Joonas Uusnäkki

DESIGN PRINCIPLES FOR PROMPT ENGINEERING WITHIN LARGE LANGUAGE MODELS: CASE STUDY ON SOFTWARE MAINTENANCE

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2023

ABSTRACT

Uusnäkki, Joonas
Design principles for Prompt Engineering within Large Language Models: Case Study on Software Maintenance
Jyväskylä: University of Jyväskylä, 2023, 56 pp.
Information Systems, Master’s Thesis
Supervisors: Mikkonen, Tommi & Myllymäki, Pasi

Software maintenance is a significant phase in software´s lifecycle that is highlighted in large, business-critical systems. In this thesis, the impact of generative AI on software maintenance is investigated. We conducted a constructive design science research on the usage of Large Language Models (LLMs) in the analysis of software code base, with 36 chats within 4 different LLMs, seeking systematic approaches for enhancing software maintenance. We discovered that LLMs systematically produce insights and suggestions in software analysis within real production systems. As a tool for the study, we introduce the PESD framework, which serves as a foundation for prompt engineering within software development environment, offering design principles for the prompt engineering processes.

Keywords: generative AI, large language models, prompt engineering, software maintenance, PESD framework, design science research

TIIVISTELMÄ

Uusnäkki, Joonas
Suunnitteluperiaatteet kehotesuunnittelulle suurissa kielimalleissa:
tapaustutkimus ohjelmistojen ylläpidosta
Jyväskylä: Jyväskylän yliopisto, 2023, 56 s.
Tietojärjestelmätiede, Pro gradu -tutkielma
Ohjaajat: Mikkonen, Tommi & Myllymäki, Pasi

Ohjelmistojen ylläpito on merkittävä vaihe ohjelmistojen elinkaarella, joka korostuu suurissa, liiketoimintakriittisissä järjestelmissä. Tässä pro gradu-tutkielmassa tutkitaan generatiivisen tekoälyn vaikutusta ohjelmistojen ylläpitoon. Toteutimme konstruktiivisen suunnittelutieteen tutkimuksen suurten kielimallien käytöstä ohjelmistokoodin analyysissä 36 keskustelun avulla 4 eri mallilla etsien systemaattisia lähestymistapoja ohjelmistojen ylläpidon parantamiseksi. Huomasimme, että suuret kielimallit tuottavat järjestelmällisesti näkemyksiä ja ehdotuksia ohjelmistoanalyysiin todellisissa tuotantojärjestelmissä. Tutkimuksen työkaluksi esittelemme PESD-viitekehyksen, joka toimii perustana kehotesuunnittelulle ohjelmistoympäristössä ja tarjoaa suunnitteluperiaatteet kehosuunnittelun tueksi.

Asiasanat: generatiivinen tekoäly, suuret kielimallit, kehotesuunnittelu, PESD-viitekehys, ohjelmistojen ylläpito, suunnittelutiede

FIGURES

FIGURE 1 Transformer architecture (Vaswani et al., 2017) 8
FIGURE 2 Prompt engineering techniques 12
FIGURE 3 Generative AI project life cycle (Barth et al., 2023).................... 15
FIGURE 4 Design Science Research framework (Hevner et al., 2004) 21
FIGURE 5 TextGen web UI 27
FIGURE 6 PESD framework 29
FIGURE 7 FEDS framework (Venable et al., 2016)...................................... 31

TABLES

TABLE 1 Examples of LLM model families 14
TABLE 2 Design Science Research 19
TABLE 3 DSR Contribution Types (Gregor & Hevner, 2013)................... 22
TABLE 4 IT artifacts.. 23
TABLE 5 Compared LLMs.. 26
TABLE 6 Prompt design principles with PESD framework 30
TABLE 7 Evaluation goals, objectives and artifact elements.................... 32
TABLE 8 Evaluation episodes 34
TABLE 9 Results... 35
TABLE 10 Detailed results 49

TABLE OF CONTENTS

ABSTRACT
TIIVISTELMÄ
FIGURES AND TABLES

1 INTRODUCTION 1

2 GENERATIVE AI 4
  2.1 Artificial intelligence 4
  2.2 Generative AI 5
    2.2.1 Natural Language Processing 6
    2.2.2 Large language models 6
    2.2.3 Prompt engineering 11
  2.3 AI models and GAI workflow 13

3 RESEARCH SETTING..16
  3.1 Research questions 16
  3.2 Case company: Nokia 17
  3.3 Design Science Research methodology 18

4 CASE STUDY..24
  4.1 Model selection 24
  4.2 Design and development...27
    4.2.1 Foundations of the framework...27
    4.2.2 Introducing PESD framework and prompt design principles ..28
  4.3 Evaluation and results 31

5 DISCUSSION 36
  5.1 Revisiting research questions..37
  5.2 Implications and future research...38
  5.3 Limitations 38

6 CONCLUSION 40

REFERENCES..41

APPENDIX 49

1 INTRODUCTION

Software development as a field of expertise in engineering is unique and dynamic in its ways of working. In contrast to traditional products like books and clothes, which are designed, developed, and delivered to customers as static entities, software products differ significantly in their dynamic nature. Modern software projects typically involve developers working on them for extended durations due to their ever-evolving requirements, rising complexity and the continuously progressing nature of the industry (Jacobson et al., 2022). Practices of DevOps, such as continuous-integration and continuous-delivery (CI/CD) act as modus operandi in fast-phased industry, enabling teams to deploy products even faster and leaving more time for innovation (Ebert et al., 2016). As a result, software products often undergo continuous changes and updates to adapt to evolving needs and technological advancements, requiring varying frameworks and ways of working.

The Software (System) Development Life Cycle (SDLC) encompasses a set of best practices and distinct phases designed to facilitate the development and maintenance of exceptional information systems (Boehm, 1976). Its origins can be traced back to the 1950s when programming languages began to be more accessible and open to the public, leading to the establishment of structured methodologies for software development. SDLC is widely used in software development, system development, application development and other development domains as a framework to organize tasks. The SDLC consists of different phases that typically include planning, design, development, testing, release, and maintenance (Ruparelia, 2010).

From these phases, the maintenance phase acts as a crucial phase in software´s lifecycle, as it takes away 80-90% of the costs according to research (Koskinen, 2004). Software maintenance is one of the most crucial parts of SDLC. It is an on-going process that initiates after the software product has been deployed into production and continues its lifecycle. Maintenance involves resolving errors, adapting to new business environments with user requirements, and other supportive tasks that maintains the system (Lientz et al., 1978). As the size of systems expand, developers encounter varying issues in maintenance of the software, inhibiting their high quality in functionality and longevity, particularly as the modern software systems are built using varying components as open-source libraries and programming paradigms.

These issues result in significant challenges in cases such as usage, deployment, and testing when handling large or medium-large software systems, particularly as the utilization and quantity of diverse technologies, code repositories, external components, and other crucial factors increase (Wei et al., 2016). The problems can compound further if there are additional issues in technical debt, such as the failure to update the software system, developers leaving the company, and only a handful of developers being responsible for maintaining the project, among others (Bogner et al., 2018).

Research on developers maintaining the software has revealed that about half of their work hours are dedicated to comprehending and understanding the code they are responsible for maintaining, making it the largest cost factor in SDLC (Standish, 1984). As such, maintenance process should be researched to acquire more systematic and improved ways to maintain software systems (Bogner et al., 2018). In this context, generative artificial intelligence (generative AI/GAI) could provide innovative solutions for learning new concepts and onboarding developers for entire codebases. Especially trainee level developers could benefit from a feature where technologies such as generative AI present introduce components, propose repair suggestions and give more sophisticated examples about the application area or element itself (Bull & Kharrufa, 2023). Thus, the maintenance phase is the primary target in the context of this thesis.

In this thesis, we conduct an empirical study on generative AI's impact on software maintenance. The key objective of our study is to investigate ways to systematically gather outputs that provide value for researchers and developers using generative AI, specifically the large language models (LLMs). We conduct design science research on LLMs with a focus on producing a framework for prompt engineering, with the evaluation done using four different models. With this framework, we delve into finding systematic information on improvements for software maintenance.

Literature review for this thesis was done mainly via Google Scholar search engine. Scientific publications were chosen from databases that varied largely from journals to scientific conferences. Other databases that were used in the making of this thesis were Elsevier, Springer Link, IEEE database and JYX among others. Previously mentioned publishers and databases were targeted for more specific searches. Keywords that were included in literature review were mainly concepts about AI and software development such as “generative ai llm,” software maintenance phases”, “NLP software development”, “life cycle costs software product” and “p-tuning limitations”.

Literature review was done rigorously with the focus on quality. This was due to a massive rise in AI related themes in unpublished and published publications which made it difficult to scan the literature diligently. Vast amounts of pre-print papers on open databases have been in rise in the context of generative AI, leaving the processes of peer review not known. Therefore, scientific publications selected to be used in this thesis intend to have a classification level of at least 2 or 3 on their submitted publication channels. Exception is due for well-established scientific publications stored as arXiv pre-print publications. Classification is based on the publication channel search of Publication Forum by the Finnish scientific community (JUFO-rating).

Since the themes of the thesis are very novel and constantly developing, the thesis uses gray literature when referencing the latest instalments on some of the more practical notations: mainly when referencing the LLM usage and latest phenomena, such as glitching tokens.

The rest of the thesis is structured as follows. Chapter 2 includes the literature review on generative AI. In this chapter, AI's history and current applications are presented. Generative AI's essential principles are introduced with their main functions on the phenomenon.

Based on the literature review, Chapter 3 presents the study's research questions, research setting, and methodology. In this chapter, the thesis introduces the main setting for the study: the intersection of AI and software maintenance. The case company is also introduced. The methodology of Design Science Research is introduced with its respective phases in this chapter. Additionally, the basis for the empirical study is established through design science.

Chapter 4 introduces the case study and results with the design and development of our research artifacts. These artifacts are also evaluated in this chapter. Results are presented with evaluations of previous research and practical implications.

