生成式 AI 輔助軟體系統架構工具之設計與開發

Design and Development of a Generative AI-Assisted Learning Software System Architecture Tool

孔崇旭，林峻劭*，陳姵予
國立臺中教育大學 資訊工程學系
* bcs113118@gm.ntcu.edu.tw

【摘要】隨著軟體系統日趨複雜，軟體工程透過標準化開發流程提升開發效率、確保軟體品質並降低維護成本，而其中需求規格的完整性與準確性對最終成品影響深遠。近年來，生成式人工智慧在軟體工程中的應用逐漸受到關注，其可輔助需求解析、加速程式碼生成，提升開發效率。然而，目前生成式人工智慧仍面臨程式碼品質參差、難以處理大型軟體架構等挑戰。因此，本研究提出一種結合程式函式、框架與自然語言需求規格的方法，透過函式級別的生成與架構優化，提升 AI 生成程式碼的準確性與穩定性，縮短開發週期，並增強開發人員在大型軟體系統中的規劃能力，以提升軟體工程的整體效能。
【關鍵字】生成式人工智慧；軟體工程；需求規格；程式框架；

Abstract: *As software systems become increasingly complex, software engineering employs standardized development processes to enhance efficiency, ensure software quality, and reduce maintenance costs. Among these, the completeness and accuracy of requirement specifications significantly impact the final product. In recent years, the application of generative artificial intelligence (AI) in software engineering has garnered growing attention, offering assistance in requirement analysis and accelerating code generation to improve development efficiency. However, current generative AI models face challenges such as inconsistent code quality and difficulties in handling large-scale software architectures. This study proposes an approach that integrates function-level code generation, software frameworks, and natural language requirement specifications. By optimizing function-level generation and architectural structuring, the proposed method enhances the accuracy and stability of AI-generated code, shortens development cycles, and strengthens developers' capabilities in planning large-scale software systems, ultimately improving the overall efficiency of software engineering.*
Keywords: Generative Artificial Intelligence, Software Engineering, Requirement Specification, Software Framework

1.前言

隨著軟體工程技術的發展，開發效率、軟體品質與長期維護成本已成為業界關注的核心議題。傳統軟體開發流程涵蓋需求分析、設計、開發與測試等階段，其中需求分析作為開發初期的關鍵步驟，不僅影響系統架構的合理性，更直接決定最終成品的品質。然而，由於需求描述的不確定性與開發過程中的人為誤差，軟體工程常面臨開發週期冗長、維護成本高昂以及需求變更所帶來的修改導致的大量時間與人力成本等挑戰。

在此背景下，生成式人工智慧（Generative Artificial Intelligence）的崛起為軟體工程領域帶來了新的可能性。生成式 AI 能夠快速生成程式碼、輔助理解軟體需求規格，並提升開

發人員的生產力，從而縮短開發週期並減少人力成本。然而，現有的生成式 AI 仍然面臨無法處理大型系統開發、程式碼品質不穩定等挑戰。因此，如何將生成式 AI 有效應用於軟體開發，提升生成程式碼的品質與可維護性，並降低開發風險，成為一項重要的研究課題。

為了解決這些問題，本研究提出一個基於程式函式、程式框架與自然語言函式規格的新穎方法，透過將程式碼生成單位細分為函式級別，並結合函式架構與明確定義的函式需求規格，來提升生成 AI 產出的穩定性與可用性。該方法不僅能夠有效防止 AI 產生幻覺（Hallucination），還能強化開發人員在大型軟體系統中的規劃能力，從而進一步縮短軟體開發週期，並提升最終產品的品質。

本研究的目標是通過結合生成式 AI 與需求規格，探索其在軟體開發領域的應用潛力，通過規劃大型軟體架構，填寫嚴謹且制式化的需求規格，最終達成自動化大型軟體系統的開發，藉此大幅縮短軟體開發流程，並維持最終成品的穩定性。

