Transforming software size estimation with prompt engineering: A ChatGPT-based framework for component-based systems

Kumesh, T.[1], Jayalal, S.[2], Wijayasiriwardhane, T. K.[3]
[1,2,3]Department of Industrial Management, Faculty of Science, University of Kelaniya, Sri Lanka [2]Center for Information Technology, Waikato Institute of Technology, Hamilton, New Zealand

***Abstract*—In software project management, it is essential to estimate software size accurately since knowing the size of the program allows for more efficient p lanning, e stimation, and scheduling of its development. The conventional software size estimation approaches such as use of Function Points (FP) and its extensions often prove to be time consuming, resource-intensive, and thereby a costly exercise, demanding specialized human expertise. This has no difference when it comes to the modern software development paradigms like Component-Based Systems Development (CBSD). On the other hand, concerning Artificial Intelligence (AI), the large language model-based chat-bots like ChatGPT, Bard AI, DALL-E, and Midjourney are excelling at automating traditional human activities and interactions in various domains including software engineering. Among all these AI tools, ChatGPT has proven its applicability in many industries including software engineering, acquiring around 100% accuracy over other chat-bots. In this paper, we therefore developed and validated an innovative framework based on AI, to measure the size of a Component-Based Systems (CBS) using ChatGPT. The framework which consists of a set of prompts has been designed to expedite the size estimation process of a CBS using an extension of FP called Component Point (CP) while substantially reducing the need of human involvement and financial o utlay. O ur a im i s n ot o nly t o e nhance t he efficiency of software size estimation but also to conserve both time and financial r esources t hat w ould o therwise b e s pent o n practicing conventional approaches. We therefore envision that the proposed approach would revolutionize the landscape of software project management.**

***Index Terms*—Artificial I ntelligence (AI), C hatGPT, Component Point (CP), Component-Based Systems (CBS), Software sizing**

I. Introduction

In the ever-evolving world, the harsh reality of software development has surfaced from the Standish Group Chaos report in 2020 [1]. A staggering 50% of software projects have failed to meet goals such as on-time delivery, budget adherence, as well as while 19% suffered a total failure, resulting in the waste of valuable corporate resources. Addressing the challenges these software project failure is no small feat, demanding specialized management skills and critical business decisions.

Effective management of software products, processes, and resources is essential to the success of software development [2]. One pivotal aspect of this management is effort estimation, a metric that serves as the backbone for estimating overall cost and schedule. It is also noteworthy that the software development effort estimation is significant in the early stages of the Software Development Life Cycle (SDLC) [3], as these are the formative stages the crucial decision-making activities, such as planning, budgeting, and allocation of resources take place. Unfortunately, erroneous estimations or planning missteps during this crucial period can result in far-reaching consequences, affecting not only software development projects but also their broader outputs [3].

As a result, early estimation of the effort of software development has become a paramount concern [4], [5]. Among the various approaches-ranging from algorithmic models [6], [7] to analogy-based methods [8], [9] and neural network approaches [10], [11] proposed for estimating the software development effort have one common fact: their dependence on the size of the software as the primary effort driver for the estimation. However, it is undeniable that the manual process of measuring software size using conventional approaches, such as the use of Function Point (FP) and its extensions, is time-consuming, resource-intensive, and thereby costly, demanding specialized human expertise [12].

The way software applications are developed has fundamentally changed as a result of the paradigm shift from conventional software development to Component-Based Software Development (CBSD). Practitioners have embraced the rewards of using CBSD over traditional software development [13]–[15]. In addition, there are certain difficulties with CBSD for techniques that assist with software development. The applicability of most of the conventional software measures to CBS have become limited, mostly as a result of the components' black-box nature. In response, [2] an extension to FP, named Component Points (CP) has been proposed for measuring the size of a CBS [2]. However, like FP analysis, the CP counting process is also time-consuming, resource-intensive, and therefore a costly exercise, demanding specialized human expertise. Therefore, there is little improvement even when it comes to relatively modern software development

Correspondence: Tharindu Kumesh (E-mail: anandakt-im18007@stu.kln.ac.lk)
Received: 16-06-2024 **Revised:** 12-08-2024 **Accepted:** 09-09-2024
Tharindu Kumesh, Shantha Jayalal and Thareendhra Wijayasiriwardhane are from University of Kelaniya (anandakt-im18007@stu.kln.ac.lk, shantha@kln.ac.lk, thareen@kln.ac.lk)
DOI: https://doi.org/10.4038/icter.v18i2.7286
The 2025 Special Issue contains the full papers of the abstracts published at the 24th ICTer International Conference.

paradigms such as CBSD. This underscores the pressing need for innovative approaches that can streamline and automate the software size measurement to reduce not only the time and effort required but also the need for human expertise, while enhancing the accuracy and efficiency of the process.

Recently, the trend towards AI-based automation has gained tremendous momentum across the globe, driven by the promise of cost reduction, human error minimization, risk mitigation, and efficiency enhancement [16], [17]. In the realm of AI, the large language model-based chatbots such as ChatGPT, Bard AI, DALL-E, and Midjourney are excelling in automating traditional human activities and interactions. Among them, ChatGPT has demonstrated its applicability in various domains including Software Engineering, and has gained around 100% accuracy [18]–[23] outperforming others.

Against this backdrop, we developed an innovative AI-based framework that can be utilized to measure the size of CBS using ChatGPT. Based on the results of the experiments we conducted, a framework consisting of specially created prompts for CP computation was developed. It has been validated by applying it to real CBS projects and has achieved the acceptable model accuracy. The developed framework has been designed to expedite the size estimation process of a CBS using an extension of FP called CP, while substantially reducing the need for human involvement and financial outlay. Our aim is not only to enhance the efficiency of software size estimation but also to conserve both time and financial resources that would otherwise be spent on conventional approaches. We therefore envision that the proposed approach will revolutionize the landscape of software project management.

II. Related work

In [24], DeMarco states "you can't control what you can't measure". This emphasizes how crucial it is to quantify software processes, products, and resources in order to effectively manage software development. Numerous software metrics have been established over the past few decades to gather data on these elements of software development. Among these, one of the most crucial indicators of a software product is its size. Accurately measuring the size of software in the early stages of software development has long been a fundamental aspiration in software engineering [3]. Considerable research efforts have been devoted to this goal, motivated not only by the critical significance of accurate size measurement but also by its crucial function as the main factor in determining effort when measuring software development effort [6], [7], [25].