With the study results covered, Chapter 5 is based on a discussion about the study. Study results are viewed and compared towards the research questions and hypotheses. Possible implications, limitations, and future research are also discussed. Finally, in Chapter 6, the thesis concludes with the summarization of the study and its findings.

This thesis was produced with the help of generative AI applications, specifically ChatGPT and Bard. These applications were primarily used to check for possible grammar mistakes and swiftly find new information about the concepts discussed in the thesis. The content in this thesis is based on scientific publications selected through a rigorous literature review. No text or other content was purely copied straight from these applications' outputs without scientific references, and modifications were made to ensure originality.

2 GENERATIVE AI

This chapter describes generative AI's current characteristics, functions, and applications. The chapter gives an overview of LLMs, prompt engineering, and generative AI applications.

2.1 Artificial intelligence

AI has been long described as the next great innovation and milestone in human history. Although AI has been researched from the 1950's until today, it was not until the late 2010s that the first practical and business applications of AI began to reach large user bases (Haenlein & Kaplan, 2019). With a tremendous growth of user base, various entities, from private companies to public sector organizations, have started making investments in the adoption of AI applications and the general development of AI (European Commission. Joint Research Centre., 2020). Remarkable milestone AI-systems, such as IBM's Deep Blue in 1997 and Google's AlphaGo in 2015, ensured AI to become the status quo in the search of the next disruption in the digital world since the Internet (Haenlein & Kaplan, 2019). These significant contributions to AI development keep inspiring the next generation of researchers and practitioners.

Among the most leading and thought provoking AI-subgenres today is generative AI, which has created fruitful discussions on opportunities and threats within its usage and development (Dwivedi et al., 2023; Stokel-Walker & Van Noorden, 2023). Generative AI consists of applications that receive input data from user, processes the data via its own algorithms and models, and outputs data based on the used models' parameters (Bull & Kharrufa, 2023; Kalyan et al., 2021; Min et al., 2023). Output data depends on the application's purpose and model's features, which can vary on uncounted amounts like translation to another language, speech recognition, writing summaries and categorizing context (Kalyan et al., 2021; Stokel-Walker & Van Noorden, 2023). Rise of generative

AI applications have inspired researchers to dive deeper into differing impacts of AI, including societal, ethical, and economic impacts.

One of the most prominent examples of generative AI is ChatGPT chatbot developed on top of the OpenAI research center's GPT (Generative Pre-Trained Transformer) model family (OpenAI, 2022). ChatGPT is capable of generating human-like responses to users who ask the system various questions based on input provided in the search field. The explosive growth of GPT based applications has sparked discussions about the ethics, licensing, and cybersecurity of AI applications (Dwivedi et al., 2023). However, considering these aspects, the applications will become and are already part of our everyday lives, both in the workplace and during leisure time. The integration of generative AI across multiple domains represents a forward-looking path and is assured to undergo extensive exploration within the research within various Information Systems (IS) journals (Dwivedi et al., 2023).

The competition in the development of AI models has provided users and software developers with access to both commercial and open-source solutions. Open-source AI solutions offer the opportunity to further develop solutions freely into different components (Douglas Heaven, 2023). This competition signifies a rapid pace of development, which calls for exploring the application areas of AI further and keeping up with the pace of progress. To ensure that the development of AI applications does not solely focus on technical aspects, it is important to consider the social aspects of AI management as well (Berente et al., 2021).

2.2 Generative AI

The domain of AI has been moving forward with outstanding pace. Numerous applications of multiple varieties have risen in recent years, thanks to the ever-developing power of computing. Applications have come far since the days of Deep Blue, advancing the AI's abilities to those of content generation, speech recognition, trend recognition, service assistance and more, for instance.

A subset of traditional machine learning that has gained massive popularity and adoption in recent years is called generative AI. Generative AI is a subset of AI that involves crafting content through user-provided data, which is processed by a model employing advanced deep learning techniques. The content generated by these models exhibits a strong interdependence with the datasets employed during their training. The nature of the content can exhibit significant variation based on the specific model in use, ranging from text and images to more complex formats such as videos and audio.

Generative AI gained great interest in practice and research from the large success of ChatGPT, a natural language processing (NLP) based LLM application that is accessible through web browser and app (OpenAI, 2022). Models like ChatGPT and BERT have introduced substantial disruptions to the IS domain, sparking discussions on crucial aspects such as privacy, misinformation, and discrimination (OWASP, 2023; Weidinger et al., 2021). These models used in the generative AI have the potential to execute transformative changes to whole industries, reshaping the future landscape of work itself.

2.2.1 Natural Language Processing

NLP is a subset of AI which focuses on interactions between humans and computers (Min et al., 2023). NLP consists of developing the actual algorithms and models beneath the AI applications that can interpret and generate human language in usable and context aware way. NLP has been a field of study for decades, originating from research on early AI and linguistics where statistical techniques were used to find and index great quantity of text efficiently (Nadkarni et al., 2011).

NLP combines various other research fields to understand, interpret, and generate human language text and speech, which some are presented here. NLP has been used in research fields such as linguistics, psychology, and other fields to extend and find solutions to complex scenarios (Goldberg et al., 2020; Manning, 2015).

NLP is used in various digital services such as Siri or Alexa in voice recognition tasks. LLMs using NLP technology are not only specifically used as a singular applications but they are implemented in and integrated with third-party applications. (Finnegan, 2023; Snapchat, 2023)

NLP is used in LLMs as a way for users to communicate with the computing models. LLMs functionality consist of user inputted prompt, the model, and completion or output. Prompts are pieces of tasks formatted in text given to LLM. Prompts define the functionality of the model itself as the tasks can includes role, purpose, target and even character the user intends the completion to be. For instance, a painter can input “What is the best paint type for kitchen cabinets” as a prompt. Poet can seek inspiration from LLM by prompting “Write three regretful lines in the last stanza of this poem:” and include the yet finished poem. Finally, a software developer can prompt questions like “Write me a basic program that prints something in JavaScript” to achieve natural text to programming language conversion via LLM.

2.2.2 Large language models

Generative AI uses variety of tools to function, the most known ones being previously referenced LLMs. LLMs are made of billions of parameters and are trained with over trillions of datasets over days, weeks or even months to achieve their functionality (Kaddour et al., 2023; Min et al., 2023; Mökander et al., 2023). LLMs are employed for NLP, where users input text as prompts to the model, which then endeavors to solve problems by emulating human-like behavior. This is achieved through predictions based on machine learning. Examples of these models and model families are GPT, Llama, BERT, BLOOM and FLAN-T5. All the models previously mentioned are called foundation or base models. These models often act as a mothership for other smaller and fine-tuned models, that are available through third parties and as open-source codebases (Bommasani et al., 2022).

The factor that enables the computing efficiency of LLMs is an architecture style called transformer architecture (Vaswani et al., 2017). Transformer architecture defines a way for current language models to have superb memory management and fast-phased completion time. This is due to transformer architecture's ability for parallel computing from its multi-head attention mechanism which increases the computing power significantly (Khan et al., 2022). With transformer architecture, scalability of complex models and ability to control and operate on large datasets is made possible (Khan et al., 2022; Wolf et al., 2020).

As a preprocessing task, the tokenization process is completed before the actual LLM tasks. Tokenization refers to process where input text is divided into smaller units called tokens (Wolf et al., 2020). For instance, "Cat has a friend." is divided into ["Cat", "has" "a", "friend", "."] tokens. This way the actual models can perform various tasks, as they lean on tokens rather than raw text. Tokenization provides structure and manageability for the text.

Transformer architecture is founded on both the encoder and decoder components, each comprising several levels. Input prompts are passed to encoder which turns the text into tokens. These tokens are passed to decoder which in turn creates new tokens. The capability is called sequence-to-sequence modeling which is vital in NLP related tasks. The tokenization is essential to the transformer architecture: it acts as a backbone for sequence-to-sequence modeling and attention mechanisms through parallel processing (Vaswani et al., 2017). Transformer architecture is presented in Figure 1.

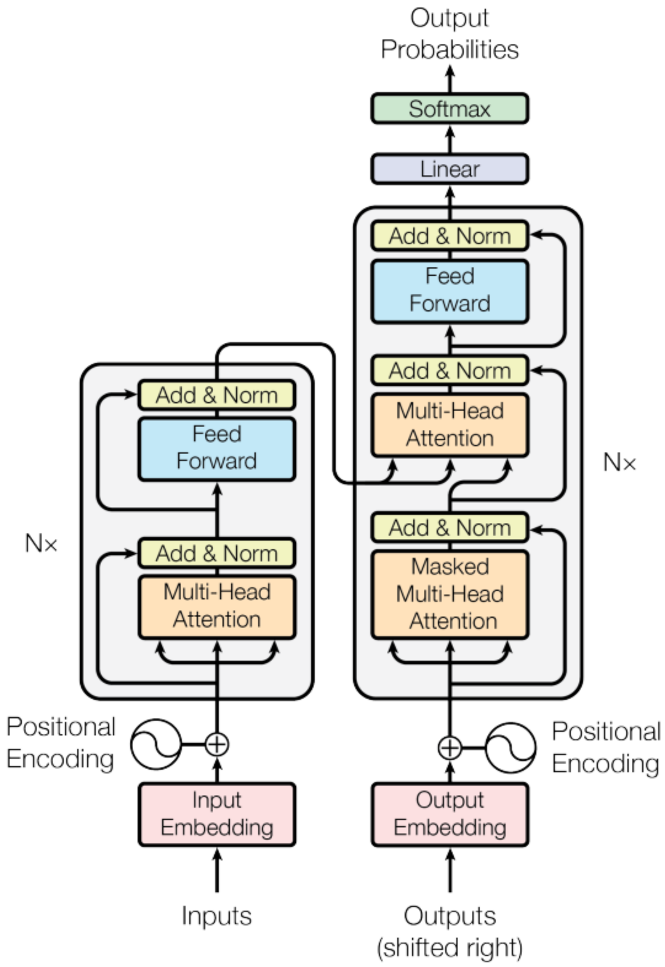


FIGURE 1 Transformer architecture (Vaswani et al., 2017)

The architecture of LLMs is not fixed and can vary, encompassing encoder-decoder models, encoder-only models, and decoder-only models. The selection of the architectural design depends on the specific use cases. For example, GPT based LLMs employ a decoder-only architecture, in which the model's target is to predict the next token from the input token. (Min et al., 2023)

Another key concept in LLMs is embedding. Embeddings are vector representations of words, sub-words, or tokens that enable the model to understand the text data. The tokens of the model each are associated with an embedding vector that gives tokens semantic information and context. Model learns these embedding vectors during its training process. These vectors are used by model to perform various tasks such as generating suggestions and understanding certain relations between tokens and words. Embedding is a highly effective method when introducing a new concept to a model. (Kalyan et al., 2021)

LLMs are built upon deep learning techniques that intricately utilize model weights. Weights, in this context, represent the parameters of the model that are fine-tuned during the training process via validation set (Berthelot et al., 2019). These weights encapsulate the learned relationships between input data and the desired outputs, enabling the model to generate coherent and contextually relevant content. The magnitude and arrangement of these weights determine the model's ability to understand patterns, make predictions, and ultimately generate meaningful and coherent responses or content (Kalyan et al., 2021).