2.相關研究

2.1. 軟體系統架構的核心價值

軟體系統架構作為軟體開發的基礎框架，其核心在於如何將龐大的系統拆解為小型且可控的單元，並建立這些單元之間的協作關係。透過合理的架構設計，開發人員能夠有效降低系統的複雜性，使得整體開發與維護工作更加清晰可行。這種模組化的方法不僅提升了開發效率，也讓大型軟體專案變得更易於管理與擴展。

根據 (Bass et al., 2021)等人所述，優秀的軟體架構師應當具備以下關鍵能力：

1.設計具備高可讀性與高可維護性的架構，同時能夠解決複雜的技術挑戰

2.具備分析現有系統的能力，能夠找出影響效能、安全性等高層次問題的根本原因

3.熟悉團隊內的開發流程，並能夠清晰地向開發團隊傳達架構設計與實施要求，通常需要高度的技術專業能力

4.熟知多種開發技術，並能夠針對不同的專案需求選擇最適合的技術方案

2.2. 軟體系統架構對軟體品質的影響

(Garlan & Perry, 1995)兩人指出，軟體系統架構的核心關注點在於大型軟體系統的組織與整體結構，而非單一模組內的演算法或資料結構。優質的架構設計不僅能夠提高系統的可理解性、可重用性、可擴展性、一致性及可管理性，同時也能夠顯著降低開發時間與成本。此外，對個人而言，深入學習軟體架構能夠拓展技術視野，提升專業競爭力，對企業而言，具備架構設計能力的團隊能夠為組織帶來更高的技術價值與市場競爭優勢。

2.3. 生成式人工智慧的興起與多領域的應用

近年來，生成式人工智慧的發展突飛猛進，並廣泛滲透至人們的日常生活(Pelau et al., 2021)。這類技術的核心優勢在於模仿人類的思維模式，能夠流暢地進行自然語言互動、資訊查詢，甚至在某些情境下提供情感支持(Adiwardana et al., 2020; Arteaga et al., 2019; Dibitonto et al., 2018; Falala-Séchet et al., 2019; Schaaff et al., 2023) 。除了這些應用，生成式 AI 在程式設計領域的角色也日益顯著。它不僅可以擔任對話式助理來協助開發者解決技術問題，還能夠直接生成程式碼，為軟體開發帶來嶄新可能性(Loh, 2023; Mollick, 2022)。而伴隨著開發工具的演進，現今的整合開發環境（Integrated Development Environment, 以下簡稱 IDE）、單元測試框架以及效能分析工具已讓程式碼撰寫的門檻大幅降低，使得不論是資深開發者還是初學者，都能更輕鬆地進行程式開發(Alizadehsani et al., 2022)。然而，即便技術層面日益進步，軟體開發依舊是一項高度仰賴人力與時間的工作，如何有效減少開發成本並

提升效能仍是業界關注的重點(Bluemke & Malanowska, 2021; Butt et al., 2021; Jadhav et al., 2022)

2.4. 生成式 AI 在程式設計的優勢與限制

雖然生成式 AI 被視為提升開發效率的重要技術，它仍然存在許多限制。首先，AI 產出的程式碼並非總是正確或穩定的，其可靠性仍受限於訓練數據與演算法的能力。其次，AI 生成的程式碼在執行效率與可維護性 方面可能存在不足，這對於長期開發專案而言是一項潛在風險(Zhang et al., 2023)。因此，依賴 AI 進行輔助開發時，開發者仍需對 AI 產出的程式碼進行細緻的審查與測試，以確保其符合專案需求並達到可接受的品質標準。

根據 (Idrisov & Schlippe, 2024)兩人於 2024 年的研究結果，即便生成式 AI 在某些情況下能夠生成正確的解法，大多數情境下仍無法穩定產出高效且可維護的程式碼。這說明，AI 雖然能夠加快開發過程，但仍然無法完全取代人類的程式設計能力。即便如此，生成式 AI 作為輔助工具，仍能夠大幅加速開發週期，使開發人員能夠更迅速地建立雛形並進行後續優化。因此，未來的研究方向應聚焦於如何透過更精確的引導方式，提升 AI 生成程式碼的品質與適用性，使其在軟體開發領域發揮更大效益。