A. Function Point

Among the many software size measures proposed, Function Points(FP) gained wide acceptance in sizing software products [26], because of its applicability in the early stages of the SDLC. FP has been extended to several software development paradigms [27]–[29] and has now become an ISO standard [30]. However, a common fact among all these measures is their reliance on a manual counting process. This necessitates the availability of comprehensive software specifications, making it imperative to possess expert knowledge to derive required information [2]. As a result, estimating software size using conventional measures such as the use of FP and its extensions has become a time-consuming, resource-intensive, and therefore costly exercise [12].

B. Component-Based Software Development

The introduction of CBSD has accelerated the shift from software development to software engineering [28], [31]. This paradigm shift has introduced a novel approach where pre-compiled and reusable components, often developed by third parties, are used in the process of software development [14]. The industry has gained substantial benefits from CBSD compared to traditional software development. Notably, the use of reusable components has led to increased efficiency, significantly reducing both the time and cost of software development [13], [14]. However, the adoption of CBSD has not been without challenges. The unavailability of source codes and information regarding the class hierarchies of components, due to their black-box nature, has rendered the conventional software measures ineffective [32], [33]. Furthermore, CBS has unique characteristics, such as the interface structure of components and the usage constraints associated with them, which further restricts the application of conventional metrics [15].

C. Component Point

In response, [2] proposed an extension to FP called Component Points (CP) to measure the size of a CBS using the specification written in the Unified Modeling Language (UML) [2]. The CP counting process takes into account specific characteristics such as the number, type and the complexity of the components, as well as their interfaces and interactions, providing a more nuanced and accurate measure compared to traditional metrics.

The CP counting, process starts with the identification and classification of components. After identification, the components are divided into three categories: Domain Components (DC), User Components (UC) and Service Components (SC). The complexity of each component is then calculated. As mentioned earlier, a component's complexity is determined by both the number and the complexity of its interfaces and interactions. The Interface Complexity (IFC) is computed based on the interface type - such as Internal Logical Files (ILF) and External Interface Files (EIF) - and the interface complexity level - determined by the Number of Operations (NO), provided and their Number of Parameters (NP) as shown in Figure 1 and 2 [2].

Interaction Complexity (ITC) is calculated using the Interaction Frequency (IF) and the Complexity Measure (CM) of data types using equation 1 given below, l denotes by the number of interfaces that the component has, m represents the number of operations that the i interface provides and k is the data type of the information content involved in the interaction performed by the j operation of the i interface. The number of interactions (No) carried out by the operation,

divided by the total number of interactions (Ni) carried out by all of the interface's operations, gives the IF. Based on the number of hierarchical layers and the different data types in the component data type graph, the complexity of each data type is computed. The total of all the member data types' complexity, calculated recursively from the root to all leaf nodes, defines a data type's CM [2].

$$ITC = \Sigma_{i=1}^{l}\Sigma_{j=1}^{m}(IF_{ij}\Sigma_{k=1}^{n}CM_{ijk}) \quad (1)$$

The average of the components' IFC and ITC is used to calculate the Interface Complexity per Interface (IFCI) and the Interaction Complexity per Interaction (ITCI). Then, the component complexity level of each component is determined using Figure 3 given below. The next step involves the calculation of Unadjusted Component Points (UCP), a weighted sum derived from Figure 4,based on the component type and complexity level. The UCP is further adjusted using the Value Adjustment Factors (VAF), resulting in the final CP count [2]. However, like FP and its other extensions, the CP counting process also requires comprehensive software specifications, making expert knowledge essential for deriving information. This has made the use of CP for estimating the size of a CBS time consuming, resource-intensive and therefore a costly exercise. Consequently, there is no difference even relatively modern software development paradigms like CBSD. This highlights the need for innovative approaches that can streamline and automate the software size measurement - not only to reduce the time and effort required, but also human expertise while improving the accuracy and efficiency of the process.

NO	NP		
	1–19	20–50	51+
1	Low	Low	Average
2–5	Low	Average	High
6+	Average	High	High

Fig. 1: The Table to Determine the Interface Complexity Level

Type	Interface complexity level			Total
	Low	Average	High	
ILF	... × 7 = ...	... × 10 = ...	... × 15 = ...	...
EIF	... × 5 = ...	... × 7 = ...	... × 10 = ...	...
IFC				...

Fig. 2: The Table to Calculate the Interface Complexity(IFC)

ITCI	IFCI		
	0–4	5–8	9+
0–1	Low	Low	Average
2–3	Low	Average	High
4+	Average	High	High

Fig. 3: The Table to Determine the Component Complexity Level

Type	Component complexity level			Total
	Low	Average	High	
DC	... × 3 = ...	... × 6 = ...	... × 10 = ...	...
UC	... × 4 = ...	... × 7 = ...	... × 12 = ...	...
SC	... × 4.5 = ...	... × 7 = ...	... × 11.5 = ...	...
UCP				...

Fig. 4: The Table to Calculate the UCP Value

D. Artificial Intelligence (AI)

The advent of AI has revolutionized various industries by automating tasks and enhancing productivity [34]. In the realm of AI tools, there are several notable platforms that have emerged, offering unique capabilities and functionalities. One such tool is ChatGPT, a conversational AI chat-bot developed by OpenAI [20]. ChatGPT has gained popularity for its ability to generate human-like text responses [20], [35], automating the task of engaging in natural language conversations. Another AI tool, Bard AI, also focuses on automating conversational interactions and has been employed in various applications [36]. Additionally, AI tools like DALL-E and Midjourney offer distinct features and functionalities [12]. These technologies improve productivity, decrease the need for human interaction, and thereby accelerate various processes. With their availability, organizations can leverage AI tools to automate traditional repetitive tasks, opening up new possibilities for enhanced customer service, information retrieval, and knowledge management [18]–[23], [37]. The utilization of AI tools, such as ChatGPT and Bard AI, in automating human conversational tasks has emerged as an exciting area of research and development in the AI domain.

E. ChatGPT

In recent times, ChatGPT [20], [38] has attracted significant interest. Its use in automating a wide range of activities across various sectors has proven its adaptability. Compared to other AI tools like Bard AI, ChatGPT is applied in many disciplines, including medicine, finance, legal support, innovation, education, academic research, journalism, civil engineering [18] and visualizations [19], [20], [37]. Researchers have conducted evaluations to assess the accuracy of automated tasks performed by ChatGPT and reported promising results. Accuracy rates have approached around 100%, with only a minimal number of errors observed, none of which significantly impact the overall outcome [19], [20], [37]. These findings highlight the effectiveness and reliability of ChatGPT in automating complex tasks.

