# Towards Human-Bot Collaborative Software Architecting with ChatGPT

Aakash Ahmad[1*], Muhammad Waseem[2], Peng Liang[3], Mahdi Fehmideh[4], Mst Shamima Aktar[3], Tommi Mikkonen[2]
[1]School of Computing and Communications, Lancaster University Leipzig, Leipzig, Germany
[2]Faculty of Information Technology, University of Jyväskylä, Jyväskylä, Finland
[3]School of Computer Science, Wuhan University, Wuhan, China
[4]School of Business, University of Southern Queensland, Queensland, Australia
a.ahmad13@lancaster.ac.uk, mwaseem@jyu.fi, liangp@whu.edu.cn, mahdi.fahmideh@usq.edu.au, shamima@whu.edu.cn, tommi.j.mikkonen@jyu.fi

## ABSTRACT

Architecting software-intensive systems can be a complex process. It deals with the daunting tasks of unifying stakeholders' perspectives, designers' intellect, tool-based automation, pattern-driven reuse, and so on, to sketch a blueprint that guides software implementation and evaluation. Despite its benefits, architecture-centric software engineering (ACSE) inherits a multitude of challenges. ACSE challenges could stem from a lack of standardized processes, socio-technical limitations, and scarcity of human expertise etc. that can impede the development of existing and emergent classes of software (e.g., IoTs, blockchain, quantum systems). Software Development Bots (DevBots) trained on large language models can help synergise architects' knowledge with artificially intelligent decision support to enable rapid architecting in a human-bot collaborative ACSE. An emerging solution to enable this collaboration is ChatGPT, a disruptive technology not primarily introduced for software engineering, but is capable of articulating and refining architectural artifacts based on natural language processing. We detail a case study that involves collaboration between a novice software architect and ChatGPT for architectural analysis, synthesis, and evaluation of a services-driven software application. Preliminary results indicate that ChatGPT can mimic an architect's role to support and often lead ACSE, however; it requires human oversight and decision support for collaborative architecting. Future research focuses on harnessing empirical evidence about architects' productivity and exploring socio-technical aspects of architecting with ChatGPT to tackle emerging and futuristic challenges of ACSE.

## KEYWORDS

Software Architecture, ChatGPT, Large Language Models, DevBots

**ACM Reference Format:**
Aakash Ahmad[1*], Muhammad Waseem[2], Peng Liang[3], Mahdi Fehmideh[4], Mst Shamima Aktar[3], Tommi Mikkonen[2]. 2023. Towards Human-Bot Collaborative Software Architecting with ChatGPT. In *Proceedings of ACM Conference (XXXX)*. ACM, New York, NY, USA, 7 pages. https://doi.org/XXXXXXX.XXXXXXX

*XXXX, XX-XX Month 2023, XX, XX*
© 2023 Association for Computing Machinery.
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

## 1 INTRODUCTION

Architecture of software-intensive systems enables architects to specify structural composition, express behavioural constraints, and rationalise design decisions - hiding implementation complexities with architectural components - to sketch a blue-print for software implementation [12]. Architecture-centric Software Engineering (ACSE) aims to exploit architectural knowledge (e.g., tactics and patterns), architectural languages, tools, and architects' decisions (human intellect) etc. to create a model that drives the implementation, validation, and maintenance phases of software systems [9]. In recent years, ACSE has been applied to investigate the role of architecture in engineering complex and emergent classes of software (blockchains, quantum systems etc.) [1] and it has been proven as useful to systematise software development in an industrial context [9]. Despite its potential, ACSE entails a multitude of challenges including but not limited to mapping stakeholders' perspectives to architectural requirements, managing architectural drift, erosion, and technical debts, or lack of automation and architects' expertise in developing complex and emergent classes of software [1, 12]. In such context, software engineers may enter a phase referred to as the *'lonesome architect'* who requires non-intrusive support rooted in processes and tools to address the challenges of ACSE by reusing knowledge and exploiting decision support in the process [10].

**Context and motivation**: The process to architect software applications and services (a.k.a., 'architecting process') unifies a number of architecting activities that support an incremental, process-centric, and systematic approach to apply ACSE in software development endeavours [1, 9]. Empiricism remains fundamental to deriving and/or utilising architecting processes that can support activities, such as analysis, synthesis, and evaluation etc. of software architetures [10]. To enrich the architecting process and empower the role of architects, research and development has focused on incorporating patterns and styles (knowledge), recommender systems (intelligence), and distributed architecting (collaboration) in ACSE process. The role of artificial intelligence (AI) in software engineering (SE) is an active area of research that aims to synergise solutions of AI and practices of SE to instill intelligence in the processes and tools for software development [4, 18]. From an ACSE perspective, research on AI generally aims to develop decision support systems or development bots that can assist architects with recommendations about design decisions, selection of patterns and styles, or predict points of architectural failure and degradation [8, 16]. Currently, there is no research that proposes innovative solutions that can enrich the architecting process with AI to enable collaborative architecting. Collaborative architecting can synergise architects' knowledge as human intellect and bot's capability as an intelligent agent who can lead the architecting process based on human conversation and supervision. Such collaboration can allow architects to delegate their architecting tasks to the bot, supervise the bot via dialog in natural language(s) to achieve automation, and relieve architects from undertaking tedious tasks in ACSE.

