza artificiale al fine di agevolare le attività di software engineering: Complessivamente ChatGPT ha dato un grande supporto in quasi tutte le attività di software engineering a partire dalla fase di raccolta e analisi dei requisiti, passando per il system e object design fino al testing. Le risposte generate spesso potrebbero richiedere correzioni tramite ulteriori prompts e/o modifiche manuali: la supervisione, giudizio ed esperienza di un software engineer esperto sono sempre richieste e non possono essere sostituite da ChatGPT. Ritengo che il bot possa costituire una svolta importante nel semplificare lo sviluppo software, aumentando la produttività e la qualità della documentazione, diminuendo al tempo stesso il tempo speso per la scrittura e la ricerca di informazioni.
Sviluppi futuri di questo studio potrebbero essere testare altri chatbot simili, in quanto nello studio è stato usato specificamente ChatGPT, unico nel suo genere quando è iniziato il lavoro di tesi. Esempi di strumenti simili sono BingAI, alla cui base è presente un modello GPT-4 capace di accedere ad internet e Bard, fondato sul modello PaLM 2 LLM di Google. Un ulteriore approfondimento potrebbe riguardare l'uso di GPT-4, in quanto nello studio è stato usato il modello gratuito GPT-3.5, meno performante rispetto al modello GPT-4 a pagamento.

Sarebbe quindi interessante testare quest'ultimo modello e valutare le differenze con GPT-3.5. Altri sviluppi futuri potrebbe riguardare l'uso della lingua inglese, in quanto nello studio è stata usata la lingua italiana, che comparata invece alla lingua inglese, è molto meno presente nei dati di addestramento di ChatGPT. È da valutare quindi un eventuale miglioramento delle performance interagendo con il bot in inglese. Ulteriori sviluppi futuri potrebbe riguardare anche usare più chat, in quanto nello studio è stata usata una singola chat in cui sono stati sottomessi tutti i prompts, rendendo a volte difficile sfruttare appieno la context window del bot. Usare più chat contemporaneamente potrebbe rendere più semplice configurare il bot per rispondere in una certa maniera. Altro approfondimento potrebbe riguardare integrare ChatGPT con altri strumenti AI, in quanto nello studio è emersa l'impossibilità da parte del bot di generare un class diagram: un supporto potrebbe essere dato ad esempio da tool AI che generano immagini come ad esempio DALL-E e Midjourney. Infine sarebbe interessante valutare l'uso e il conseguente impatto di ChatGPT da parte degli sviluppatori in ambito aziendale.

Bibliografia

[1] "15 amazing real-world applications of ai everyone should know about ". [Online]. Available: https://www.forbes.com/sites/bernardmarr/2023/05/10/15-amazing-real-world-applications-of-ai-everyone-should-know-about/?sh=28c9e08685e8 (Citato a pagina 4)

[2] J. Ivers and I. Ozkaya, "Ai for software engineering," CARNEGIE-MELLON UNIV PITTSBURGH PA, Tech. Rep., 2021. (Citato a pagina 4)

[3] A. Perini, A. Susi, and P. Avesani, "A machine learning approach to software requirements prioritization," *IEEE Transactions on Software Engineering*, vol. 39, no. 4, pp. 445–461, 2012. (Citato a pagina 4)

[4] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "Towards supporting software engineering using deep learning: A case of software requirements classification," in *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE, 2017, pp. 116–120. (Citato a pagina 5)

[5] M. A. Rahman, M. A. Haque, M. N. A. Tawhid, and M. S. Siddik, "Classifying non-functional requirements using rnn variants for quality software development," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 25–30. (Citato a pagina 5)

[6] M. Singh, V. Anu, G. S. Walia, and A. Goswami, "Validating requirements reviews by introducing fault-type level granularity: A machine learning approach," in *Proceedings of the 11th Innovations in Software Engineering Conference*, 2018, pp. 1–11. (Citato a pagina 6)

[7] G. Zhao and J. Huang, "Deepsim: deep learning code functional similarity," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 141–151. (Citato a pagina 6)

[8] A. K. Dwivedi, A. Tirkey, R. B. Ray, and S. K. Rath, "Software design pattern recognition using machine learning techniques," in *2016 ieee region 10 conference (tencon)*. IEEE, 2016, pp. 222–227. (Citato a pagina 6)

[9] H. Thaller, L. Linsbauer, and A. Egyed, "Feature maps: A comprehensible software representation for design pattern detection," in *2019 IEEE 26th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 2019, pp. 207–217. (Citato a pagina 7)

[10] "what is natural language processing (nlp)?". [Online]. Available: https://www.oracle.com/hk/artificial-intelligence/what-is-natural-language-processing/ (Citato a pagina 7)

[11] "natural language processing". [Online]. Available: https://www.deeplearning.ai/resources/natural-language-processing/ (Citato a pagina 8)

[12] "what is a chatbot?". [Online]. Available: https://www.oracle.com/chatbots/what-is-a-chatbot/ (Citato a pagina 8)

[13] "an overview of bard: an early experiment with generative ai". [Online]. Available: https://ai.google/static/documents/google-about-bard.pdf (Citato a pagina 8)

[14] "how chatgpt works: the model behind the bot". [Online]. Available: https://towardsdatascience.com/how-chatgpt-works-the-models-behind-the-bot-1ce5fca96286 (Citato a pagina 8)

[15] H. Tian, W. Lu, T. O. Li, X. Tang, S.-C. Cheung, J. Klein, and T. F. Bissyandé, "Is chatgpt the ultimate programming assistant–how far is it?" *arXiv preprint arXiv:2304.11938*, 2023. (Citato a pagina 8)

[16] G. Sridhara, S. Mazumdar *et al.*, "Chatgpt: A study on its utility for ubiquitous software engineering tasks," *arXiv preprint arXiv:2305.16837*, 2023. (Citato a pagina 8)

[17] A. Ahmad, M. Waseem, P. Liang, M. Fahmideh, M. S. Aktar, and T. Mikkonen, "Towards human-bot collaborative software architecting with chatgpt," in *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, 2023, pp. 279–285. (Citato a pagina 8)

[18] S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, "The programmer's assistant: Conversational interaction with a large language model for software development," in *Proceedings of the 28th International Conference on Intelligent User Interfaces*, 2023, pp. 491–514. (Citato a pagina 9)

[19] "40 chatgpt use cases in 2023". [Online]. Available: https://research.aimultiple.com/chatgpt-use-cases/ (Citato a pagina 10)

[20] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, "Design patterns: Abstraction and reuse of object-oriented design," in *ECOOP'93—Object-Oriented Programming: 7th European Conference Kaiserslautern, Germany, July 26–30, 1993 Proceedings 7*. Springer, 1993, pp. 406–431. (Citato a pagina 30)