Another domain that has not escaped the influence of ChatGPT is software engineering. A number of research studies have explored the application of ChatGPT in various aspects of software engineering [21]. Auto code generation, error identification, bug fixing [22], improving code quality, refactoring, requirements elicitation [39] are among the tasks that have already been successfully accomplished using ChatGPT, exhibiting a high level of accuracy. Moreover, ChatGPT has demonstrated its capabilities in software modeling and

designing, achieving an accuracy rate of approximately 100% [19] outperforming other Generative AI tools like Bard AI. Given the impressive performance of ChatGPT in these areas, it naturally raises the question of why the measurement of software products, processes and resources cannot also be accomplished using ChatGPT. Considering its ability to understand and generate outputs with high accuracy, ChatGPT appears to be a promising tool for automating the measurement of resources, procedures, and products of software development.

III. Proposed Framework

In this section, an innovative AI based framework is proposed to measure the size of a CBS using ChatGPT. The objectives of the research are as follows:

- Conduct an experimental task to develop an initial framework consisting of a set of prompts, designed to measure the size of a CBS using ChatGPT.
- Apply the developed prompts into CBS projects whose sizes have already been estimated in the literature.
- Conduct a comparative analysis, to validate and contextualize the initial framework.
- Based on the results of the comparative analysis develop guidelines to improve the prompts to improve the accuracy of the framework for measuring the size of any CBS.
- Apply the improved framework to real industrial CBS projects to validate its applicability in industry.

The developed framework uses an extension of FP called CP [2] and UML specifications to quantify the size of a CBS. However, there are several problems associated with the manual CP counting process. These include the need for expertise knowledge, significant time, resource involvement, and consequently high costs. Another challenge is the inability to obtain size measurements in a timely manner for project planning and estimation, particularly during the critical early stages of the SDLC [3]. Given the abundance of studies examining ChatGPT's capability, [19], [22], [23], [38], [40] this research seeks to leverage those findings to automate the CP counting process and, consequently the size measurement process of a CBS using ChatGPT - over other generative AI tools like Bard AI.

An experimental approach was used, inspired by previous research [19], [22], [37], [41]. The first step involved the generation of initial prompts, following the guidelines introduced in similar research [42]. These initial prompts underwent a fine-tuning process through iterative experimentation. The observations gained from these experiments helped to fine-tune the initial prompts for using ChatGPT to complete CP calculation, which resulted in the creation of an initial framework consisting of brand-new collection of prompts. The initial framework was subsequently applied to a set of projects to ensure the broader applicability of the developed framework. The were readily available on the internet, and their CP was calculated using the conventional CP calculation method as documented in the literature [2], Validation was further conducted using the Mean Magnitude of Relative Error (MMRE) and the Percentage of Predictions within 30% of the actual (PRED(30%)) following the literature [42]–[46]. The most recent version of ChatGPT, version 3.5, which is free and open to the public was used through the experiment.

The iterative fine-tuning process of this study unfolded across three distinct phases, each characterized by significant modifications to the prompts employed. Within each phase, substantial changes were introduced with the dual purpose of getting a precise response and enhancing the accuracy of the generated response. In each phase, the prompts were applied to three different CBS projects, which were taken from the literature [47]. The CP of those projects was calculated in [2](Table I). The comparative analysis helped to check the alignment of the fine-tuned prompts in each phase with the conventional CP calculation approach.

A. Selected Projects

1) GPS Navigation System (GPS): The Global Positioning System (GPS) serves four primary functions: positioning, mapping, navigation, and system management, making it a versatile navigation tool. Through integration with the Global Positioning System Receiver (GPSR), the GPSR Controller ensures accurate location determination by delivering precise position coordinates. Utilizing the Geographic Information Server (GIS) Connection service, the system interacts with the (GIS) to access maps with relevant coordinates, enabling comprehensive mapping capabilities. Users can load maps, view current locations, trace routes, calculate distances, and obtain real-time system status updates.

2) Taxi Automation System (TAS): The Taxi Automation System (TAS) was developed to assist human controllers in efficiently managing a fleet of radio-controlled taxis in response to customer calls. TAS is tasked with tracking active drivers and their locations, automatically assigning taxis to new service requests. Its functionality spans recording charges for account clients, managing customer requests, and maintaining driver details. Dispatching and assigning taxis to jobs is a critical aspect, necessitating effective communication between controllers and drivers, often facilitated by advanced communication tools such as in-cab computers.

3) Synchronous Distance Education System (SDES): The system aims to develop a software application facilitating synchronous media-based learning. Employing H323 and T120-compliant protocols, SDES offers real-time communication features such as chat, online browsing, application sharing, audio, and video within a client-server, three-tier architecture. Users are categorized as teachers and students, with teachers granted additional rights for managing online lectures.

TABLE I: Manually calculated UCP of each project.

Project Number	The Project	UCP
1	GPS	21.5
2	TAS	2.5
3	SDES	105.5

B. Experiments

1) Phase 01: A number of experiments with varying results were carried out in the first phase. The project scenario was described, and then the calculation of the size of the

system was directly requested without generating explanations. In Project 1, certain values were calculated multiple times without the UCP or CP being computed. Project 2 showed a recurrence of responses similar to those generated for Project 1, including multiple instances of explanations about the CP calculation process. Although the project was briefly introduced, but neither the UCP nor the CP was computed. In Project 3 the responses were the same as in Projects 1 and 2 and had no impact on the CP calculations. Notably, even though the required information was already accessible, ChatGPT kept asking for more information on the project data. The primary goal of computing UCP or CP was never achieved in any of the situations.

2) Phase 02: A number of experiments were carried out in the second phase, which was designed to solve the problems found in Phase 01. Starting a conversation with ChatGPT and carrying out the CP computation step-by-step was one of the observations from phase 01. The importance of referring previous steps was one of the observations of another experiment. The literature was unable to provide any support for this.When referring to earlier steps, we tried to include the keywords "Refer," "Referring," and "Above." Fortunately, the experiment turned out quite well. The current step began by making reference to previous steps. We started using variable names similar to those used in programming, for example, "Global Positioning System", in line with earlier literature. Use, "sum of" instead of symbols like "Σ" to improve ChatGPT's understanding of formulas in simple language.

Upon applying these modified prompts to three distinct projects, Project 1 exhibited positive improvement. It began referring to previous steps, correctly determined the required component data type graphs, and interpreted formulas accurately. However, despite these improvements, the UCP remained uncalculated, and the system continued to request additional details from the client, failing to determine the necessary data for specific calculations.

By referring to the earlier steps, the response for Project 2 was structurally identical to project 1 in Phase 02. Neither the UCP nor the CP was computed. The figures for the IFCI and ITCI were calculated incorrectly even the IFC and ITC were available in the previous phase. The formulas were correctly understood, but ChatGPT became confused when a prompt contained more than one task. The response for project 3 was comparable to the responses of Project 1 and 2, and so had no bearing on the CP computation. It began making reference to the earlier stages, the formulas were correctly understood but ChatGPT became confused when a prompt contained more than one task.