**Objective of the study**: Chat Generative Pre-trained Transformer (ChatGPT) has emerged as a disruptive technology, representing an unprecedented example of a bot, that can engage with humans in context-preserved conversations to produce well-articulated responses to complex queries [3, 14]. ChatGPT is not specifically developed to address software engineering challenges, however; it is well capable of generating versatile textual specifications including architectural requirements, UML scripts, source code libraries, and test cases [11, 15]. Recently published research has started to explore the role of ChatGPT in engineering education, software testing, and source code generation [14, 15]. Considering ACSE that can benefit from intelligent and automated architecting, driven by architects' conversational dialogs and feedback, there is no research to investigate the role that ChatGPT can play as a DevBot in architecting process. To this end, our study focused on a preliminary investigation to understand *if ChatGPT can process an architecture story (scenario(s)) conversed to it by an architect and undertake architecting activities to analyse, synthesise, and evaluate software architecture in a human-bot collaborative architecting.*

**Contributions**: We followed a process-centric approach [9] and adopted scenario-based method [6] for ChatGPT-enabled architectural analysis, synthesis, and evaluation of a microservices-driven software. Preliminary results demonstrate ChatGPT's capabilities that include but are not limited to processing an architecture story (conversed to it by an architect) for articulating architectural requirements, specifying models, recommending and applying architectural tactics and patterns, and developing scenarios for architecture evaluation. Primary contributions of this study are to:

- Investigate the potential for human-bot collaborative architecting, synergizing ChatGPT's outputs and architects' decisions, to automate ACSE with a preliminary case study.
- Identify the potential and perils of ChatGPT assisted ACSE to pinpoint issues concerning ethics, governance, and socio-technical constraints of collaborative architecting.
- Establish foundations for empirically-grounded evidence about ChatGPT's capabilities and architects' productivity in collaborative architecting (ongoing and future work).

The results of this study can help academic researchers to formulate new hypotheses about the role of ChatGPT in ACSE and investigate human-bot collaborative architecting of emergent and futuristic software. Practitioners can follow the presented guidelines to experiment with delegating their tedious tasks of ACSE to ChatGPT.

## 2 RESEARCH CONTEXT AND METHOD

We next contextualize some core concepts (Section 2.1, Figure 1) and discuss the research method (Section 2.2, Figure 2). Terms and concepts introduced here will be used throughout the paper.