Thus, the larger the model itself is, as in pre-training data, the better the understanding of the prompt domain the model has (Kalyan et al., 2021). It is evident that larger models consistently outperform their smaller counterparts in terms of response quality. However, it is critical to acknowledge the significant resource demands associated with these larger models. This accessibility challenge arises due to the substantial computational resources required, which translate into elevated costs and accessibility barriers. Furthermore, the global availability and supply chain disruptions of GPUs intensify this phenomenon (Fung, 2023).

Before a LLM can be used in action, it must be pre-trained. In the case of NLP, pre-training contains the act of predicting the next word or missing word in a sentence and thus training the entire model (Kalyan et al., 2021; Peters et al., 2018). This way the model is deployed with the basic understanding of the language. Pre-training is executed with large datasets, usually massive text corpuses, containing billions of words and differing in size, quality, languages and genre (Min et al., 2023). These corpuses are public databases that are available, such as Wikipedia, BookCorpus or Oxford English Corpus. Pre-training uses unlabeled data from these corpuses, which teaches the model the grammar, nuances, syntax, and structure of language it is being laid open to. Pre-training is done generally with ongoing parallelized processing on a large amount of graphics processing units (GPUs) (Vaswani et al., 2017). GPU’s ability for massive processing power harnesses LLMs to be pre-trained. The purpose of pre-training is to give the model the basic understanding of the language, not to gain mastery in a specialized role in custom tasks.

Customization is highly available for these models with fine-tuning. If user intends to use LLMs only in their scoped environment, user can fine-tune these models to achieve a customized model specifically for their use cases (Howard & Ruder, 2018). Fine-tuning is the process in which the pre-trained LLM is modified to act as a specialist for given occasion that the user wants it to be. Fine-tuning is conducted on pre-trained models via supervised learning by utilizing user-labeled data as input to the model. Training uses supervised learning techniques to adjust its internal weights on specific tasks (Brown et al., 2020). For example, user could assemble a dataset or known server errors and label them as critical and non-critical. With fine-tuning, there is no need to heavily train the model again with potentially billions of data and the model can still be used with relatively low effort. Fine-tuning would be beneficial for use cases in which there is little-to-none need for general system but rather a specialist which can process input data more effectively. (Min et al., 2023) This approach enables the model to generate even more precise and contextually appropriate outputs for the given prompted input.

Although LLMs possess considerable potential to affect numerous fields and ways of working, they also are accompanied with their own challenges. Selection of these challenges are presented here.

LLMs possess limitations of at least two kinds, which can be addressed with different techniques. Training methodologies, such as pre-training and fine-tuning, target the first kind of limitations. These limitations arise when the model is not precisely trained to perform in the intended user domain, leading to peculiar outputs and misinformation from the model, irrespective of its familiarity with the domain itself. Ensuring alignment between user expectations and the model's output is a critical issue when employing LLMs in a specific problem domain. (Weidinger et al., 2021)

The second kind of limitations occurs when the model lacks sufficient familiarity with the input domain due to missing information. This deficiency arises from the absence of details and relationships between crucial concepts within the domain. For instance, models developed and trained before the year 2020 would lack information on the COVID-19 virus or the war in Ukraine, not to mention any potential missing narratives and voices from the dataset. To address these limitations, one must introduce the missing information, for example with prompt engineering techniques, through training techniques via APIs with injected data, or through vector databases. In the latter case, the required information is loaded into a vector database, which maps data points to vectors that are then matched with the existing data based on the best match. The choice of techniques to employ depends on the size and complexity of the available information. (Weidinger et al., 2021)

Datasets, with which these models are trained on, are in critical position when examining models' integrity and functionality. Bias is one of the most unfortunate assets of AI which is formed based on these datasets that represent human societies throughout time (Akter et al., 2021). Because these datasets are set on historical data, bias can be shown in unfortunate situations. For example, instances of political biases have been found in LLMs, thereby contributing to the polarization on the outputs of these models (Feng et al., 2023).

AI hallucinations is also a major problem in the usage of LLMs. AI hallucinations refer to data outputs where the end results are not based on the actual training data. Hallucinations may occur when the model is unfamiliar with the actual domain it is being used and its coverage on the used data is not competent. These hallucinations can be divided into intrinsic and extrinsic hallucinations. Intrinsic hallucinations occur when the output data does not logically relate to the input prompts state. Extrinsic hallucinations are present in outputs where the output data cannot be evaluated and reviewed based on search engines or other information sources. (Ji et al., 2023)

At the corporate level, the cost factor emerges as pivotal in influencing the decision to acquire and utilize LLMs. LLMs use substantial amount of computing power in their operations, which in turn can affect the deployment of these LLMs in private and public organizations as the pre-training process of a LLM can cost millions of dollars. The computing power and data storage needed to pre-train and fine-tune these models are centralized for handful of large organizations, leaving the research and development efforts away from other organizations. (Kaddour et al., 2023; Min et al., 2023)

Glitching tokens have also played a part in LLM obstacles. Glitching tokens are usually a specific input words that unexpectedly cause the model to make a malfunction to the output. Outputs act as odd and bizarre as they do not have a logical reference to the original input prompt. These types of tokens can occur when data of pre-trained model and the actual token differ. (Rumbelow & mwatkins, 2023)

Last, for their massive need for computing power and energy to be trained and operated on, LLMs requirement for immense energy has a substantial effect on the ICT sector's carbon footprint (Freitag et al., 2021; Strubell et al., 2019). As every energy-consuming innovation, LLMs and AI in general will be subjects to larger constraints on consumption and carbon emissions on ICT-sector, if we aspire them to become positive for our environment (Freitag et al., 2021).

2.2.3 Prompt engineering

As stated, fine-tuning a LLM can enable the model to perform contextually accurately within the user's desired scope. However, there exists a much more lightweight alternative for modifying the model's outputs: prompt engineering.

Prompt engineering is a technique in which the user of a LLM or similar AI tool is designing and implementing effective prompts to seek certain behavior and output. Like fine-tuning, the purpose of prompt engineering is to accessibly affect the models for it to produce context-aware outputs. Desired outputs are pursued through effective input prompts, which can be attained with attention given on clarity, context, instructions, examples, bias, evaluation, and sensitivity. Prompt engineering is highly effective and valuable in situations where user pursues specification from the model that is used in relatively general purposes. Model retraining is not directly involved in prompt engineering, which falls below pre-training and fine-tuning. Instead, prompt engineering is done in the actual application interface and executed with different tactics, such as in-context learning.

In-context learning (ICL) is a major, behavioristic phenomenon where LLMs learn and adapt from different the different states of input prompts. Within ICL, the model parameters remain unaltered, and the phenomenon occurs as an integral part of prompt engineering. ICL is pursued via task-related examples that are given to the model. This can be pursued with prompt engineering that incorporates n-shot learning, generally categorized to zero-shot, one-shot or few-shot learning. (Brown et al., 2020)

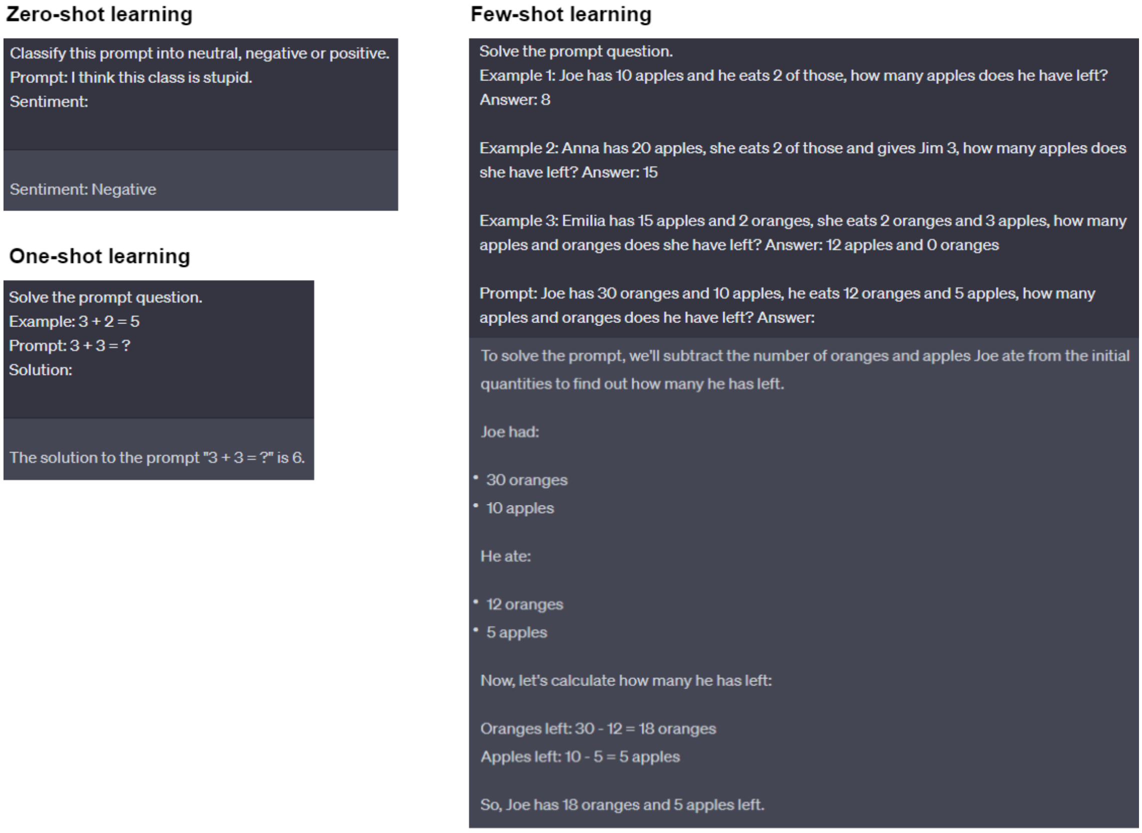
Zero-shot, one-shot, and few-shot learning pertain to data availability and context in machine learning. They operate differently depending on the data's availability: the less data the model is exposed to, the more challenging its task becomes, requiring greater effort in computing context-aware outputs. In the context of LLMs, the techniques can also be labeled as zero-shot *prompting*. With these machine learning techniques, LLMs can be enabled to interpret new domain areas and tasks without resource consuming pre-training or fine-tuning (Sung et al., 2018).