3) Phase 03: The third phase of our research, was designed to address and resolve issues identified in Phase 02 as well as those encountered during Phase 03 itself. Each step was dedicated to a single task, ensuring clarity and precision in execution. Emphasis was placed on providing clear definitions for unfamiliar terms in the prompts. For example, use the definition of the term "Interface" was used to enhance the accuracy of system interface identification. Additionally, a vital improvement involved the deletion of ChatGPT's CP calculation history to prevent unintended references to prior processes.

Upon applying these modifications to three distinct projects, positive outcomes were observed in Project 1. The UCP was successfully calculated, and the system accurately determined the required data from previous steps, as shown in Table II. Similarly, in Project 2, the UCP calculation mirrored the success of Project 1, with each step dedicated to a single task, as evidenced in Table II. Project 3 exhibited parallel success, with Projects 1 and 2, with the UCP was tested in. Since Phase 03 was the only successful phase, each project was tested in four attempts to verify the calculated UCP value. Phase 03 attempts involved calculations of each project's UCP. Table II lists the actual UCP values of each project along with the UCP calculated by ChatGPT in each attempt.

TABLE II: Calculated UCP for each project by ChatGPT.

	UCP				
Project	**Actual Value**	**1st attempt**	**2nd attempt**	**3rd attempt**	**4th attempt**
Project 1 (GPS)	21.5	24.5	20	48	16
Project 2 (TAS)	20.5	30	15	18	23
Project 3 (SDES)	105.5	96	69.5	110	98.5

C. Guidelines for Prompt Development

During our experiment of using ChatGPT to count the CP, we discovered some insightful findings from the observations that guide the development of efficient prompts for the more accurate CP counting. We developed a set of useful guidelines for CP calculation utilizing ChatGPT based on our findings. These guidelines formed the basis for the creation of a complete framework that included a set of prompts designed to facilitate effective and accurate CP calculation in CBS. Unlike previous approaches [19], [22], [41], which utilized a single prompt containing project information, examples, and restrictions for output generation, our experiments offer a conversational structure to perform the CP calculation process step by step. The step-by-step conversation with ChatGPT not only complies with the performance requirements but also helps to reduce confusion caused by multiple task requests being made at once.

Incorporating keywords like "Refer," "Referring," and "Above" facilitated cohesive data linkage between steps, and the deletion of previous conversations in CP counting prevented unintentional model references to prior discussions. The using of plain English for formulas like "sum of" instead of "Σ", use clear definitions for unfamiliar terms (Figure 13), and employing block letters for CP-specific words (Table III) were key strategies for enhancing accuracy and understanding. In the initial step, the project information needs to be entered. A customized prompt to help with the input of the required information while getting around the limitations of ChatGPT [48]. The project's name appears at the beginning of the prompt, serving as a distinctive reference point for later ones. Then,the user is asked to identify the relevant component data, allowing for a targeted and effective input procedure. For

better organization and clarity, a structured approach is used in the description of the system's operations, placing the word "Operation" before each operation. In order to improve the accuracy of the CP computation more, relevant information are consistently added when an operation depends on particular variables. It could be made easier to identify different data types in the system by clearly declaring the data types, which will provide important insights into the workings of software. It is important to define the types of components, interfaces, operations, and parameters before classifying the components and identifying interfaces, operations, and parameters in the project. There are separate prompts for each task. Make sure to perform only one task in each prompt. The number of parameters and operations determines the complexity level; Figure 10 specifies threshold values. Before assessing the complexity level of the interface, refer to the previous steps where the identified operations are available. The conventional CP approach uses a table to determine the complexity level (Figure 1). It is recommended to use the table in Figure 10 facilitate better interpretation of the table (Figure 1) when using ChatGPT. Define what an interface type is before asking ChatGPT to determine the interface type of each interface.

In the conventional CP method, the IFC is calculated using the formula shown in 2; however, ChatGPT can't interpret the formula. We have developed a new prompt for calculating the IFC as outlined in Figure 12. The threshold values should be presented in the format shown in Figure 12. Specifically, ask ChatGPT to refer to the complexity level of each interface identified in the previous step before calculating the IFC. The conventional IFCI calculation has a formula. Due to the need for a clearer interpretation, we have developed a new way to express the formula as mentioned in Figure 13.

Determining how many interactions are carried out by each activity is crucial. The number of interactions that each interface performed determines the IF. As in other prompts, a simplified version of the IF calculation formula has been developed to provide a better understanding of the IF calculation (Figure 15). Before calculating the IF, specifically ask to refer to the previous step, in which the number of interactions performed by each interface was already identified. The identification of data types in the system to calculate the CM of each data type is a crucial task. The data types are identified using the component data type graph. It is important to specifically ask to determine the component data type graph first and then identify the data types in the system.

As we did in other prompts, we developed a simplified version of the CM calculation formula by removing the symbols. One important aspect here is that we provide an example of how to calculate the CM of a given data type (Figure 17). This approach improves the accuracy and reliability of the calculated CM of a given data type. The IF of each operation in each interface and the CM of each data type used in the interactions carried out by each operation in each interface are utilized for calculating the ITC. The simplified version of the ITC formula was used to provide a better understanding. Especially ask ChatGPT to refer to the IF and CM calculations in previous steps before moving to the ITC calculation. As mentioned, the rest of the prompts follow the same prompt structure. The threshold values are given in a specific format. Before moving to the required task of the prompt, make sure to instruct ChatGPT to refer to the relevant data from the previous steps. Following these recommendations enabled us the creation of a framework for CP calculation which consists of a set of prompts to fully utilize ChatGPT for more accurate size estimations.

TABLE III: CP specific words which needs to be started with block letters.

Interface
Operation
Parameter
Component
Data Type
Interaction
Complexity

$$IFC = \Sigma_{j=1}^{2}\Sigma_{k=1}^{3}(I_{jk} \times W_{jk}) \quad (2)$$

IV. Results and Discussion

A. Introduction to the Developed Framework

This section introduces a collection of prompts that perform step-by-step process for counting CP of GPS Navigation System using ChatGPT. The CP calculation details of the GPS Navigation system is already documented [2]. We are leveraging the standard CP counting process for the GPS Navigation System from the literature. Referring to this example application of proposed guidelines, users can perform their CP calculation for any CBS. To systematically perform this complex task, we use the guidelines proposed earlier to craft step-by-step prompts tailored for one selected CBS (GPS Navigation System). These prompts follow the guidelines prepared using the observations gained from our experiments with ChatGPT.