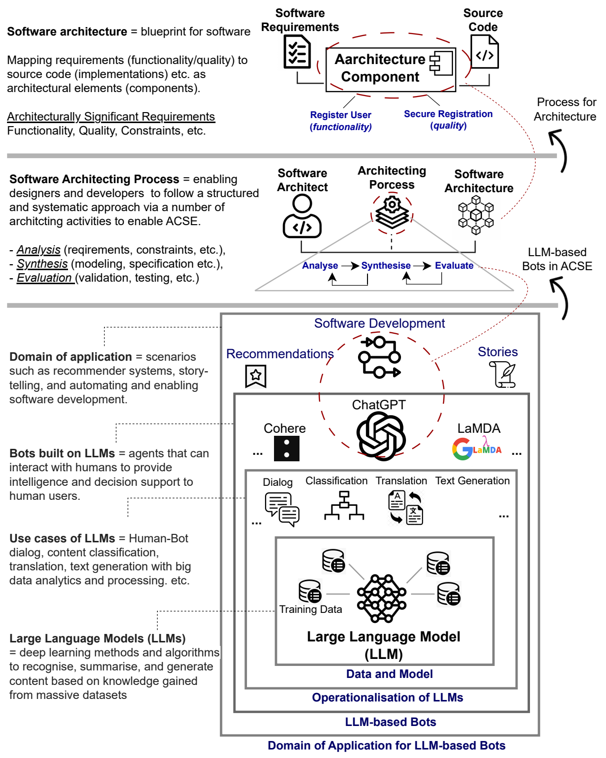

**Software architecture** = blueprint for software
Mapping requirements (functionality/quality) to source code (implementations) etc. as architectural elements (components).
Architecturally Significant Requirements
Functionality, Quality, Constraints, etc.
Software Requirements
Aarchitecture Component
Source Code
Register User (functionality)
Secure Registration (quality)
Process for Architecture
**Software Architecting Process** = enabling designers and developers to follow a structured and systematic approach via a number of architcting activities to enable ACSE.
- Analysis (reqirements, constraints, etc.),
- Synthesis (modeling, specification etc.),
- Evaluation (validation, testing, etc.)
Software Architect
Architecting Porcess
Software Architecture
Analyse
Synthesise
Evaluate
LLM-based Bots in ACSE
**Domain of application** = scenarios such as recommender systems, story-telling, and automating and enabling software development.
Software Development
Recommendations
Stories
**Bots built on LLMs** = agents that can interact with humans to provide intelligence and decision support to human users.
ChatGPT
Cohere
LaMDA
**Use cases of LLMs** = Human-Bot dialog, content classification, translation, text generation with big data analytics and processing. etc.
Dialog
Classification
Translation
Text Generation
**Large Language Models (LLMs)** = deep learning methods and algorithms to recognise, summarise, and generate content based on knowledge gained from massive datasets
Training Data
Large Language Model (LLM)
Data and Model
Operationalisation of LLMs
LLM-based Bots
Domain of Application for LLM-based Bots

**Figure 1: Context: LLMs, DevBots, Process, and Architecture**

### 2.1 Human-Bot Collaborative Architecting

**Software Architecture** as described in the ISO/IEC/IEEE 42010:2011 standard, aims to abstract complexities rooted in source code-based implementations with architectural components and connectors that represent a blueprint of software applications, services, and systems to be developed [12]. Architecture-centric approaches have proven to be useful in academic solutions as well as in industrial projects by lending architectural knowledge, such as patterns, styles, languages, and frameworks, to design and develop software effectively and efficiently [10]. To enable software designers and architects with a systematic and incremental design of software architectures, there is a need for **architecting process** - also referred to as the process for architecting software [1, 9]. Architecting process can have a number of fine-grained architecting activities that support a separation of architectural concerns in ACSE. For example, the architecting process reported in [9] and illustrated in Figure 1 is derived from five industrial projects and incorporates three architecting activities namely *architectural analysis*, *architectural synthesis*, and *architectural evaluation*. For instance, the architectural evaluation activity in the process focuses on scenarios to evaluate the designed architecture [6]. In the architecting process, an architect can extract and document the requirements that express the required functionality and desired quality of the software, referred to as Architecturally Significant Requirements (ASRs). ASRs need to be mapped to source code implementations via an architectural model that can be visualized or textually specified using architectural languages, such as the Unified Modeling Language (UML) or Architectural Description Languages (ADLs) [13]. Architecture models that reflect the ASRs need to be evaluated using an architecture evaluation method, such as Software Architecture Analysis Method (SAAM) or Architecture Tradeoff Analysis Method (ATAM) [6].

**Software Development Bots** (DevBots) represent conversational agents or recommender systems, driven by AI, to assist software engineers by offering certain degree of automation and/or inducing intelligence in software engineering process [16]. From the software architecting perspective, the role of AI in general and DevBots to be specific is limited to bots answering questions or providing recommendations about architectural erosion and maintenance [8]. There is no research that investigates or any solution that demonstrates an architecting process by incorporating DevBots to enable human-bot collaborative architecting of software systems. Such a collaboration can enrich the architecting process that goes beyond questions & answers and recommendations, and synergizes architects' intellect (human rationale) and bot's intelligence (automated architecting process) in ACSE. Collaborative architecting can empower novice designers or architects, who lack experience or professional expertise to specify their requirements in natural language and DevBots can translate them into ASRs, architectural models, and evaluation scenarios. As illustrated in Figure 1, the emergence of ChatGPT as a conversational bot, based on large language models (LLM), can dialog with the architect to lead the creation of architectural artifacts with human supervision.

## 2.2 Research Method

We now present the overall methodology for the research, comprising of three main phases, as illustrated in Figure 2.

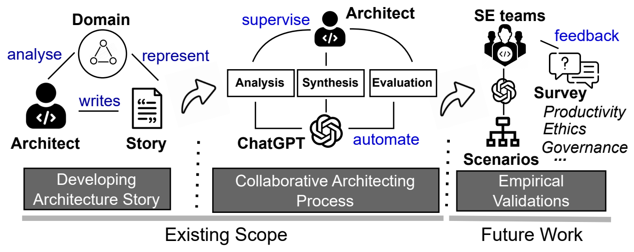


**Figure 2: Overview of the Research Method**

**Phase 1 - Developing the Architecture Story**: Software architecture story refers to a textual narration of the envisaged solution, i.e., software to be developed by expressing the core functionality, desired quality (i.e., ASRs) and any constraints in a natural language. The story is developed based on analyzing software domain that represents an operational context of the system or collection of scenarios operationalised via a software solution. The architect can analyze the domain and identify scenarios to write an architecture story that acts as a foundation for the architecting process. The architecture story is fed to ChatGPT via a prompt as a pre-process to collaborative architecting.