3.系統設計與展示

本研究結合生成式人工智慧，開發一個輔助軟體開發流程的系統。使用者繪製類別圖後，將產生出的程式框架上傳至系統上，並以單一個函式為單位，依據引導填寫函式的需求規格。當所有軟體系統中的函式都生成完後，使用者可以通過該系統將所有函式進行合併。如果在生成式人工智慧生成完成式碼後覺得不妥，可以重新撰寫需求規格並再次上傳，亦可在系統上直接通過程式編輯器進行修改。此外，本系統的需求規格與程式碼皆有版本控制功能，可以讓使用者推回至前任一版本。

3.1. 系統架構

系統架構設計圖如(圖 1)所示，以下進行分項說明：

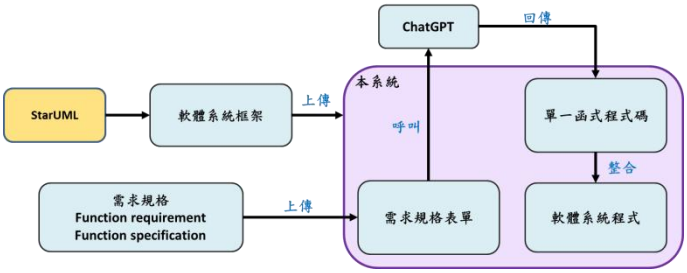

圖 1 系統架構設計圖

1.使用者通過 StarUML 繪製類別圖後，通過該軟體的插件生成無函式內容的程式框架

2.將軟體系統框架上傳於本系統

3.將對應函式的需求規格通過需求規格表單填寫至本系統

4.系統通過呼叫 ChatGPT API，透過軟體系統框架與需求規格生成單一的函式程式碼

5.通過本系統合併所有函式，最終完成軟體系統的開發

3.1.1 生成式 AI 模型版本與 API 相關設定

本系統使用 OpenAI gpt-4o 進行函式程式碼的生成，模型使用 OpenAI gpt-4o，並設定 temperature 為 0，用以減少幻覺並提升生成內容的一致性；max_tokens 設為上限值以避免截斷；其餘生成參數（如 frequency_penalty）皆設為 0，避免系統干預模型自然輸出。

為提升生成內容對程式架構與需求規格的符合度，系統在每次呼叫 API 時均設定 system 級別提示語（system prompt），明確指定模型角色為「具備專業經驗的 Java 資深開發人員」，並要求其依照使用者所提供之函式結構與需求內容進行函式本體的生成。此類提示語已透過研究團隊多輪測試與調整，能有效引導模型生成邏輯一致、可讀性高且可編譯的程式碼，亦為本系統提升生成穩定性之關鍵機制之一。

3.2. 系統展示

系統頁面圖如(圖 2)至(圖 5)所式，以下進行分項說明：

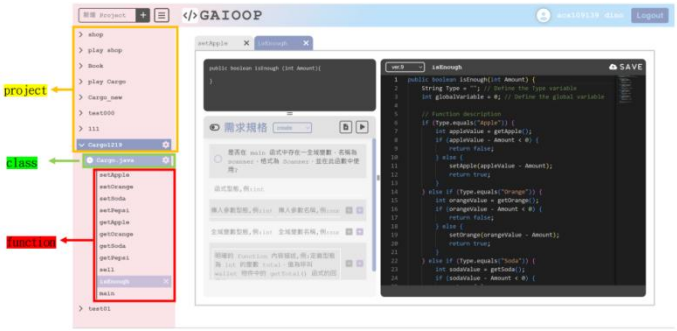

圖 2 系統首頁

(圖 2)所示為本研究所開發的系統的首頁，使用者可以在該頁面的左上角進行新專案(Project)的建立。在左側欄位中的物件(Class)選項，可以將產生完成的程式框架上傳。當上傳完成後，即可在左側欄位中看到該物件中所擁有的函式(Function)。