First, the project scenario is fed to ChatGPT as shown in Figure 5. Then using the prompt given in Figure 6, the components are classified. This is followed by the identification of interfaces within each component using a prompt as shown in Figure 7. The prompt in Figure 8 shows how to evaluate the operations associated with these interfaces. Subsequently, the parameters for each operation are identified using the prompt in Figure 9. Each interface's complexity level is measured using the instruction in Figure 10, whereas the interface types are identified using the prompt in Figure 11. IFC for each interface is then calculated using the prompt in Figure 12 and the IFCI for each component is calculated following the prompts in Figure 13.

Next the No and Ni for each interaction are calculated by the prompt of Figure 14. Further, the IF is determined by the prompt in Figure 15. Data types in the system are identified by the prompt in Figure 16. The prompt in Figure 17 is used to compute the CM for each type of data. ITC for each interface is calculated based on the prompt in Figure 18, and the ITCI for each component is determined in Figure 19. Consequently, the complexity level of each component is determined using the prompt in Figure 20. Finally, the prompt in Figure 21

is utilized to calculate the UCP of CBS. By considering the criticism on adjusting the UCP into CP [49], we decided to stop the CP calculation up to UCP.

The Global Positioning System has mainly 4 Components. They are user dialog, GPSR controller, Navigator and GIS connection. User dialog has Interfaces for input and output. The GPSR controller Component Interfaces with Global Positioning System Receiver (GPSR) and provides position coordinates whereas the GIS connection component communicates with Geographic Information Server (GIS) and delivers the maps that contain the relevant coordinates. There is an Operation to clear output, an Operation to load the map, an Operation to display current location based on the given point Parameter, an Operation to display route travelled based on a given point Parameter, an Operation to display distance travelled based on distance Parameter and an Operation to display the status. There is an Operation to get the position details. There is one Operation to get map for a given point Parameter and destination Parameter. There is on Operation to get position details. There are Operations to start, stop, navigate and to set the destination Based on the destination Parameter. Based on the connection string the connection is made with GIS System and has an Operation to disconnect.

Understand the context of the system and its architecture, how data flows and how the interactions happen. No need to generate a response.

Fig. 5: The prompt to input the system scenario to the ChatGPT

User Components (UC) stands for components that are primarily used for user interactions of a CBSS. Service Components (SC) Generally provide common services such as database access, messaging, communications, transactions and system level integration services in a Component Based Software System (CBSS). Domain Components (DC) Designated for components that mainly implement domain functions in a CBSS. In the case where a component qualifies to be classified into more than one type, it is classified into the most dominant type based on its usage in the CBSS. Data Types are Not Categorize into any Component Type.

Refer to the Global Positioning System Described Above. Identify to which Exact Component Type the above-mentioned components are belongs.

Fig. 6: The prompt for component classification

Interface is Fundamental to a component which characterizes the functionality provided. The interface defines the services provided by a component and acts as a basis for its use and implementation. An interface comprises a set of operations that act as access points for an interaction with the outside environment. Components are not Interfaces. One Component can have one or many Interfaces. Identify the Interfaces in the Global Positioning System.

Fig. 7: The prompt for interface identification

An Interface comprises a set of operations that act as access point for an interaction with the outside environment. An Interface is simply a collection of operations and only provides a description of them. An operation specifies how inputs, outputs and a component's state relate and the effect of calling the operations on a relationship. Identify the Operations in each Interface Identified above.

Fig. 8: The prompt to identify the operations

A parameter is a named variable passed into an operation. Parameter variables are used to import arguments into Operations. Refer to the above-mentioned Information of Global Positioning System. Determine the Parameters in each Interface.

Fig. 9: The prompt for parameter identification

On the basis of these Number of Operations (NO) and Number of Parameters (NP) values, the complexity level of each interface is then evaluated. Complexity level can be Low, Average or High.

Interface Complexity Levels are, If NO 1 - NP between 1 and 19 = Low, NO 1 – NP between 20 and 50 = Low, NO 1 – NP is More than 51= Average, NO between 2 and 5 – NP between 1 and 19 = Low, NO between 2 and 5 – NP between 20 and 50 = Average, NO between 2 and 5 – NP is More than 51 = High, NO is More than 6 – NP between 1 and 19 = Average, NO is More than 6 – NP between 20 and 50 = High, NO is More than 6 – NP is More than 51= High.

Refer to the above-mentioned Information of the Global Positioning System. Refer to the Number of Operations (NO) and Number of Parameters (NP) Identified above. Evaluate the Complexity Level of each Interface.

Fig. 10: The prompt to identify the complexity level of each interface

There are 2 Interface types. The Interfaces that consist of Operations and exchange data with the rest of the system become candidates for this classification. Among the candidates, the Interfaces whose Operations change attributes of other Interfaces in the data exchange are classified into Internal Logical Files (ILF), whereas all other interfaces are grouped into External Interface Files (EIF).

Refer to the above-mentioned information of the Global Positioning System. Identify the Interface Type of each Interface identified in above.

Fig. 11: The prompt to identify the interface type of each interface

Interface Complexity (IFC) of an Interface = (1 * weight given for the interface)

The weights for interfaces are, Low-ILF = 7, Average-ILF =10, High-ILF =15 Low-EIF = 5, Average-EIF =7, High-EIF =10. Refer to the Complexity Level Calculated for every Interface Above and Interface Types of every Interface. Based on the weights, Calculate the Interface Complexity of each Interface.

Fig. 12: The prompt to Calculate the Interface Complexity (IFC)

Interface Complexity per Interface (IFCI) of a Component = Sum of Interface Complexity (IFC) of each Interface in that Component / Number of Interfaces in that Component.

Calculate the Interface Complexity per Interface (IFCI) in each Component in the Global Positioning System.

Fig. 13: The prompt to Calculate the interface complexity per interface (IFCI) in each component

Refer to Interactions identified above. Calculate the Number of Interactions (No) performed by each Operation and the Number of Interactions (Ni) performed by all Operations of each Interface in the Global Positioning System.

Fig. 14: The prompt to calculate number of interactions

The Interaction Frequency (IF) for an Operation of a given Interface = The Number of Interactions (No) performed by the Operation / The Number of Interactions (Ni) performed by all Operations of the Interface.

Refer to the Number of Interactions (No) performed by each Operation calculated above and the Number of Interactions (Ni) performed by all Operations of each Interface calculated above. Calculate the Interface Frequency (IF) for each Operation of each Interface of a Component in the Global Positioning System.