Zero-shot learning is a technique where a LLM is harnessed with tasks that are not labeled during the training process. With this, the model must work harder to achieve a context-based outputs within completely new problem. Zero-shot learning is typically practiced as single prompts to the model system.

One-shot learning differs from zero-shot learning in that it gives the model one labeled example. With this example, the goal is to issue the model better chances to make correct prediction.

Few-shot learning deals with same kind of prompts but gives labeled examples to the model. This technique provides the model partly examples for it to solve the novel problems it is faced. The aim of one-shot learning is to develop the context-awareness of the model on limited examples.

With these prompt engineering techniques, model can be employed to act as the user intends it to behave. The illustrations of these techniques are presented in Figure 2.



Zero-shot learning

Classify this prompt into neutral, negative or positive.
Prompt: I think this class is stupid.
Sentiment:

Sentiment: Negative

One-shot learning

Solve the prompt question.
Example: 3 + 2 = 5
Prompt: 3 + 3 = ?
Solution:

The solution to the prompt "3 + 3 = ?" is 6.

Few-shot learning

Solve the prompt question.
Example 1: Joe has 10 apples and he eats 2 of those, how many apples does he have left? Answer: 8

Example 2: Anna has 20 apples, she eats 2 of those and gives Jim 3, how many apples does she have left? Answer: 15

Example 3: Emilia has 15 apples and 2 oranges, she eats 2 oranges and 3 apples, how many apples and oranges does she have left? Answer: 12 apples and 0 oranges

Prompt: Joe has 30 oranges and 10 apples, he eats 12 oranges and 5 apples, how many apples and oranges does he have left? Answer:

To solve the prompt, we'll subtract the number of oranges and apples Joe ate from the initial quantities to find out how many he has left.

Joe had:

- 30 oranges
- 10 apples

He ate:

- 12 oranges
- 5 apples

Now, let's calculate how many he has left:

Oranges left: 30 - 12 = 18 oranges
Apples left: 10 - 5 = 5 apples

So, Joe has 18 oranges and 5 apples left.

FIGURE 2 Prompt engineering techniques

In Figure 2, we present a few examples of n-shot learning. The steps required to incorporate these techniques into prompt engineering involve ensuring cohesive and context-aware text, verifying the logic behind the task, and providing more information about the task domain. These techniques can be applied in any domain, irrespective of their themes, and they may take some time for the desired logic to be discovered by the model. N-shot learning is not solely used to obtain a straightforward answer from the model app but rather to deploy the model with context, making it ready for more complex tasks in the domain throughout the prompt lifecycle and beyond.

On top of prompt engineering, there is a technique that acts also with NLP LLMs as they are: prompt tuning (p-tuning). In prompt tuning, the model is harnessed with example data and logic, just like in prompt engineering. However, prompt tuning uses data called soft prompts, AI-created input data that is put into a model in the start of personalized prompts. Soft prompts cannot be edited or viewed in text: they consist of continuous embeddings that are detected by AI, which are abstract and random in their nature (Qin & Eisner, 2021). With this process the AI cannot explain why it chose these soft prompts which can create risks when using LLM affected by the prompt tuning process, such as bias (Khashabi et al., 2022).

Prompt engineering on the other hand uses hard prompts as discrete ones that are created by human users. The objective of prompt tuning is the same in prompt engineering: to use the standard pre-trained model as a specialist in designated domain. Prompt tuning differs from prompt engineering also in capabilities to vary its expertise in various domains more swiftly and easily. Prompt tuning is practiced with a smaller, trainable model that creates task specific tokens via encoding (Lester et al., 2021).

2.3 AI models and GAI workflow

To develop AI applications for various purposes, it is essential to understand the fundamental principles behind the underlying principles. The algorithms of the latest AI models are primarily based on machine learning and deep learning techniques, which have evolved over time to process unstructured data, such as speech or images (LeCun et al., 2015). Among these models, LLMs have gained noticeable attention within the research and practical communities. These models function as applications, chatbots, algorithms and more where learning form user inputted data acts as a backbone for the model's purpose. They can "converse" with users and respond to different inputs in a manner that appears aware or human-like (Dwivedi et al., 2023).

As software evolves, it is essential to consider the versions and developmental aspects such as processing power and size of the applications when considering them to be used in a study or as a part of business process. LLMs have been deployed and have gone through changes during their development since last decade´s massive rise in their quantity. Table 1 represents some of the key model families, chosen architecture, release year, their developer(s) and if the model is open-source or not. Open-source validation is based on whether the model is fully open-source, allowing developers to modify them with techniques such as fine-tuning.

TABLE 1 Examples of LLM model families

Model families	Architecture	Release	Developer	Open source
GPT	Decoder-only	Jun 2018	OpenAI	No
BERT	Encoder-only	Oct 2018	Google	Yes
PaLM	Decoder-only	Apr 2022	Google	Yes
Llama	Encoder-only	Feb 2023	META	Yes

Throughout the years, various models have been developed, allowing us to gain an overview of these models based on their respective functionalities. Models that are in the most large-scale usage right now tend to use the encoder-only architecture for their functionality on NLP.

GPT model family (Generative Pre-trained Transformer) is developed and maintained by OpenAI research center. GPT family is based on decoder-only architecture and enjoys clearly the greatest popularity at the moment from all LLMs with GPT-3.5 and GPT-4.0 available on ChatGPT (GlobalData, 2023).

BERT is a language model family introduced by Google (Devlin et al., 2019). It has a decoder-only architecture and is based on transformer architecture. BERT introduced bidirectional context, which powers the model to gain understanding of the context with analysis on left and right sides of input words. Bidirectional context is achieved with a technique called Masked LM (MLM). This technique gained popularity and made it possible to models derived from BERT and others to be suited for their specific tasks even better (Devlin et al., 2019).

PaLM is a decoder-only-based LLM family with an architecture derived from the transformer architecture. Its initial release was in 2021, with PaLM 2 being released in May 2023, featuring improvements in coding, reasoning, and language capabilities. PaLM 2 is used, for example, as a part of Google's Bard application, serving as the primary technology that enables communication between the user and the model. (Chowdhery et al., 2022; Ghahramani, Zoubin & Google, 2023)

Llama is based on decoder-only architecture. It was created by Meta in February 2023 and has since gained popularity. As in July 2023, Llama 2 was released to the public as open-source project for research and commercial use with even more parameters than its precursor. (Meta, 2023a, 2023b)

When a development team starts to work on a generative AI project, they must take into notice multiple aspects in different phases. Following workflow divides the generative AI project into phases regarding scope, model selection, adaption and alignment and application integration.

First, the development team analyses the business problem that they are facing. In this phase, the requirements are specified to match the business problem. Requirements present the current altercations of the system and are designed to lead the development team to choose a model to be used in the project. Second, a model must be selected to match the use case definitions. This is one of the most crucial steps in the workflow as the model usage is affected by use cases, computing resources, domain-specific tasks, and business environments. Selection of the model typically involves the choice between already pre-trained and possibly fine-tuned model or choosing to pre-train your own model. Latter of these takes substantial total of resources and therefore already pre-trained models tend to be in favor.

Third, after choosing the model, the development team adapts the model to be in use in their business domain. This phase can behave as an iterative one as the adaptation and alignment of the model can be executed in various ways: with prompt engineering, fine-tuning, human feedback, prompt tuning, and more. Evaluation is also a key part in this phase as the development team must be sure that their model is working correctly in their use cases.

Finally, the model is integrated to applications as third-party app, plugin, standalone app or as other. In this phase, model is optimized to be deployed in the said use case. As the model is integrated to an application, the application becomes AI-enabled as it is powered by LLM. This workflow is presented in Figure 3.