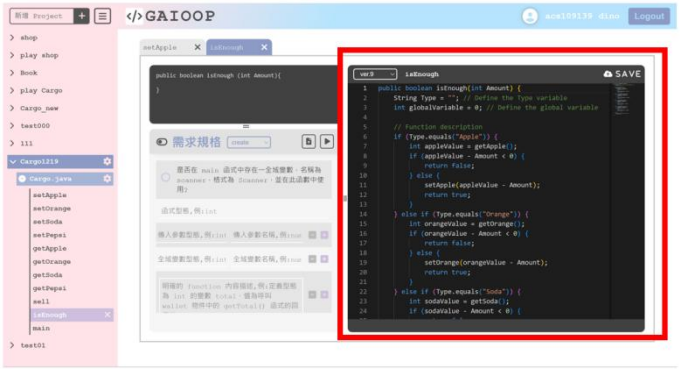

圖 3 程式碼編輯器

(圖 3)所示為本研究所開發的系統的其中一項功能，該功能容許使用者在系統上即時編輯程式碼，不論是生成式人工智慧所生成的，還是已經被使用者自行修改過的程式碼，都可以再進行修改，此外，該功能還可以顯示整個程式碼的縮圖(如紅色方框右上角)，容許使用者快速導覽單一函式中的整體程式碼，同時可以讓使用者通過滑鼠左鍵點擊快速跳轉至所點選的段落。

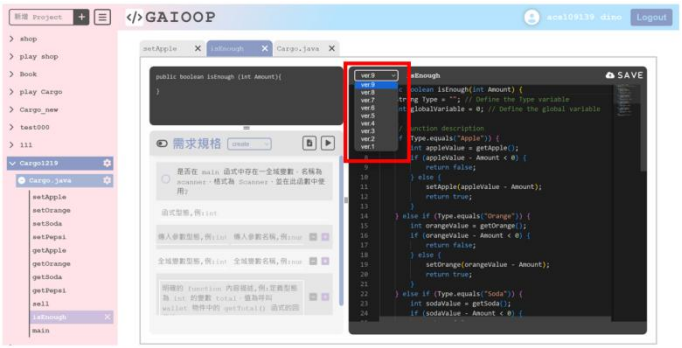

圖 4 版本控制功能

(圖 4)展示了本系統的版本控制功能，此功能建構於 MINIO 容器之上。使用者可在提交需求規格後、手動編輯程式碼後、選擇歷史版本後，或手動存檔後，將該版本的需求規格或程式碼設為最新版本，以確保版本管理的靈活性與可追溯性。

圖 5 需求規格填寫表單

(圖 5)呈現了本系統的需求規格表單，該表單對使用者的需求規格撰寫提供了明確的引導。透過標準化的填寫方式，使用者需針對 函式型態、傳入參數、同物件內的全域變數以及函式的具體內容描述進行規範化的撰寫，以確保需求規格的完整性與嚴謹性。

4.目前結果

本研究已完成輔助學習系統的功能建置，並於內部實驗環境下進行多輪測試，以確保需求規格撰寫、程式碼生成與整合流程的基本正確性與操作穩定性。為評估本系統在實際場域中的實用性與使用者接受度，本研究招募臺中某大學資訊工程學系碩士班與學士班學生共十五人，進行實測操作與問卷評估。

4.1. 系統使用成效評估

使用者測試內容包含：繪製 UML 類別圖、填寫函式需求規格、觸發 AI 生成對應函式程式碼，以及最終整合完整程式架構。統計結果顯示，多數使用者能在約兩小時內完成一套中型應用系統的開發，平均使用提示語（Prompt）次數為 50 次，遠低於使用一般網頁版 ChatGPT 需 100 次以上交互對話與 9 小時的作業時間。此結果證實本系統具備縮短開發流程與強化任務導向生成效率之潛力。