**Phase 2 - Enabling Collaborative Architecting** is based on three activities adopted from [9], detailed below.

- *Architectural analysis* is driven by architecture story fed to ChatGPT for articulating the ASRs via (i) automatically generated and recommended requirements (by ChatGPT), or (ii) manual specification of the requirements (by the architect), or (iii) a continuous dialog between ChatGPT and the architect to refine (add/remove/update) the requirements.
- *Architectural synthesis* consolidates the ASRs to create an architecture model or representation that can act as a point of reference, visualizing the structural (de-)composition and runtime scenarios for the software. We preferred UML for architectural synthesis due to a number of factors, such as available documentation, ease of use, diversity of diagrams, tool support, and wide-scale adoption as a language to represent software systems [13]. During synthesis we also incorporated reuse knowledge and best practices in the form of tactics and patterns to refine the architecture.
- *Architectural evaluation* evaluates the synthesized architecture against ASRs based on scenarios from the architectural story. Architectural evaluation is conducted incrementally for full or partial validation of the architecture or its parts based on use cases or scenarios from ASRs. We used the Software Architecture Analysis Method (SAAM) to supervise ChatGPT for evaluating the architecture [6].

**Phase 3 - Conducting the Empirical Validations** complements the initial two phases with empirical validations of collaborative architecting as an extension of this study, outlining future work. The existing scope aims to explore and present the role of ChatGPT in human-bot collaborative software architecting (in Section 3). Future work on empirically grounded guidelines to understand a multitude of socio-technical issues associated with ChatGPT-driven collaborative architecting is discussed later (in Section 5).

## 3 CASE STUDY ON COLLABORATIVE ARCHITECTING

This section details the process of collaborative architecting demonstrated with a case study for scenario-based exemplification and illustrations (see Figure 3). The **case study** detailed in [2] aims to develop a software application named CampusBike that can be used via a browser or as an app, allowing campus visitors to 'register', 'view available bikes', 'reserve a bike', 'make payments', and 'view usage reports' etc. for eco-friendly mobility in and around the campus. The **architect** has a working knowledge of software design (UML, patterns etc.) and implementation (programming and scripting languages) and is considered a motivated novice engineer with the responsibility to design and develop CampusBike software.

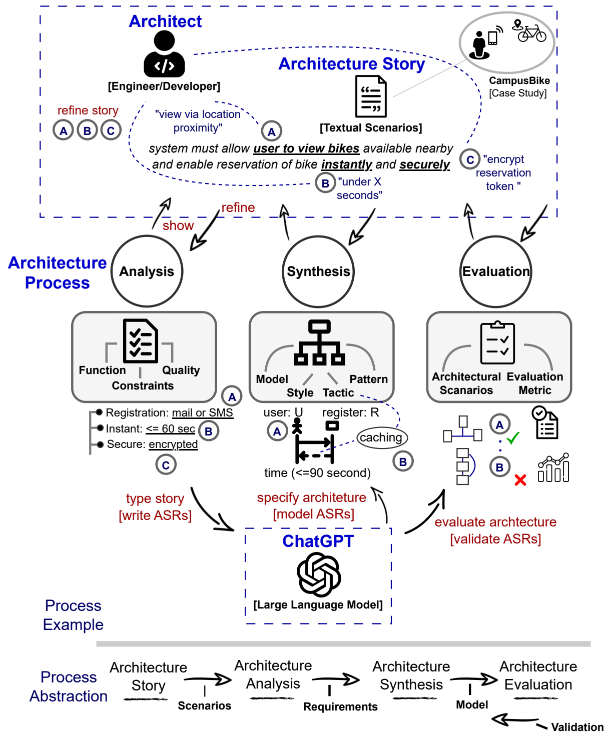
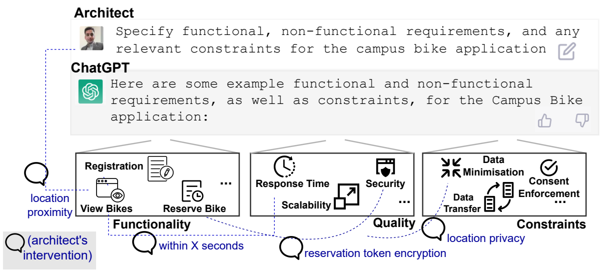


**Figure 3: Overview of the Human-Bot Collaborative Architecting Process with ChatGPT**

> **Snippet of Architecture Story**
>
> "… as a step towards maintaining a 'Green Campus' - minimising the carbon footprint, congestion, and noise created by vehicles, University's administration has decided to introduce a bike service where campus visitors can avail of a pay-per-use bicycle facility on an hourly or daily basis for enhanced mobility in and around the campus. Potential bikers can register and view available bikes in their proximity (let us say within 500 meters) and reserve them for a specific time after payment. To facilitate this service, the administration needs a software application called 'CampusBike' that is available on Web and mobile devices for potential bikers …"
>
> **Scenario Example**: View available bikes (using location proximity), reserve a bike for a specific time (pay-and-reserve).