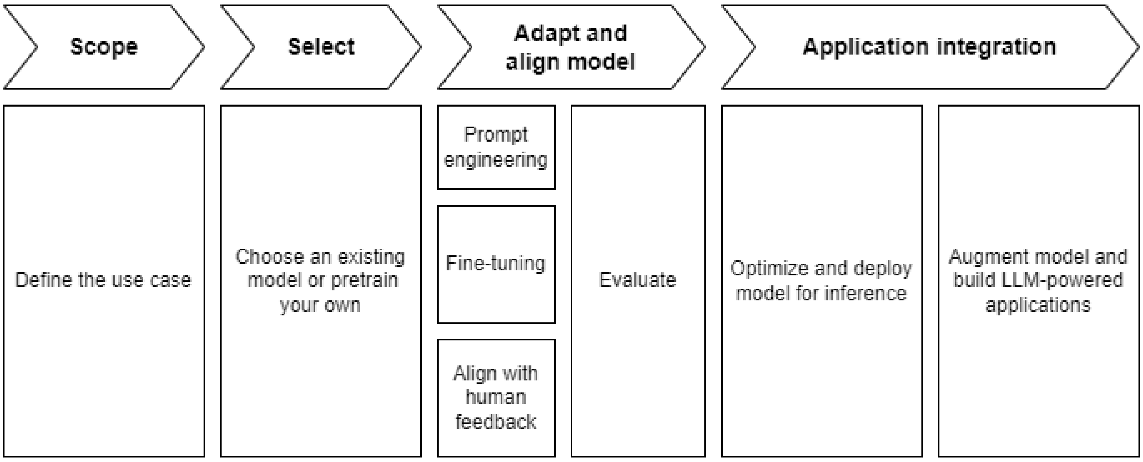

Scope
Select
Adapt and align model
Application integration
Define the use case
Choose an existing model or pretrain your own
Prompt engineering
Fine-tuning
Align with human feedback
Evaluate
Optimize and deploy model for inference
Augment model and build LLM-powered applications

FIGURE 3 Generative AI project life cycle (Barth et al., 2023)

3 RESEARCH SETTING

This chapter describes the case company, case system, and the research setting of the empirical study. The chapter presents the implemented research methodology and justifies the choice with its respective phases.

3.1 Research questions

Research questions are formulated at the intersection of generative AI and software maintenance. These inquiries aim to address the ongoing debate in problem-solving, focusing on the comparison between the utilization of AI solutions and traditional human-powered approaches. Research questions and their hypotheses are presented as follows:

RQ1: How to systematically evaluate LLM outputs in software maintenance?

> *Hypothesis*: It is feasible to construct a framework that facilitates objective comparison of outcomes.

RQ2: How can generative artificial intelligence help software developers in system management and maintenance?

> *Hypothesis*: Generative artificial intelligence can help systematically when organizing system development points and management.

RQ3: How generative artificial intelligence manage development to reduce technical debt?

> *Hypothesis*: Technical debt can be detected in the system using generative artificial intelligence, but a larger scope of technical debt cannot be identified.

With the research questions in mind, the object of the study is to develop design principles to the prompt engineering and to develop an intelligent framework to utilize LLMs in software development. This is done as a Proof-of-Concept (PoC) project where the design principles are tested with a case system containing production code in real life software system product.

3.2 Case company: Nokia

With great possibilities in AI, a variety of research areas have opened in both technical and societal disciplines. Companies have taken on the task of investing in research projects that explore AI to develop their products, work practices, and overall business operations. One such company is the telecommunications corporation Nokia Solutions and Networks Oy, which invests in various product development and innovation projects. As a case company, Nokia provides the framework for this master's thesis in the context of telecommunications software, where software utilizing AI is expected to become more prevalent.

Investments in machine learning and AI, particularly in applications for mobile networks, will enable relevant AI-powered network applications and platforms in the near future (Maeder, 2023). Like other major IT companies, Nokia also prioritizes application development in its operations. Various application development teams at Nokia work on the same software, aiming to improve software using different methods. Therefore, the findings of this thesis could be beneficial for other teams in Nokia as well.

As stated before, introduction of AI-powered applications in large corporations is critically hard. This holds true for Nokia also, as the company is run with massive amounts of data. This data should be handled with extra care as LLMs possess vulnerabilities in data privacy, security breaches and others as stated in LLM06 in OWASP analysis (OWASP, 2023).

Case system is a data-heavy backend system that consists of millions of lines of code, varies in its programming languages and has been maintained for multiple years. The system has been built with opportunistic design where unexpected and unplanned opportunities have risen during the development and maintenance processes due to the evolution of external libraries and subsystems (Mäkitalo et al., 2020). Open-source libraries are heavily in-use with the system, and it is tied to other sub-systems within Nokia’s solutions. In the past, development has been done internally via Nokia’s own team and external consultants have been used in development of system components. System acts as a vital part of the application ecosystem but endures with quiet information and technical debt within earlier selection of technologies and compact development team. Also, system acts as a bottleneck for high-value data between subsystems, which increases the importance of the system even more.

With technical debt we are referring to all code base related debt with libraries, programming language choices, frameworks, requirements, documentation, architectures and other that arise from the compromised tasks done during fast-phased software development environment (Cunningham, 1993). One of the purpose of this study is to answer the call to come up with improved solutions to manage technical debt (Li et al., 2015; Melo et al., 2022). As a case system, the ever-growing requirement management acts as in a key role when choosing what to develop or upgrade next. This process is in a vital position in development team as they have to make decisions on task management regarding stakeholder needs, prioritization, validation, documentation, assigning and other aspects (Li et al., 2014). A study has shown that the lack of detail in system requirement documentations leads to indistinct requirements, adding up to the technical debt (Melo et al., 2022). Generative AI would give more insight into the management of these tasks, therefore answering the research call previously mentioned.

Case company’s role in the study is to provide the target system and needed resources for the research team. The incentive for the case company is to help find efficient and innovative solutions to the issues in management and maintenance of the target system and likewise systems. Thus, future systems could benefit from larger scale usage of LLMs in multiple phases in their lifecycle.

3.3 Design Science Research methodology

The case study of this thesis in done as a constructive work, utilizing Design Science Research methodology. Design Science Research (DSR) is a research methodology that combines the goals of design and research to develop and evaluate innovative solutions to practical problems (Peffers et al., 2007). DSR is particularly applied in the field of information systems, engineering, and technology to explore and evaluate new artifacts and to seek understanding on practical problems. With its iterative and constructive process, DSR creates an effective frame for researchers and practitioners.

In Information Systems (IS), DSR is used in creation and evaluation of socio-technical artifacts that can vary from frameworks and methods to tools and software systems, typically divided into foundations and methodologies (Gregor & Hevner, 2013; Hevner et al., 2004). The creation process, actions and effects of these artifacts are reviewed to enhance the existing knowledge base in the domain research field.

DSR research is based on varying stages, all of which serve a specific purpose. Stages are organized to provide a structured and systematic approach to addressing complex problems while maintaining the principles of scientific inquiry (Hevner et al., 2004; Peffers et al., 2007). The methodology gives a clear way to generate hypotheses, run an experiment and evaluate the artifact based on its iterative process. With this process, other researchers can execute the same research based on the same scientific parameters presented in the study under examination. The stages of DSR are regularly divided into phases presented in Table 2, derived from prior research (Hevner et al., 2004; Peffers et al., 2007).

TABLE 2 Design Science Research

Research phases	**Objective**
1. Problem identification and motivation	Identifying clear research problems to seek solution for and justifying them.
2. Objective definition	Defining the research objectives, checking the status of artifact scope and formulating research questions.
3. Design and development	Creating solution to address the problem. The design process incorporates existing theory and practical knowledge.
4. Presentation	Illustration of the new artifact and demonstrating its functionality in its operational environment.
5. Evaluation	Evaluating the created solution using objective measures and methods. The evaluation aims to demonstrate that the solution meets the defined requirements and provides practical utility.
6. Communication	Informing the research results to external parties, including the scientific community and other relevant stakeholders.

In the context of this thesis, design science provides a scientific methodology for the empirical study, target being the integration of generative AI into the maintenance phase of large software system. The functionality and features of this approach are examined using design science. Furthermore, the knowledge base involved in IS research will increase due to the empirical study. The phases of DSR are presented below with more detail.

Problem identification and motivation phase consist of targeted real-world problems that are identified and viewed in the scope of IS. Definitions should be done to problems that are either novel or expansive. This way the target of the study is expanding the knowledge base of the research field, not verifying, and viewing problems that are already been explored in previous research in the same nature.

Object definition phase presents the clear research targets as objectives, which requires an excellent awareness of current operating models and expertise in artifact's operation. These objectives can be quantitative, as in improvements of the latest features or actions of the artifact, or qualitative, as in expected behavior and performance of new artifacts, based on their roles derived from research problems.

Design and development phase acts as the most practical phase in DSR. Here the actual artifact is created in its respectful operating environment. The artifact is developed with the focus on prior design principles and theories. These are derived from previous research and industry's state-of-the-art practices.

During the presentation phase, the artifact is transitioned into a practical setting. Within this phase, the demonstration of the artifact's functionality is showcased to the appropriate audience. The artifact is depicted within its operational context, illustrating how it functions in its intended environment.

The evaluation phase is where the most attention should be paid in the design research. Within this phase is the opportunity to test the effectiveness of the approach on the case artifact. Evaluation is done to analyze artifact within the stated research objectives and if it aligns with the requirements and practicalities listed in the hypothesis. In our case, the evaluation is done by incorporating LLM outputs into real-life development scenarios within development team and analyzing the outcomes based on FEDS-framework (Venable et al., 2016).

Finally, the communication phase consists of revealing the study results to the research community and broader audience. To enlarge the knowledge base on design science, it is most important to share the results.

Thus, the study demonstrates the potential benefits of generative AI and analyzes the possibilities brought by generative AI based on the results, throughout the software development life cycle. Such intelligent development could be applied into agile methodologies, where software developers could have better knowledge base for system development (Perkusich et al., 2020).