實驗中使用者共建置四種類型之應用系統：個人聯絡簿、圖書管理、工作任務管理與倉儲管理系統。系統自動生成的程式碼行數平均介於 1800 至 2300 行，內容涵蓋多個模組與物

件結構，顯示本系統具備支援中大型應用程式開發的能力，且模組化程度與語法正確性亦可接受。

4.2. 初步分析說明

為評估系統之易用性與使用者體驗，研究團隊對參與實驗的學生實施系統易用性量表的調查，其中包含 SUS(System usability scale)軟體易用性評估、5Es(Effectiveness, Efficiency, Engaging, Error tolerance, Easy to use)軟體評估與軟體淨推薦分數表(Net Promoter Score)評估，其評估結果統計如表 2、圖 6~圖 8 所示。統計結果分別說明如下：

系統易用性量表設計涵蓋學習性、效率、一致性、信心與使用意願等面向，最終經轉換得出 0~100 分的 SUS 總分，作為整體可用性評估指標。本次實驗共蒐集來自碩士班與學士班學生共 15 份有效問卷，經 SUS 計分後，其統計結果如下表 1 所示：

表 1 系統易用性量表分析

平均數（Mean）	標準差（SD）	最高分	最低分	樣本數（N）
82.0	19.21	100.0	27.5	15

根據(Bangor et al., 2009)等人所提出的 SUS 評等標準，SUS 平均得分為 82.0 時，對應之形容詞等級為「Excellent（優秀）」，其綜合評分區間約落於 80 至 90 分之間，平均為 85.5 分。此等級代表系統的整體可用性極佳，在使用者心中具備高度的正面評價，且屬於可用性測量中的上位百分群（約第 90 百分位）。

以下為 SUS 軟體易用性評估、5Es 軟體評估與軟體竟推薦分數表的詳細分析

4.2.1. SUS 軟體易用性評估(如圖 6 所示) 十題中的所有正向題的平均皆為 4.3 分，這證明了本系統對於使用者來說相當友善；「我想我需要有人幫助才能使用這個輔助學習系統。」與「我覺得這個輔助學習系統過於複雜。」在反向題中的分數較差，這顯示了本系統可能還不夠新使用者友善，在剛接觸平台時操作可能會不熟悉，需要加入更多介面的提示或操作引導，以進一步提升自主使用的流暢度。

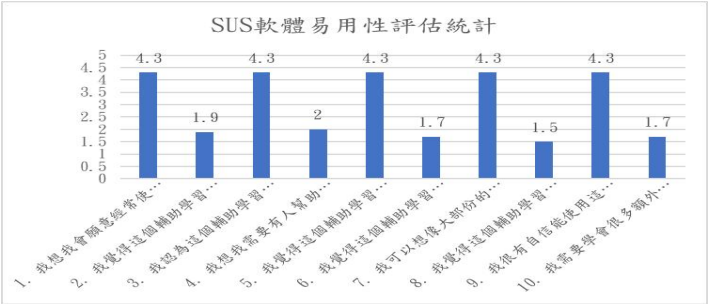

圖 6 SUS 軟體易用性評估統計

4.2.2. 5Es 軟體評估(如圖 7 所示) 本研究參考(Quesenbery, 2004)所提出的 5Es 使用性評估模型，從五個方面分析系統的可用性表現，分別為：有效性（Effectiveness）、效率性（Efficiency）、吸引力（Engaging）、容錯性（Error Tolerance）與易用性（Easy of use）。此模型強調不同使用情境下使用者需求的多樣性，並提供評估使用者互動體驗的多維觀點。

調查結果顯示，「我覺得這個輔助學習系統的畫面是簡潔的」獲得平均 4.5 分，代表系統在吸引力方面具有良好表現，使用者對其視覺設計與互動風格給予正面肯定。反之，「我無法接受輔助學習系統內出現許多錯誤」（負向題）僅獲得 2.0 分，反映出系統在容錯性方面仍有待改善，使用者在操作過程中仍可能遭遇未預期錯誤或回復不易的情況。