## 3.1 Formulating the Architecture Story

Architecture story refers to a textual narration of the envisaged solution, i.e., software to be developed by expressing the core functionality and any constraints narrated in a natural language. As per the methodological details in Figure 2, the story is developed based on analysing software domain that represents an operational context of the system or collection of scenarios operationalized via a software solution. The architect can analyse the domain and identify any scenarios to write an architecture story, fed to ChatGPT, that sets the foundation for architectural analysis activity in the process. Detailed architecture story is available at [2], with its sample snippet and two scenarios highlighted below.


Architect

Specify functional, non-functional requirements, and any relevant constraints for the campus bike application

ChatGPT

Here are some example functional and non-functional requirements, as well as constraints, for the Campus Bike application:

**Figure 4: Formulating and Refining the Requirements**

## 3.2 Architectural Analysis

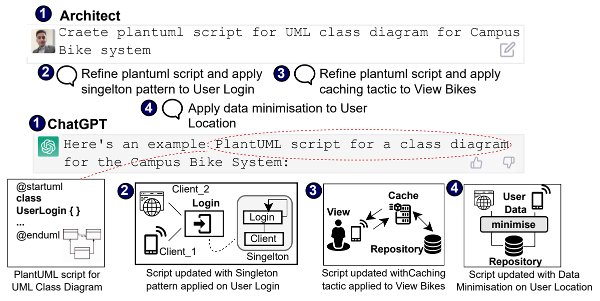
Once the architecture story is fed to ChatGPT, during architectural analysis, the focus is to specify ASRs as required functionality (e.g., view available bikes) and desired quality (e.g., response time < N) along with any constraints (e.g., compliance with relevant data security policies) of CampusBike software. ChatGPT is capable of outlining the ASRs or any necessary constraints if queried by the architect. However, as per the case study, ChatGPT expressed the ASRs and constraints that were refined (add, remove, and modify any requirements) by the architect. For example, the 'Reserve Bike' requirement articulated by ChatGPT read as: '*… system must allow user to view bikes available nearby and enable reservation of the bike instantly and securely*'. The architect refined the requirements:

> **Architect's Refinements**
> Functionality: View Bike - via location proximity
> Quality: Instantly - within 90 seconds, Securely - encrypt reservation token
> Constraint: apply data minimization on registration data (GDPR constraint)

After narrating the architecture story, Figure 4 shows architects' query and ChatGPT's response (human-bot collaboration) to specify the functionality, quality, and constraints, collectively referred to as the ASRs. ASRs are iteratively refined via a dialog between the two to produce a final list presented here [2].

## 3.3 Architectural Synthesis

The ASRs are synthesized into an architectural model that can be expressed with an architectural (modeling) language, like UML or other architectural languages [13]. We used UML class and component diagrams to create the architecture model, specifically; component diagrams to represent the overall architecture, and class diagram for fine-grained representation of the architectural design. During synthesis, we refined the UML class diagram with the application of singleton pattern to 'UserLogin' class to restrict a single login session across the devices. We applied the caching tactic on 'ViewBikes' and data minimization constraint on 'User Location'. Figure 5 shows the architect's instruction for ChatGPT's to create the script for UML class diagram. Additional dialog between the two enabled application of singleton pattern, caching tactic, and data minimisation constraint on class diagram, presented in [2].



Architect

```
Craete plantuml script for UML class diagram for Campus
Bike system
```

Refine plantuml script and apply singelton pattern to User Login

Refine plantuml script and apply caching tactic to View Bikes

Apply data minimisation to User Location

ChatGPT

```
Here's an example PlantUML script for a class diagram
for the Campus Bike System:
```

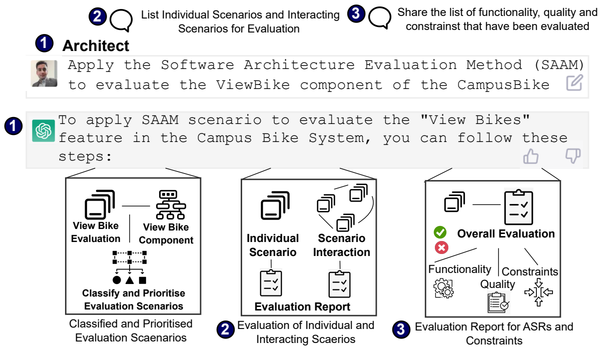
```
@startuml
class
UserLogin { }
...
@enduml
```

1
2
3
4
Client_2
Login
Login
Client
Singelton
Client_1
Cache
View
Repository
User Data
minimise
Repository
PlantUML script for UML Class Diagram
Script updated with Singleton pattern applied on User Login
Script updated withCaching tactic applied to View Bikes
Script updated with Data Minimisation on User Location

Figure 5: Modeling and Refining the Architecture Design

## 3.4 Architectural Evaluation

Once synthesized (Figure 4), the architecture needs to be evaluated to assess if it satisfies the ASRs and the constraints (Figure 5). We have used the SAAM method [6] to evaluate the synthesized architecture, as illustrated in Figure 6. For example, the architect specifies the application of SAAM to evaluate the 'View Bike' component. ChatGPT presents the scenario for evaluating the 'View Bike' component individually and also scenarios where it interacts with other components. Based on the interaction of individual and interacting scenarios, an evaluation report is produced that shows the evaluation of the functionality, quality, and constraints of CampusBike architecture.