Fundamental research framework of DSR is presented as follows: DSR is based on real-life phenomenon, where environment, consisting of people, organizations, and technology, provides a platform to research via business and organizational needs and targets. Knowledge base is acting as a growing database for best practices to be applied into future IS research. IS research presents the act of development and evaluation of the artifacts and theories. The framework is presented in Figure 4. (Hevner et al., 2004)

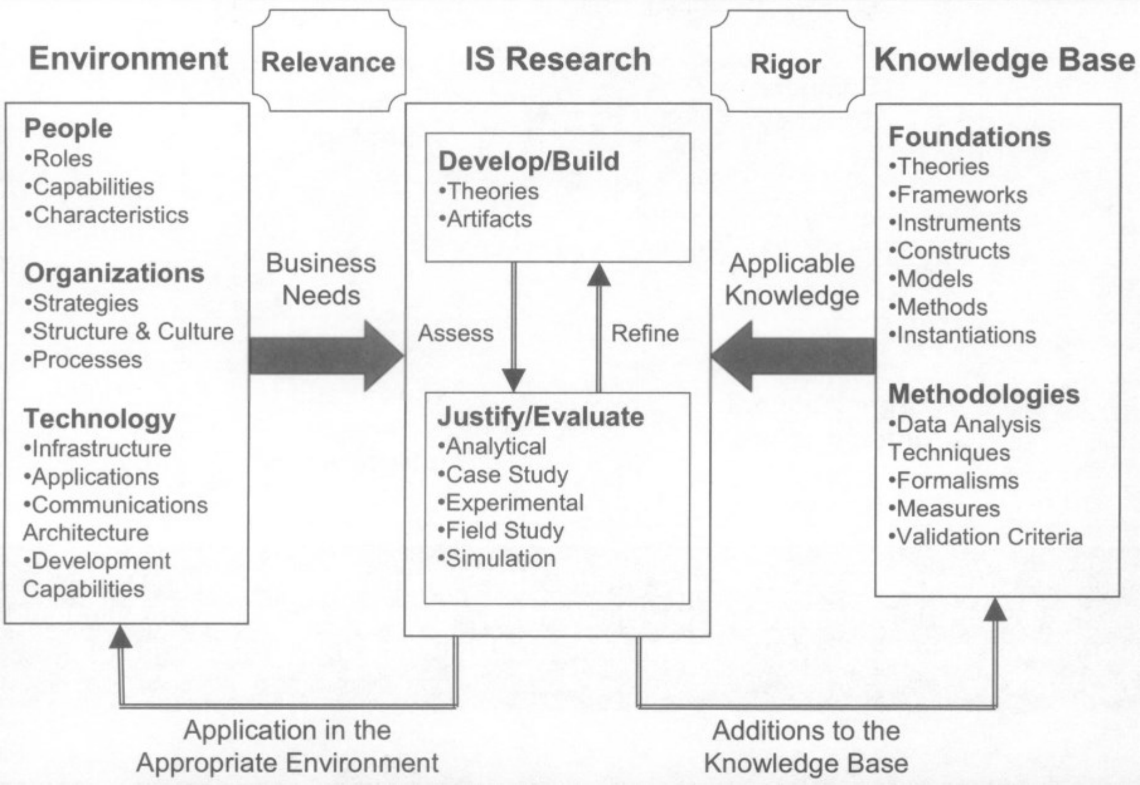


FIGURE 4 Design Science Research framework (Hevner et al., 2004)

With the DSR framework, the study harnessed a clear scientific framework that acts as the backbone for the empirical study. The core contribution to the knowledge base for our study is presented next.

Knowledge base consists of differing research aspects. With the design artifact developed in a DSR study, the research contributions can be divided into three levels according to their respective natures and maturity (Gregor & Hevner, 2013). The levels represent their abstraction type, varying from practical tools to highly applicable theories. These abstraction levels of artifacts are presented in Table 3.

TABLE 3 DSR Contribution Types (Gregor & Hevner, 2013)

	Contribution Types	Example Artifacts
More abstract, complete, and mature knowledge	Level 1: Situated implementation of artifact	Software products, applied processes
↕	Level 2: Nascent design theory—knowledge as operational principles/architecture	Frameworks, methods, design principles, models
More specific, limited, and less mature knowledge	Level 3: Well-developed design theory about embedded phenomena	Design theories

When viewing this thesis´ research setting and questions, we can divide these aspects into presented contribution types. RQ1 and RQ2 seek answers to the management and maintenance capabilities of LLM by exploring the usage of LLMs in software development processes. Thus, the outcome for RQ1 and RQ2 will be a particular type of new artifact in a form of method or framework for this kind of development. RQ3 is based on finding new solutions to minimize technical debt, thus determining a process artifact with specialized prompts. With RQ3, the situational factor of LLMs in the development workflow makes it an applied process. Hence, the outcomes from our design science research are as follows:

- RQ1 & RQ2: Level 2: Framework/method.
- RQ3: Level 1: Applied process

As IS research is practiced in natural environment and deals with practical systems and their life cycles, the thesis is dealing with IT artifacts. IT artifacts are traditionally divided into constructs, models, methods, and instantiations (March & Smith, 1995). Within the themes of this study, there are multiple IT artifacts dealt with. Artifacts vary based on their natures, consisting of existing software, development frameworks and methodologies. In Table 4, the IT artifacts present in the research setting are shown.

TABLE 4 IT artifacts

IT artifact	**Artifact type**	**Level**	**Properties**
Case system	Instantiation	Primary	- Data-heavy backend system
LLM	Model	Primary	- Multiple models
Development framework	Method	Primary	- Case study artifact: PESD framework
Development environment	Construct	Secondary	- VS Code - GitLab-server
User Interface for LLM	Construct	Secondary	- Text generation web UI
Version control system	Construct	Secondary	- Git
Programming languages	Construct	Secondary	- Typescript - NodeJS

From these IT artifacts, the case system, LLM, and development framework act as primary artifacts in the research setting. All the other IT artifacts listed act as secondary artifacts, supporting the research as enabling forces that make the software development environment actionable.

In the presentation phase, the applied designs, prompt inputs and example outputs are presented by considering the protected nature of the case company's protected data. The model's functionality is outlined in the phase as well as the framework of the developed artifact. We also argument on what decisions went through on the design choices of our input prompts and the whole workflow of the developed artifact.

The findings in this DSR study are communicated to the public as a part of JYX-library. JYX-library is a known research repository managed by the University of Jyväskylä. JYX consists of theses and dissertations in digital format.

4 CASE STUDY

This chapter deals with the development and evaluation of the study’s artifact. With the empirical setting and methodology described, we delve into the processes involved in creating and assessing the study's artifact. Finally, the evaluation of the development of the artifact and the study's results is presented.

4.1 Model selection

When choosing the right LLM for research purposes, one must consider the domain in which the model is used for. Models tend to be more efficient when they are fine-tuned or even pre-trained before using them in a scoped setting. These processes should be conducted rigorously, as training the model with second-rate datasets carries a significant risk of introducing bias and generating fictitious scenarios (Kaddour et al., 2023).

There are thousands of different models available through open-source with substantial amount of those that are fine-tuned by singular contributors with different techniques or purposes, with free open-source models being widely available for anyone to develop and utilize. Some examples of model formats are reduced memory and computation cost by quantized models (GPTQ method) and commodity hardware-run models (GGUF/GGML). GPTQ based models have been quantized to reduce memory and computing cost in developing and using these models (Frantar et al., 2023). With this, the quantization is done by representing the model weights with less accurate data types such as int16 rather than int64 (Hugging Face, 2023a). GGUF-models and formerly GGML-based models are also made to be run as quantized models with libraries such as llama.cpp, which enables models to be run on a CPU.

Generally, the size of LLMs is measured by the number of parameters they contain. The amount can vary largely from hundreds of billions to one billion parameters. For the models that are used in consumer hardware the quantity is commonly varying from 1, 7, 13, 30 to 65 billion. The GPUs and CPUs available in consumer hardware are capable of functioning with these sizes, although not comparing to actual cloud-based applications such as ChatGPT with its GPT-4 rumored to include as many as 1.8 trillion parameters (Schreiner, 2023). Due to the freedom to modify the software, access to the underlying software logic, and mostly absence of commercial software licenses, the thesis uses open-source models in the empirical part of the study.

During the design and development part of this thesis, we performed a search for compatible models for our use case. These models were selected based on their ability to be run locally on Nokia's own laboratory server computer containing NVIDIA T4 GPU with 16GB memory. Also, we, evaluated models with our Lenovo laptop setup which consisted of Windows 10 Enterprise operating system, Intel processor 11th Gen Intel(R) Core (TM) i7-1185G7 @ 3.00GHz with 4 Core(s), 8 Logical Processor(s), 16GB of RAM and no external or internal GUIs available. We first started our empirical study on a local laptop environment, but quickly realized that we needed more computing power for our LLMs to run smoothly with our codebase as we changed to the server computer environment.

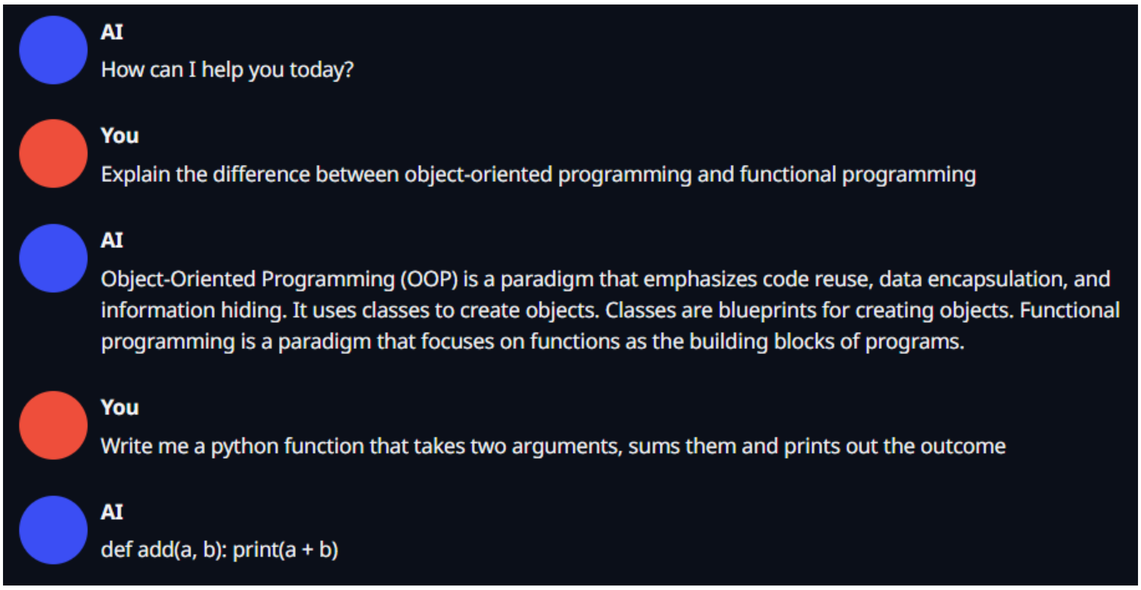
Models that were selected for the thesis included Vicuna, Code Llama, Dolphin, and WizardLM. Vicuna is a chat assistant that is derived from the original Llama by fine-tuning the model with user-shared conversations, targeting research on LLMs (LMSYS Org, 2023; LMSYS Org & Hugging Face, 2023). Code Llama was created by fine-tuning the Llama 2 model to better suit coding domain. The model is intended to use in code generation, coding completion, debugging, code related natural language tasks and other (Meta, 2023c; Meta & Hugging Face, 2023). Dolphin is a LLM that is based on MistralAI. Dolphin´s dataset is based on Microsoft´s Orca and is licensed with Apache-2.0 (Hartford & Hugging Face, 2023b). WizardLM is a LLM that is also based on the original Llama model, finetuned with complex open-domain instructions called Evol-Instruct (Hartford & Hugging Face, 2023a; Xu et al., 2023). These models were selected for the study for their open-source nature, state-of-the-art technology, and lightweight size. Also, the compared models have been listed in the Open LLM Leaderboard with high performance their weight classes (Hugging Face, 2023b). In our specific use case, we employ models consisting of 7B parameters. This choice enables these models to be executed more seamlessly within our computing environment. Table 5 lists models and their use cases.