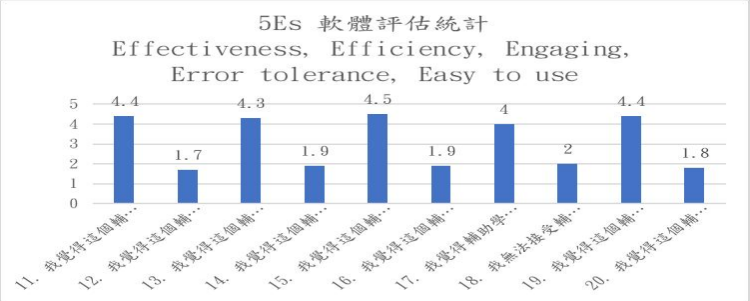

圖 7 5Es 軟體評估

4.2.3. 軟體淨推薦分數表(如圖 8 所示) 本研究依據(Reichheld, 2003)提出之 Net Promoter Score（NPS）模型，調查使用者對本系統的推薦意圖。根據 15 份有效問卷，有 7 人為推廣者（給予 9~10 分）、6 人為中立者（7~8 分）、2 人為貶抑者（0~6 分），計算得出 NPS 分數為 +33.3，顯示系統具備一定程度的使用者忠誠度與正向推薦傾向。雖未達企業級產品常見之+50 標準，但已展現良好發展潛力。

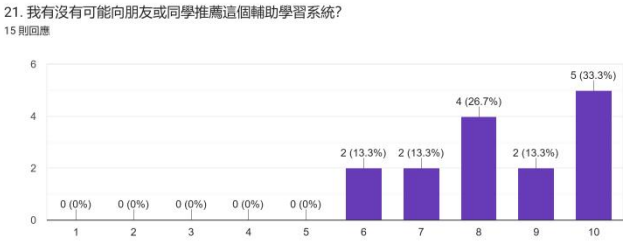

圖 8 軟體淨推薦分數表

4.3. 小結與問題反思

整體成果顯示，該系統在架構設計、生成式人工智慧整合以及任務導向操作流程等面向皆具高度成熟度，能有效協助使用者完成從需求規格撰寫到程式碼生成的完整開發流程。實測結果亦指出，其在開發效率與使用者滿意度上展現顯著優勢。

然而，儘管系統在大多數易用性指標中獲得正向評價，根據 SUS 量表中反向題項的回應結果顯示，部分初次接觸系統者在操作過程中出現輕微的不確定感。此現象可能源於使用者對需求規格填寫格式、操作邏輯或流程步驟尚未完全熟悉。

為進一步強化研究的說服力與成果的解釋能力，未來將拓展使用者研究的實驗方法，採用實驗組與對照組進行實驗，控制組將使用一般網頁版 ChatGPT 進行自然語言需求描述與程式碼生成，實驗組則使用本研究開發之系統進行相同任務操作。藉此可在開發時間、交互次數、錯誤率與程式碼品質等量化指標上進行對比分析，更精確地驗證本系統在引導結構化需求撰寫與任務導向生成流程方面的優勢。

5.未來展望

本研究已順利完成系統的基本建置，並驗證其於需求規格管理與程式碼生成方面的可行性。然而，目前系統尚缺乏自動化測試機制，需由開發者手動驗證生成的 Java 程式碼是否正確，當程式碼無法編譯或執行錯誤時，使用者需自行排除問題，這不僅降低了開發效率，也增加錯誤遺漏的風險。未來將針對自動化測試流程進行整合與優化，以提升系統的實用性與穩定性。此外，本研究也將聚焦於引入靜態程式碼分析、測試覆蓋率評估及人工智慧輔助的程式碼品質評分等技術，以作為補強生成碼可靠性之客觀佐證。

6.致謝

本研究承蒙教育部「教學實踐研究計畫」(計畫編號：PEE1137600)與國立臺中教育大學「高教深耕計畫」之經費支持與資源挹注，特此致謝。亦感謝本校師長與研究團隊在研究過程中的協助與建議，使本研究得以順利進行。

參考文獻

Adiwardana, D., Luong, M.-T., So, D. R., Hall, J., Fiedel, N., Thoppilan, R., Yang, Z., Kulshreshtha, A., Nemade, G., & Lu, Y. (2020). Towards a human-like open-domain chatbot. *arXiv preprint arXiv:2001.09977*.