Fig. 15: The prompt to calculate Interaction Frequency (IF) of each interface

The complexity of the information content involved for each interaction is computed. The information content involved for an Interaction is derived from the signature of the corresponding Operation. To measure the complexity of information content, each Data Type of the information content is represented in a component data type graph. A component data type graph is a hierarchical directed graph of which the nodes represent data entities and arcs represent the relationships among them. The leaf nodes denote primitive data types. Based on the number of hierarchical levels and number of different data types in a component data type graph, the complexity of each data type is computed. The Complexity Measure (CM) of a data type is defined as the sum of complexities of all its member data types calculated recursively from root to all leaf nodes.

Identify the required Data Types for the Global Positioning System. Construct a component data type graph for each Data Types in the Global Positioning System.

Fig. 16: The prompt to construct component data type graph for each data type

The Complexity Measure (CM) of a Data Type = The level where the data type occurs in the hierarchical graph + sum of the complexity of all of its primitive data types, recursively calculated from the root type to all of its primitive data types.

Example:

The user-defined data type 'CourseDetail' is information content.

containing three fields (CourseID, CourseName). CourseID and CourseName are fields of a primitive data type's integer and string, respectively.

CourseDetails: [

{CourseId: String},

{CourseName: String}

]

CM (CourseDetail) = 1 +CM (CourseID, 2)

= 1 +CM (Integer, 2) +CM (String, 2)

= 1 + 2 + 2

= 1 + 2 + 2 = 5

Refer to the Data Types identified above. Calculate the Complexity Measure (CM) of Each data type identified above.

Fig. 17: The prompt to Calculate the Complexity Measure (CM) of each data type

Once the Interaction Frequency (IF) for each operation of each interface and the CM for each data type involved for the interactions performed by each interface of a component are determined, the Interaction Complexity (ITC) of the component is computed.

Interaction Complexity (ITC) = sum of (Interaction Frequency of an Operation of the Operation * Sum of Complexity measure of each Data Type of the information content involved for the interaction performed by each operation of each interface) for each interface in each Operation

Refer to the Interaction Frequency (IF) of each operation of each Interface Calculated above. Refer to the Complexity Measure (CM) of each Data Type Calculated above. Calculate the ITC of each component in the Global Positioning System.

Fig. 18: The prompt to calculate the Interaction Complexity (ITC) of each component

Interaction Complexity per Interaction (ITCI) = ITC (Interaction Complexity) / m (number of interactions that the component performs).

Refer to the Interaction Complexity (ITC) calculated Above. Calculate the Complexity per Interaction (ITCI) for each component of the Global Positioning System.

Fig. 19: The prompt to calculate the Complexity per Interaction (ITCI) for each component

Based on the Interaction Complexity per Interaction (ITCI) and Interface Complexity per Interface (IFCI) of each Component calculated above is used to calculate the component complexity level of each Component.

The Complexity Levels are ITCI between 0 and 1 – IFCI between 0 and 4 = Low, ITCI between 0 and 1 - IFCI between 5 and 8 = Low, ITCI between 0 and 1 - IFCI above 9 = Average, ITCI between 2 and 3 - IFCI between 0 and 4 = Low, ITCI between 2 and 3 - IFCI between 5 and 8 = Average, ITCI between 2 and 3 - IFCI above 9 = High, ITCI above 4 - IFCI between 0 and 4 = Average, ITCI above 4 - IFCI between 5 and 8 = High, ITCI above 4 - IFCI above 9 = High.

Refer to the Interface Complexity per Interface (IFCI) and Interaction Complexity per Interaction (ITCI) calculated above. Measure the Complexity Level of each Component of the Global Positioning System.

Fig. 20: The prompt to determine the complexity level of each component

Unadjusted Component Point (UCP) = sum of (number of components of a component type in a complexity level * weight given for a component type in a complexity level)

The weights for complexity levels are, DC that has Low complexity = 3, DC that has Average Complexity =6, DC that has High Complexity =10. UC that has Low complexity = 4. UC that has Average Complexity =7, UC that has High Complexity =12. SC that has Low complexity = 4.5. SC that has Average Complexity =7. SC that has High Complexity =11.5.

Refer to the Component Types Identified above. Based on the weights and given information Calculate the Unadjusted Component Point (UCP) of the Global Positioning System.

Fig. 21: The prompt to calculate the Unadjusted Component Point (UCP) of the system

B. Discussion

To validate and contextualize the developed framework, we have conducted a comparative analysis using the results we got in phase 03 in experiment section with the UCP counts obtained through the conventional CP counting process on the same three CBS projects documented in [2]. The findings from this comparative analysis not only contribute to the ongoing discourse on the applicability of AI based large language models in software engineering tasks but also offer insights into the alignment between ChatGPT-generated results and conventionally accepted methodologies in the domain of CP counting. We attempted to evaluate the error of the suggested framework. Considering the discussion of adjustment of UCP [49], we specifically focused on the UCP counts derived from the proposed framework in phase 03.

To quantitatively evaluate the performance of our framework, we employed two distinct validation metrics: MMRE and PRED (30%). MMRE, is a critical accuracy metric in which the lower numbers denote better performance. In our validation, our proposed framework gained a commendable 27% MMRE (Table IV), showcasing its effectiveness in minimizing errors and deviating towards an optimal MMRE value of 0. Furthermore, by achieving a PRED (30%) value of 75% (Table IV), our framework surpassed the widely accepted model accuracy of 70%, affirming its capability to provide accurate predictions within a 30% deviation. This result shows the validity of our proposed framework, describing it as a reliable and acceptable model for industry applications.

V. Conclusions and Recommendations

In this paper we address a crucial issue in software development – the high failure rate of software projects, often

TABLE IV: Validation results

Project	UCP						
	Actual value	1st attempt	2nd attempt	3rd attempt	4th attempt	MMRE %	PRED (30) %
01(GPS)	21.5	24.5	20	48	16	43.45	75
02(TAS)	20.5	30	15	18	23	24.38	75
03(SDES)	105.5	96	69.5	110	98.5	13.18	75
Overall MMRE and PRED (30%)						27	75

attributed to the lack of quantification of products, processes, and resources. Recognizing the significance of cost estimation in software development, we emphasize the critical factor of software size in cost estimation. Notably, as experts predict a future for CBS, it becomes imperative to measure CBS size and then cost of software early in the SDLC. Traditional methods like CP are time-consuming, resource-intensive, and reliant on specialized expertise.

In response to these challenges, we developed an innovative framework which consists of a set of prompts based on the results of the experiments we did. The Framework is used to automate the measurement of size of CBS, leveraging ChatGPT to streamline the process and reduce the dependence on expert knowledge, time, and resources typically associated with manual CP counting. ChatGPT has proven the ability to solve problems accurately in many industries including software engineering, when compared to other Generative AI tools like Bard AI. This novel framework seems a significant step forward in the realm of software engineering.