List Individual Scenarios and Interacting Scenarios for Evaluation

Share the list of functionality, quality and constrainst that have been evaluated

Architect

```
Apply the Software Architecture Evaluation Method (SAAM)
to evaluate the ViewBike component of the CampusBike
```

```
To apply SAAM scenario to evaluate the "View Bikes"
feature in the Campus Bike System, you can follow these
steps:
```

1
2
3
View Bike Evaluation
View Bike Component
Classify and Prioritise Evaluation Scenarios
Classified and Prioritised Evaluation Scaenarios
Individual Scenario
Scenario Interaction
Evaluation Report
Evaluation of Individual and Interacting Scaerios
Overall Evaluation
Functionality
Quality
Constraints
Evaluation Report for ASRs and Constraints

Figure 6: Evaluating the Architecture

## 4 RELATED WORK

We discuss the most relevant existing research that overviews the application of AI in SE and ACSE (Section 4.1), and the role of ChatGPT in software development (Section 4.2).

## 4.1 AI in Software Engineering and Architecting

The research on synergizing AI and SE can be classified into two distinct dimensions namely AI for SE (artificial intelligence in software engineering) and SE for AI (software engineering for artificial intelligence) [18] [4]. Considering the AI for SE perspective, Xie [18] argued that SE research needs to go beyond traditional efforts of applying AI for tool-based automation and pattern selection with an exploration of methods that instil intelligence in software engineering processes and solutions. Specifically, SE solutions need to maintain the so-called 'intelligence equilibrium' – i.e., unifying and balancing machine intelligence and human intellect – in processes, patterns, and tools etc. for emergent classes of software, such as blockchain and quantum applications [17]. Barenkamp *et al.* [4] combined the findings of a systematic review and interviews with software developers to investigate the role of AI techniques in SE processes. The results of their study pinpoint three areas where SE needs intelligence to tackle (i) automation of tedious and complex SE activities such as code debugging, (ii) big data analytics to discover patterns, and (iii) evaluation of data in neural and software-defined networks. Considering the context of AI in software architecting, Herold *et al.* [8] investigated existing research and proposed a conceptual framework for the application of machine learning to mitigate architecture degradation.

## 4.2 ChatGPT Assisted Software Engineering

From the SE perspective, ChatGPT is viewed as an unprecedented example of a chatbot that can produce well-articulated responses to complex queries. However, it remains an unexplored territory in terms of its potential and perils in the context of software development processes [5, 7]. Most recently, a number of proposals and experimental findings indicate that the research on ChatGPT focuses on supporting engineering education [11, 14], software programming [3, 7], and testing [15]. Avila-Chauvet *et al.* [3] detailed how conversational dialogs of a programmer with ChatGPT enable a human-bot assisted development of an online behavioral task using HTML, CSS, and JavaScript source code. They highlighted that although ChatGPT requires human oversight and intervention, it can write well-scripted programming solutions and reduces the time and effort of a developer during programming. A similar narrative in a blogpost [7] advocated for an incremental process (human dialog with ChatGPT) to enable genetic programming - JavaScript code to solve the traveling salesman problem. In addition to developing the source code, a couple of studies have focused on testing and debugging with ChatGPT [11, 15]. Sobania *et al.* [15] evaluated the performance of ChatGPT in automated bug fixing. In contrast to the status-quo on automated techniques for bug fixing [16], ChatGPT offers a dialogue with a software tester for an incremental identification and fixing of bugs.

**Conclusive summary**: Based on a review of the existing literature, there do not exist any research or development that explores the role of ChatGPT (LLM-driven AI) that can engage software engineers in conversational dialogs to lead and support ACSE. This study complements the most recent research efforts on software test automation and bug fixing with ChatGPT [15] but focuses on architecture-centric development for software systems. In the broader context of AI for SE [18], this study argues for human-bot collaborative architecting that can enrich ACSE process with the architects' knowledge and supervision synergized with bot's capabilities to architect software-intensive systems and services.

## 5 DISCUSSION AND VALIDITY THREATS

We discuss the socio-technical aspects of collaborative architecting (Section 5.1) and highlight potential threats to validity (Section 5.2).