Alizadehsani, Z., Gomez, E. G., Ghaemi, H., González, S. R., Jordan, J., Fernández, A., & Pérez-Lancho, B. (2022). Modern integrated development environment (ides). Sustainable Smart Cities and Territories,

Arteaga, D., Arenas, J., Paz, F., Tupia, M., & Bruzza, M. (2019). Design of information system architecture for the recommendation of tourist sites in the city of Manta, Ecuador through a Chatbot. 2019 14th Iberian conference on information systems and technologies (CISTI),

Bangor, A., Kortum, P., & Miller, J. (2009). Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies*, *4*(3), 114-123.

Bass, L., Clements, P., & Kazman, R. (2021). *Software architecture in practice*. Addison-Wesley Professional.

Bluemke, I., & Malanowska, A. (2021). Software testing effort estimation and related problems: A systematic literature review. *ACM Computing Surveys (CSUR)*, *54*(3), 1-38.

Butt, S. A., Misra, S., Piñeres-Espitia, G., Ariza-Colpas, P., & Sharma, M. M. (2021). A cost estimating method for agile software development. Computational Science and Its Applications–ICCSA 2021: 21st International Conference, Cagliari, Italy, September 13–16, 2021, Proceedings, Part VII 21,

Dibitonto, M., Leszczynska, K., Tazzi, F., & Medaglia, C. M. (2018). Chatbot in a campus environment: design of LiSA, a virtual assistant to help students in their university life. Human-Computer Interaction. Interaction Technologies: 20th International Conference, HCI International 2018, Las Vegas, NV, USA, July 15–20, 2018, Proceedings, Part III 20,

Falala-Séchet, C., Antoine, L., Thiriez, I., & Bungener, C. (2019). Owlie: a chatbot that provides emotional support for coping with psychological difficulties. Proceedings of the 19th ACM international conference on intelligent virtual agents,

Garlan, D., & Perry, D. E. (1995). Introduction to the special issue on software architecture. *IEEE Trans. Software Eng.*, *21*(4), 269-274.

Idrisov, B., & Schlippe, T. (2024). Program code generation with generative ais. *Algorithms*, *17*(2), 62.

Jadhav, A., Kaur, M., & Akter, F. (2022). Evolution of software development effort and cost estimation techniques: five decades study using automated text mining approach. *Mathematical Problems in Engineering*, *2022*(1), 5782587.

Loh, E. (2023). ChatGPT and generative AI chatbots: challenges and opportunities for science, medicine and medical leaders. *BMJ leader*, leader-2023-000797.

Mollick, E. (2022, December 14). *ChatGPT is a tipping point for AI*. Harvard Business Review https://hbr.org/2022/12/chatgpt-is-a-tipping-point-for-ai

Pelau, C., Dabija, D.-C., & Ene, I. (2021). What makes an AI device human-like? The role of interaction quality, empathy and perceived psychological anthropomorphic characteristics in the acceptance of artificial intelligence in the service industry. *Computers in Human Behavior*, *122*, 106855.

Quesenbery, W. (2004). Balancing the 5Es of usability. *Cutter IT Journal*, *17*(2), 4-11.

Reichheld, F. F. (2003). The one number you need to grow. *Harvard business review*, *81*(12), 46-55.

Schaaff, K., Reinig, C., & Schlippe, T. (2023). Exploring ChatGPT's empathic abilities. 2023 11th international conference on affective computing and intelligent interaction (ACII),

Zhang, B., Liang, P., Zhou, X., Ahmad, A., & Waseem, M. (2023). Practices and challenges of using github copilot: An empirical study. *arXiv preprint arXiv:2303.08733*.