Based on prior studies in related fields and from our own experiments, we first developed an initial framework comprising a set of prompts that might be used to measure the size of CBS. Based on the observations of the experiments we conducted, the initial framework was fine-tuned to improve accuracy. Simultaneously, a set of guidelines were developed to enhance the accuracy of CBS measure. The size of a selected CBSs was then measured using the framework. The size of those selected CBS has already been calculated in the previous study using the manual CP calculation. A comparative analysis was conducted to evaluate the acceptability and validity of the developed framework. We have got the model accuracy beyond the acceptable model accuracy. In certain stages of the CP calculation procedure, we saw fewer differences compared with more similarities.

The disparities observed in the initial prompts highlight the importance of fine-tuning prompts more to enhance the accuracy in ChatGPT's predictions. Drawing on existing literature, it is recognized that fine-tuning prompts can be a strategic approach to improve the accuracy of the outputs. Still the developed framework has not been validated with the real-world software project collected from the industry, even though it is applied to a few CBS projects. As a future direction, we have planned to conduct a validation process using the data collected from the industry and the applicability of the framework into other generative AI tools like Bard AI is to be experimented with. In addition to that, there is a future direction to explore various prompt engineering frameworks, including one-shot, few-shot, and many-shot learning approaches, to enhance the adaptability and accuracy of the developed framework in diverse application scenarios. This research has the potential to significantly change the way of measuring the size of a CBS in the early stages of the SDLC to lead to improved estimation techniques.

REFERENCES

[1] *The Chaos Report - 2022*. The Standish Group International, Inc, 2020. [Online]. Available: https://www.successthroughsafe.com/blog-1/2021/11/13/standish-chaos-report-2021

[2] T. Wijayasiriwardhane and R. Lai, "Component point: A system-level size measure for component-based software systems," vol. 83, dec 2010, pp. 2456–2470. [Online]. Available: https://doi.org/10.1016/j.jss.2010.07.008

[3] J. G. Borade and V. R. Khalkar, "Software project effort and cost estimation techniques," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, pp. 730–739, jan 2013. [Online]. Available: https://www.researchgate.net/publication/313243865

[4] K. Pillai and V. S. S. Nair, "A model for software development effort and cost estimation," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 23, no. 8, pp. 485–497, aug 1997. [Online]. Available: https://doi.org/10.1109/32.624305

[5] A. Ren and Y. Chen, "Cocomo ii based project cost estimation and control," pp. 1293–1299, 2015/12. [Online]. Available: https://doi.org/10.2991/icemaess-15.2016.265

[6] A. J. Albrecht, , and J. E. Gaffney, "Software function, source lines of code, and development effort prediction: A software science validation," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. SE-9, pp. 639–648, nov 1983. [Online]. Available: https://doi.org/10.1109/TSE.1983.235271

[7] B. W. Boehm, "Software engineering economics," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. SE-10, p. 99–150, 1984. [Online]. Available: https://doi.org/10.1007/978-3-642-48354-7_5

[8] M. Shepperd and C. Schofield, "Estimating software project effort using analogies," *IEEE TRANSACTIONS ON SOFTWARE ENGINEERING*, vol. 23, no. 11, pp. 736–743, nov 1997. [Online]. Available: https://doi.org/10.1109/32.637387

[9] E. Mendes and S. Counsell, "Web development effort estimation using analogy," aug 2000, pp. 203–212. [Online]. Available: https://doi.org/10.1109/ASWEC.2000.844577

[10] G. Wittig and G. Finnie, "Estimating software development effort with connectionist models," *Information and Software Technology*, vol. 39, pp. 469–476, 1997. [Online]. Available: https://doi.org/10.1016/S0950-5849(97)00004-9

[11] S. Kanmani, J. Kathiravan, S. S. Kumar, and M. Shanmugam, "Neural network based effort estimation using class points for oo systems," pp. 261–266, mar 2007. [Online]. Available: https://doi.org/10.1109/ICCTA.2007.89

[12] S. Mondal, S. Das, and V. G. Vrana, "How to bell the cat? a theoretical review of generative artificial intelligence towards digital disruption in all walks of life," *Technologies*, vol. 11, p. 44, mar 2023. [Online]. Available: https://doi.org/10.3390/technologies11020044

[13] P. Vitharana, F. Zahedi, and H. Jain, "Design, retrieval, and assembly in component-based software development," vol. 46, pp. 97–102, nov 2003. [Online]. Available: https://doi.org/10.1145/948383.948387

[14] T. Ravichandran and M. A. Rothenberger, "Software reuse strategies and component markets," *Commun. ACM*, vol. 46, no. 8, p. 109–114, aug 2003. [Online]. Available: https://doi.org/10.1145/859670.859678

[15] S. Mahmood and R. Lai, "A complexity measure for uml component-based system specification," *Software - Practice and Experience*, vol. 38, pp. 117–134, feb 2008. [Online]. Available: https://doi.org/10.1002/spe.769

[16] J. Frohm, V. Lindström, M. Winroth, and J. Stahre, "The industry's view on automation in manufacturing," *IFAC Proceedings Volumes*, vol. 39, no. 4, pp. 453–458, 2006, 9th IFAC Symposium on Automated Systems Based on Human Skill and Knowledge. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1474667015330925

[17] A. Biswas and P. K. Dutta, "Novel approach of automation to risk management: The reduction in human errors." Springer Science and Business Media Deutschland GmbH, 2021, pp. 683–696. [Online]. Available: https://doi.org/10.1007/978-3-030-49795-8_65

[18] M. Aluga, "Application of chatgpt in civil engineering," *East African Journal of Engineering*, vol. 6, pp. 104–112, jun 2023. [Online]. Available: https://journals.eanso.org/index.php/eaje/article/view/1272

[19] H.-G. Fill and F. Muff, "Visualization in the era of artificial intelligence: Experiments for creating structural visualizations by prompting large language models," may 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2305.03380

[20] S. S. Gill and R. Kaur, "Chatgpt: Vision and challenges," *Internet of Things and Cyber-Physical Systems*, vol. 3, pp. 262–271, 2023. [Online]. Available: https://doi.org/10.1016/j.iotcps.2023.05.004

[21] N. Nascimento, P. Alencar, and D. Cowan, "Comparing software developers with chatgpt: An empirical investigation," may 2023. [Online]. Available: http://arxiv.org/abs/2305.11837