## 5.1 Socio-Technical Issues of ChatGPT in ACSE

In addition to highlighting ChatGPT's potential, we also highlight some perils as shortcomings of collaborative architecting process that need to be discussed in the context of socio-technical aspects. By socio-technical aspects, we refer to a unified perspective on issues such as *what can be 'social' concerns* and *what are the 'technical' limitations* of collaborative architecting. Dedicated research is required to systematically investigate such issues, however, we only pinpoint several prominent ones, as below.

**Response Variation**: In the context of human-bot conversational dialogs, ChatGPT may produce varied responses for exact same queries. For example, we observed that a query such as '*... what architectural style can be best suited to CampusBike system*' may yield varied responses, such as microservices, layered, client-server etc. architecture can be best suited for the system. This and alike variation in recommendations or scripted artifacts (UML script, ASR specification etc.) can impact the consistency of architecting process and ultimately varied analysis, synthesis, and evaluation of the architecture. One of the solutions to minimize response variations is an iterative dialog with ChatGPT to refine its output and architects' oversight to ensure that the architectural artifacts being produced are consistent and coherent.

**Ethics and Intellectual Property**: Textual specifications, architecture specific scripts, and source codes etc. articulated by ChatGPT could give rise to ethical issues or in some cases copyright or intellectual property infringements. For example, ChatGPT generated script for a component (GETLOCATION) that senses user location in CampusBike system may lead to leakage of user location privacy and non-compliant software with regulatory guidelines (GDPR, CCPA etc.) that must be dealt with vigilance. In such cases, the role of architect is critical to ensure the generated architecture does not violate ethics or intellectual property rights (if any).

**Biased Outputs**: The biases in outputs of such conversational bots can be attributed to a number of possible aspects including but not limited to input, training data, and/or algorithmic bias. From an architectural perspective, recommendation bias about specific architectural modeling notation, tactic, pattern, or style etc. may be based on its widespread adoption or bias in training data rather than optimal use in a specific context. Moreover, architectural recommendations (specific style), design decisions (pattern selection), or validation scenarios (evaluation method) may suffer such bias to produce sub-optimal artifacts in ACSE.

## 5.2 Threats to the Validity

Validity threats represent limitations, constraints, or potential flaws in the study that can affect the generalization, replicability, and validity of results. Future work can focus on minimizing these threats to ensure methodological rigor and generalization of results.

**Internal validity** examines the extent to which any systematic error (bias) is present in the design, conduct, and analysis etc. of the study. To design and conduct this study, and considering the internal validity, we followed a systematic approach and utilized a well-known architecting process [9] and architecture evaluation method [6]. The case study based approach combined with incremental architecting (Figure 3) helped us to analyze and refine the study, however, more work is required to understand if the study can be validated with a different architecting process or by adopting other evaluation methods.

**External validity** examines whether the findings of a study can be generalized to other contexts. We only experimented with a single case study of moderate complexity that can compromise study's generalization. Specifically, scenarios with the increased complexity of architecting process (cross-organisational development), class of software to be developed (mission-critical software), and human expertise (novice/experienced engineers) can affect the external validity of this research. Future work is planned, highlighted in the conclusions section, to validate the process of collaborative architecting by engaging architecting teams and analyzing their feedback to understand the extent to which the external validity can be minimized.

**Conclusion validity** determines the degree to which the conclusions reached by the study are credible or believable. In order to minimize this threat, we followed a three-step process (Figure 2) to support a fine-grained process to architect the software and validate the results (future work). Moreover, a case study based approach was adopted to ensure scenario-based demonstration of the study results. However, some conclusions (e.g., architect's productivity, ChatGPT's efficacy) can only be validated with more experimentation involving multiple case studies, diverse teams, and real context scenarios of collaborative architecting.

## 6 CONCLUSIONS AND FUTURE RESEARCH

ChatGPT has emerged as a disruptive technology, an unprecedented conversational bot, that mimics human conversations and generates well-articulated textual artifacts (recommendation, scripts, source codes etc.) - often referred to as a 'solution that seeks a problem'. Among a plethora of its use cases that range from content creation to digital assistance and acting as a virtual teacher etc., ChatGPT's role as a DevBot and its capability to architect software-intensive systems remain unexplored. This research investigates the potential and perils of ChatGPT to assist and empower the role of an architect who leads the process of architecting, and collaborate with a human to enable ACSE. The research advocates that in the context of AI for SE, traditional efforts of applying AI for tool-based automation should focus on a broader perspective, i.e., enriching existing processes by instilling intelligence in them via efforts like human-bot collaborative architecting. The case study reflects a practical case of *how a software can be architected with ChatGPT?* and *what factors need to be considered in collaborative architecting?* Variance in responses and artifacts, types of ethical implications, level of human decision support/supervision, along with legal and socio-technical issues must be considered while integrating ChatGPT in SE or ACSE processes. The research needs empirical validations, grounded in evidence and experimentation, to objectively assess factors like enhancing engineers' productivity, SE process optimization, and assisting novice developers and designers to engineer complex and emergent classes of software effectively with ChatGPT.