TABLE 5 Compared LLMs

LLM	Model size	Training	Developer(s)
Vicuna	7B	Pre-trained & Fine-tuned	LMSYS Org
Code Llama	7B	Pre-trained & Fine-tuned	Meta
Dolphin	7B	Pre-trained	Hartford, Eric /Mistral AI
WizardLM	7B	Pre-trained	Hartford, Eric /WizardLM

Because of the research setting being highly private, we are using a local environment. This is due to the high risk in data leakage which is stated as one of the top risks in LLM implementation (OWASP, 2023). We decided to use open-source solution of Text Generation (TextGen) web UI due to its popularity, customizability and flexibility to run different models locally (oobabooga, 2022/2023). TextGen web UI can be downloaded as zip-file for all popular operating systems from Windows to Linux, macOS, and WSL. The application can also be installed via command line with additional settings such as CPU mode only.

To use the LLMs locally, they must be downloaded to be used via command line or in 3rd party software. Models can be used as they are, or model usage can be modified with parameter modifying. For example, max new tokens-parameter can be modified to assign the model a limit on how many new tokens a model can output. Free-to-use LLMs are openly available to be downloaded from AI communities such as Hugging Face where users can upload their own LLMs. Also, communities tend to rank open LLMs in their precision, availability, model size and other parameters for users to choose and evaluate their chosen model (Hugging Face, 2023b). These models tend to be modifications of base models, such as Llama or BERT, that are fine-tuned or formatted in distinct ways. TextGen web UI can download these models straight from the Hugging Face model repository, which enables user to experiment with different models and settings. To interact with the models, we utilized the chat functionality, which includes chat, chat-instruct, and instruct modes. We present example prompts with their respective outputs in appendix, with prompt example 1, 2, and 3. Figure 5 presents the chat view of the UI.



AI
How can I help you today?

You
Explain the difference between object-oriented programming and functional programming

AI
Object-Oriented Programming (OOP) is a paradigm that emphasizes code reuse, data encapsulation, and information hiding. It uses classes to create objects. Classes are blueprints for creating objects. Functional programming is a paradigm that focuses on functions as the building blocks of programs.

You
Write me a python function that takes two arguments, sums them and prints out the outcome

AI
def add(a, b): print(a + b)

FIGURE 5 TextGen web UI

In addition to the chat view, the TextGen web UI also features the default view and notebook view. The default view has widened input and output spaces. When pressing "generate," the model begins to extend the input text by predicting the next words as tokens. The same functionality is implemented in the notebook view, but the notebook view only contains one textbox.

4.2 Design and development

Below, we present the design and development of our study's design artifact. We motivate the development of the artifact with references to research and present the artifact along with its use cases and core design principles.

4.2.1 Foundations of the framework

Developmental steps taken for the design principles during the study were derived from a workflow that was presented in Chapter 2 within Figure 3. We divided the design into following steps.

For the foundational or base model that is intended to be AI assistant, an initial prompt is needed. This initial prompt describes the context and a role to the model itself. Example text can be something like this:

"You are a coding assistant specialized in C# and .NET development. You are expert in embedded systems with data-driven operations within video games."

Secondly, we provided the model a task. This task should be in line with the task that the developer is working on. Developer should describe the task as straightforward as possible. Example text can be something like this:

"Help me with one file that contains utilities for version check function."

Next, the actual problem target is submitted to the model itself. In our example, the source code of a single file is submitted to the model as an input. Example text can be something like this:

"Here is the content of the file: {content}. Help me refactor it to smaller components."

However, developer must keep in mind that if the input prompt is too large, the model might not have the computing power to process the prompt via the processing entity. For example, the GPU might not have the memory required to be allocated to the prompt processing.

After gaining outputs from the model, developer can seek to improve clarity and context with searching for more detailed information from the model itself. Outputs of the model should be evaluated based on developer´s desired or assumed outcome. Example text can be something like this:

"From the previous code snippet, what does function XYZ return? What would you do differently in this situation?"

4.2.2 Introducing PESD framework and prompt design principles

As prompt engineering requires input prompts from the user through the deployment of zero-shot, one-shot, or few-shot learning, users must devise solutions to gather information from prompt outputs. Prompt engineering appears to be relatively casual when users type in their inputs to generative AI applications. This poses a problem when conducting a rigorous search for information on selected topics from the target artifact: one input could yield separate outputs than other inputs, even if they target the same artifact component. This can lead to misinformation in various situations, for example, if the artifact is reviewed by multiple developers or the prompts contain unintuitive text for humans (Kaddour et al., 2023; Mökander et al., 2023). Research has shown interest at presenting prompt patterns and specific domain frameworks within prompt engineering (Lo, 2023; White et al., 2023). A set of definite design principles are also needed, leading to better inputs. Thus, a general framework for prompt engineering should be established to standardize prompt inputs, clarify the prompt engineering workflow, and enable the rigorous evaluation of LLMs.

With this in mind, we present prompt engineering software development (PESD) framework. PESD can be used for various tasks in software development, such as test generation, refactoring, or test generation. Also, it can be used to evaluate the truthfulness, context-awareness, and accuracy of the used LLM. The framework is presented in Figure 6.

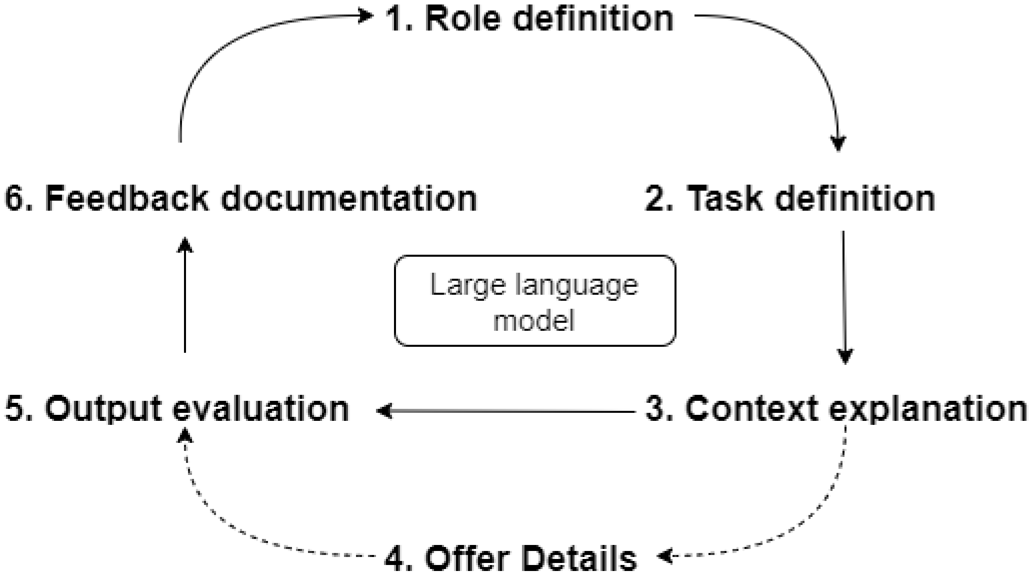


FIGURE 6 PESD framework

The phases of the framework represent the workflow of intelligent development where software development meets AI through usage of LLM. Developer starts with role definition, where a role is assigned to the model itself, such as test engineer expert, backend specialist, or similar. Next, the task is defined for the model with the context presented, for example as in code function, test file or an error message. If the output does not meet expectations with high quality, the developer offers more details within the context of the prompt until the expectations are met. After satisfying outputs are provided, the developer documents the process and the outputs by evaluating the chosen model and its level of success in the process. This way the development team can have insights whether the model is useful for their purposes or not. The PESD framework contributes to the knowledge base as a framework referenced within the contribution types of RQ1 and RQ2.

The framework is primarily targeted to be used with minor LLMs in parameter size such as 13B, 7B and smaller models, although the principles can be applied to more powered LLMs, like GPT-3.5 As such, framework suits well in local environment development or similar, where computing power is limited.

Utilizing the PESD framework, we conducted an empirical study within the actual development environment. Initially, we assessed the performance of various models when subjected to standardized prompts, subsequently comparing the resulting outputs. This process enabled us to select the model for deployment in the empirical phase of our study. We continued inputting the same prompts to the models until our GPU's VRAM was exhausted. This way, we gained an understanding of which of the tested models were most suitable for our development environment. Prompt design principles acts in prompts with few-shot learning prompts, having more depth than zero-shot learning prompts. The design principles contribute to the knowledge base as applied processes referenced within the contribution type of RQ3. The prompts that we used are shown in Table 6.