[22] D. Sobania, M. Briesch, C. Hanna, and J. Petke, "An analysis of the automatic bug fixing performance of chatgpt," in *2023 IEEE/ACM International Workshop on Automated Program Repair (APR)*, jan 2023, pp. 23–30. [Online]. Available: https://doi.org/10.1109/APR59189.2023.00012

[23] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," feb 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2302.11382

[24] P. G. Armour, "The business of software a measure of control," *Communications of the ACM*, vol. 55, pp. 26–28, jun 2012. [Online]. Available: https://doi.org/10.1145/2184319.2184329

[25] A. L. Al-saleem and A. Y. Hammo, "Software size estimation: A survey," vol. 04, no. 9, pp. 62–70, 2022. [Online]. Available: https://doi.org/10.47577/technium.v4i9.7251

[26] C. Kemerer and B. Porter, "Improving the reliability of function point measurement: an empirical study," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, pp. 1011–1024, nov 1992. [Online]. Available: https://doi.org/10.1109/32.177370

[27] C. Symons, "Function point analysis: difficulties and improvements," *IEEE Transactions on Software Engineering*, vol. 14, no. 1, pp. 2–11, jan 1988. [Online]. Available: https://doi.org/10.1109/32.4618

[28] R. D. Banker, R. J. Kauffman, C. Wright, and D. Zweig, "Automating output size and reuse metrics in a repository-based computer-aided software engineering (case) environment," *IEEE Transactions on Software Engineering*, vol. 20, pp. 169–187, mar 1994. [Online]. Available: https://doi.org/10.1109/32.268919

[29] G. Costagliola, F. Ferrucci, G. Tortora, and G. Vitiello, "Class point: an approach for the size estimation of object-oriented systems," pp. 52–74, jan 2005. [Online]. Available: https://doi.org/10.1109/TSE.2005.5

[30] *The IFPUG Function Point Counting Method*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 323–363. [Online]. Available: https://doi.org/10.1007/978-3-540-68188-5_12

[31] A. Cechich, M. Piattini, and A. Vallecillo, *Assessing Component-Based Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 1–20. [Online]. Available: https://doi.org/10.1007/978-3-540-45064-1_1

[32] S. Sedigh-Ali, A. Ghafoor, and R. Paul, "Software engineering metrics for cots-based systems," *Computer*, vol. 34, no. 5, pp. 44–50, may 2001. [Online]. Available: https://doi.org/10.1109/2.920611

[33] N. S. Gill and P. S. Grover, "Component-based measurement: few useful guidelines," *SIGSOFT Softw. Eng. Notes*, vol. 28, no. 6, p. 4, nov 2003. [Online]. Available: https://doi.org/10.1145/966221.966237

[34] T. M. Fagbola and S. C. Thakur, *Towards the Development of Artificial Intelligence-based Systems: Human-Centered Functional Requirements and Open Problems*, feb 2019. [Online]. Available: https://doi.org/10.1109/ICIIBMS46890.2019.8991505

[35] B. D. Lund and T. Wang, "Chatting about chatgpt: how may ai and gpt impact academia and libraries?" *Library Hi Tech News*, feb 2023. [Online]. Available: https://doi.org/10.1108/LHTN-01-2023-0009

[36] Ömer Aydın and Ömer Aydin, "Google bard generated literature review: Metaverse," *Journal of AI*, vol. 7, pp. 1–14, aug 2023. [Online]. Available: https://www.researchgate.net/publication/370933947

[37] S. R. Ali, T. D. Dobbs, H. A. Hutchings, and I. S. Whitaker, "Using chatgpt to write patient clinic letters," *The Lancet Digital Health*, mar 2023. [Online]. Available: https://doi.org/10.1016/S2589-7500(23)00048-1

[38] A. Bahrini, M. Khamoshifar, H. Abbasimehr, R. J. Riggs, M. Esmaeili, R. M. Majdabadkohne, and M. Pasehvar, "Chatgpt: Applications, opportunities, and threats," in *2023 Systems and Information Engineering Design Symposium (SIEDS)*, apr 2023, pp. 274–279. [Online]. Available: https://doi.org/10.1109/SIEDS58326.2023.10137850

[39] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design," mar 2023. [Online]. Available: http://arxiv.org/abs/2303.07839

[40] P. P. Ray, "Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope," pp. 121–154, jan 2023. [Online]. Available: https://doi.org/10.1016/j.iotcps.2023.04.003

[41] H.-G. Fill, P. Fettke, and J. Köpke, "Conceptual modeling and large language models: Impressions from first experiments with chatgpt," vol. 18, 2023. [Online]. Available: https://doi.org/10.18417/emisa.18.3

[42] R. Bhatnagar and M. Ghose, "Early stage software development effort estimations - mamdani fis vs neural network models," *Computer Science Conference Proceedings*, vol. 2, pp. 377–384, jan 2012. [Online]. Available: https://doi.org/10.5121/csit.2012.2135

[43] D. Port and M. Korte, "Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research," 2008, pp. 51–60. [Online]. Available: https://doi.org/10.1145/1414004.1414015

[44] I. Attarzadeh and S. H. Ow, "Proposing a new high performance model for software cost estimation," vol. 2, dec 2009, pp. 112–116. [Online]. Available: https://doi.org/10.1109/ICCEE.2009.97

[45] O. Attarzadeh, Iman and S. Hock, "Proposing a new software cost estimation model based on artificial neural networks," in *2010 2nd International Conference on Computer Engineering and Technology*, vol. 3, jun 2010, pp. 487–491. [Online]. Available: https://doi.org/10.1109/ICCET.2010.5485840

[46] S. A. Shahwaiz, A. A. Malik, and N. Sabahat, "A parametric effort estimation model for mobile apps," in *2016 19th International Multi-Topic Conference (INMIC)*, 2016, pp. 1–6. [Online]. Available: https://doi.org/10.1109/INMIC.2016.7840114

[47] N. Salman, A. H. Dogru, and A. Doğru, "Design effort estimation using complexity metrics," *Journal of Integrated Design and Process Science*, vol. 8, pp. 83–88, aug 2004. [Online]. Available: https://www.researchgate.net/publication/262322343

[48] B. Farnham, S. Tokyo, B. Boston, F. Sebastopol, and T. Beijing, *What Are ChatGPT and Its Friends? Opportunities, Costs, and Risks for Large Language Models*, 2023. [Online]. Available: https://www.oreilly.com/radar/what-are-chatgpt-and-its-friends/

[49] C. Lokan, "An empirical analysis of function point adjustment factors," *Information and Software Technology*, vol. 42, no. 9, pp. 649–659, 2000. [Online]. Available: https://doi.org/10.1016/S0950-5849(00)00108-7