**Needs for future research**: We plan to extend this study as a stream of research that explores human feedback and validation (i.e., architects' perspective) and integrating ChatGPT in a process to develop software services for quantum computing systems. More specifically, quantum computing and quantum software engineering has emerged as a quantum computing genre of SE that faces a lack of human expertise to synergize the skills of engineering software and knowledge of quantum physics. We are currently working in engaging a number of software development teams with diverse demography attributes (e.g., geo-distribution, type of expertise, level of experience, class of software system) in controlled experiments to architect software systems using ChatGPT and document architects' responses. Specifically, with a case study that involves ChatGPT assisted architecting shall allow us to capture feedback of architects via interviews or documents to empirically investigate aspects like usefulness, rigor, acceptance, impact on human productivity, and potential perils of ChatGPT in ACSE.

## REFERENCES

[1] Aakash Ahmad, Arif Ali Khan, Muhammad Waseem, Mahdi Fahmideh, and Tommi Mikkonen. 2022. Towards process centered architecting for quantum software systems. In *Proceedings of the 1st IEEE International Conference on Quantum Software (QSW)*. IEEE, Barcelona, Spain, 26–31.

[2] Aakash Ahmad, Muhammad Waseem, Peng Liang, Mahdi Fehmideh, Mst Shamima Aktar, and Tommi Mikkonen. 2023. Replication Package for the Paper: Towards Human-Bot Collaborative Software Architecting with ChatGPT. https://github.com/shamimaaktar1/ChatGPT4SA.

[3] Laurent Avila-Chauvet, Diana Mejía, and Christian Oswaldo Acosta Quiroz. 2023. ChatGPT as a Support Tool for Online Behavioral Task Programming. *SSRN preprint SSRN:4329020* (2023).

[4] Marco Barenkamp, Jonas Rebstadt, and Oliver Thomas. 2020. Applications of AI in classical software engineering. *AI Perspectives* 2, 1 (2020), 1.

[5] Ali Borji. 2023. A Categorical Archive of ChatGPT Failures. *arXiv preprint arXiv:2302.03494* (2023).

[6] Liliana Dobrica and Eila Niemela. 2002. A survey on software architecture analysis methods. *IEEE Transactions on Software Engineering* 28, 7 (2002), 638–653.

[7] Fernando Doglio. 2022. The Rise of ChatGPT and the Fall of the Software Developer — Is This the Beginning of the End? https://tinyurl.com/3mxrfmjh

[8] Sebastian Herold, Christoph Knieke, Mirco Schindler, and Andreas Rausch. 2020. Towards Improving Software Architecture Degradation Mitigation by Machine Learning. In *Proceedings of the 12th International Conference on Adaptive and Self-Adaptive Systems and Applications (ADAPTIVE)*. IARIA, 36–39.

[9] Christine Hofmeister, Philippe Kruchten, Robert L Nord, Henk Obbink, Alexander Ran, and Pierre America. 2007. A general model of software architecture design derived from five industrial approaches. *Journal of Systems and Software* 80, 1 (2007), 106–126.

[10] Johan F Hoorn, Rik Farenhorst, Patricia Lago, and Hans Van Vliet. 2011. The lonesome architect. *Journal of Systems and Software* 84, 9 (2011), 1424–1435.

[11] Sajed Jalil, Suzzana Rafi, Thomas D LaToza, Kevin Moran, and Wing Lam. 2023. ChatGPT and Software Testing Education: Promises & Perils. *arXiv preprint arXiv:2302.03287* (2023).

[12] Philippe Kruchten, Henk Obbink, and Judith Stafford. 2006. The past, present, and future for software architecture. *IEEE Software* 23, 2 (2006), 22–30.

[13] Ivano Malavolta, Patricia Lago, Henry Muccini, Patrizio Pelliccione, and Antony Tang. 2012. What industry needs from architectural languages: A survey. *IEEE Transactions on Software Engineering* 39, 6 (2012), 869–891.

[14] Junaid Qadir. 2022. Engineering Education in the Era of ChatGPT: Promise and Pitfalls of Generative AI for Education. *TechRxiv preprint techrxiv.21789434* (2022).

[15] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An Analysis of the Automatic Bug Fixing Performance of ChatGPT. *arXiv preprint arXiv:2301.08653* (2023).

[16] Simon Urli, Zhongxing Yu, Lionel Seinturier, and Martin Monperrus. 2018. How to design a program repair bot? insights from the repairnator project. In *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. ACM, 95–104.

[17] Eoin Woods. 2016. Software architecture in a changing world. *IEEE Software* 33, 6 (2016), 94–97.

[18] Tao Xie. 2018. Intelligent software engineering: Synergy between AI and software engineering. In *Proceedings of the 11th Innovations in Software Engineering Conference (ISEC)*. ACM, 1–1.