TABLE 6 Prompt design principles with PESD framework

Phase	Principles	Prompt example
1. Role definition	- Narrow the role for the use case. - List artifacts that are dealt within the role. - Define the target of the role, if necessary.	*You are a coding assistant specialized in Typescript and node.js development. You are an expert with a large software system that is business-critical, contains hard to maintain code and has technical debt.*
2. Task definition	- Communicate the task as if addressing an expert.	*Analyze a component X that has entities Y and Z.*
3. Context explanation	- Focus on clarity. - Ask about the model´s suggestions.	*Here is the content of the component: {context}. What would you modify to increase maintainability? Show me your modifications.*
4. Offer details	- Emphasize attention on past outputs: what was not provided with them?	*What does entity Z do? What would you do differently with it?*

These prompts can be employed as individual prompts or combined into one, depending on the specific task at hand. As the Table 6 present, there are no prompt inputted into the model in phases 5 and 6. Phases 5 and 6 are evaluation based that are needed to weigh in the value of the tested model. Chosen models for the study were used in search of systematic maintenance enhancements with these same prompts used in analysis of various software components. All these models were run with the same parameters in the TextGen web UI, with attention to the following: the maximum amount of new output tokens was set to 1455, temperature was set to 0.7, and "truncate the prompt up to this length" was set to 2048. The maximum output tokens represent the output length of the model, temperature represents the creativity of the model´s outputs, and "truncate the prompt up to this length" represents the limit at which the leftmost tokens are removed if the prompt exceeds the given length.

As test targets, we used three files that each had varying functions and components aimed at prechecking specific software versions and conditions in given software component. Files were selected based on their differing complexities and relevancy in the case system´s functionality. First file was smaller with around 20 lines of code, containing only one main function with two subfunctions. Second file had more complexity in it, with 57 lines of code, containing more subfunctions. Third file had the most complexity with 162 lines of code with functions varying in their nature. On average, we had six prompt inputs within our chats, with four being initial prompts and two involving detailed inquiries.

From this, we analyzed a total of 36 chats, divided among four models, with three files and three chats each. With three chats repeated within each file, we were able to validate each chat to minimize anomalous results.

4.3 Evaluation and results

Evaluation acts as the core phase of DSR. In this phase, the DSR is acting as a scientific method to evaluate new artifacts with rigor and relevance. Evaluation makes sure that the study acts on high quality scientific standards, is harnessed to criticism and scrutiny and meets the practical utilities to enlarge knowledge base that was set for the artifact itself. Hence, the dual purpose of evaluation should be done with relevancy and rigor, mirroring on behavioral science theories, if the methodology is to be labeled as science (Hevner et al., 2004; Venable et al., 2016).

The evaluation process can be approached through various methodologies, one of which is the Framework for Evaluation in Design Science (FEDS)-framework. This framework offers a comprehensive evaluation structure, comprising four distinct facets tailored to the specific characteristics of the artifact under examination (Venable et al., 2016). Withdrawing from FEDS-framework, the evaluation of the case study is going to have multiple episodes based on the evaluation strategy. Strategies are presented in Figure 7.

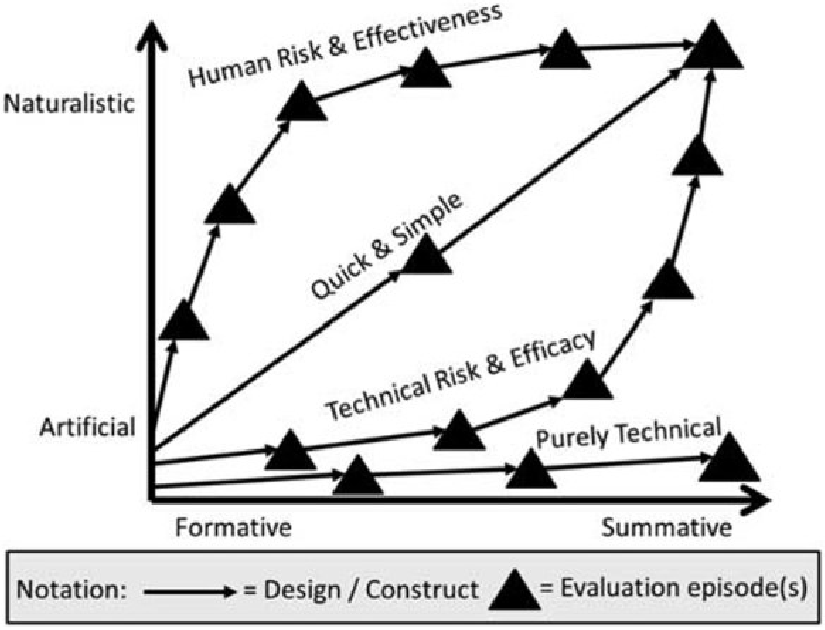

Human Risk & Effectiveness
Naturalistic
Quick & Simple
Technical Risk & Efficacy
Artificial
Purely Technical
Formative
Summative
Notation: → = Design / Construct ▲ = Evaluation episode(s)

FIGURE 7 FEDS framework (Venable et al., 2016)

FEDS consist of four steps that are (1) explicate the goals of the evaluation, (2) choose the evaluation strategy or strategies, (3) determine the properties to evaluate, and (4) design the individual evaluation episode(s). In phase 1, we must explicit goals with four different goals in mind. Goals are achieved in varying phases of the evaluation process and can be iteratively reviewed. Iterative process enables the modifications of the artifact elements based on the evaluation process. These goals are presented in Table 7.

TABLE 7 Evaluation goals, objectives and artifact elements

Goal	Objective	Artifact elements
Rigor	Efficacy and effectiveness: - Outcome caused by the artifact - Artifact works in real life situations	- Case system´s code base acts as the target of real-life scenario, we do not test our framework with example code bases - Analysis is only done via LLM as non-human agent - Interpretation of outputs is done via human developer
Uncertainty and risk reduction	Human/social risks & technical risks: - Early detection - Support higher quality	- Risk of data leakage in PoC-project is minimized with laboratory server environment - Risk of data corruption is increased when fine-tuning the model with flawed or erroneous code base
Ethics	Safety and risks for environment, humans, animals etc.	- Increased power consumption - Evaluation is not putting researchers at risk
Efficiency	Goals compared to resources: - Time - Money	- Resources were sufficient for PoC-project. - For larger projects, more computing power via higher amount of GPUs or private cloud would be necessary - The study was done without strict deadlines

In phase 2, we choose evaluation strategy to begin with. Given the technical complexities and the human-centric management inherent in the utilization of LLMs within software system environments, our evaluation strategy aligns with the Technical Risk & Efficacy approach as delineated within the FEDS-framework. This is due to the artifact being used by human users in highly technical settings, initiating the evaluation from artificial environment towards more naturalistic one. Evaluation is also heavily reliant on the technical capacity and limitations of LLMs, contributing to the decision to proceed with this strategy.

In phase 3, we define properties to be evaluated. These properties can vary from features and applications to requirements and design goals. The primary objective is to ensure that these properties are unique and valid to the artifact's usage environment. The properties to be evaluated in our LLM environment are as follows:

- *Validity*: the model outputs correct answers depending on the context
- *Value*: the model outputs contribute real value to software maintenance
- *Competence*: the ability to process complex inputs from the user
- *Convenience*: the models are easy to setup in LLM application

Finally with properties set, phase 4 introduces the design of evaluation episodes. These episodes focus on the actual evaluation of the artifact: episodes are presented in Figure 7 as triangles that start from formative/artificial and finish as summative/naturalistic. Within this phase, researchers should consider following key points: environmental constraints, analysis of goals in evaluation and evaluation episode quantity and order of execution. As indicated in Table 4, the primary IT artifacts in our research settings were the case system, LLMs, and our development framework. Environmental constraints for these artifacts consist of resource intensity and scalability, especially concerning the LLM itself. With the case system comprising hundreds of files, we had to restrict our empirical study to three different files, varying in complexity, domain, and functionality. Our evaluation goals were analyzed and organized from broadest to most detailed: we first evaluated our computing environment, then individual models, and lastly our prompts. Subsequently, we assessed our framework since all other environments had already been examined and evaluated. This way, we could minimize resource intensity by optimizing our input prompts and scaling up the framework for even larger prompts.

We employed the Technical Risk & Efficacy strategy, which paved the way for us to mitigate technical risks through early evaluations of properties within our LLM environment. We noticed relatively early that our local laptop environment would not be powerful enough for the optimal usage of LLMs. With this, we transformed our environment by switching to laboratory server computer with significantly more computing resources via remote access of case company´s own private network. Therefore, we could improve our research environment and hence test our prompt design principles even better.

To determine the most suitable model for our research, we conducted a small comparative test of different models using PESD framework, as noted in "Model selection" subchapter. This allowed us to choose the model based on our own experiment in addition to the external reviews within AI communities.

Prompt evaluation process enabled us to identify the optimal initial models that were used in our use case. Also, we modified our framework artifact with the addition of "Output evaluation" and "Offer details" to gain more valuable outputs from the models themselves. The evaluation episodes within our empirical study are presented in Table 8.

TABLE 8 Evaluation episodes

Episodes and goals with order of execution	Objective	Actions
1. Environment evaluation	Check the preconditions for optimal model usage	- Test the local application´s power with the LLMs - Change local environment to the lab server environment
2. Model evaluation	Check model output validity and value	- Conduct the empirical study with standardized prompt principles - Mirror the outputs on the files processed - Change models according to their context validity and real value of outputs
3. Prompt evaluation	Check that the prompts used are optimal and targeted for use cases	- Test prompt designs with the PESD framework - Change prompt designs according to outputs and GPU memory allocation limits
4. Framework evaluation	Test the framework phases in varying situations	- Modify framework phases after selection of the model and initial prompts

After these episodes, we came up with further development of PESD framework that had improvements in model selection, prompt design, and in its phases. We changed from Falcon LLM by Technology Innovation Institute of Abu Dhabi (TII) to WizardLM when comparing their outputs within our standardized prompts and came to conclusion that WizardLM would suit our research setting better (Technology Innovation Institute, 2023). With the final framework established, we begun to systematically gather data as chats with the designed prompts for PESD framework.

Here, we present the results of our empirical study, derived from prompt engineering chats created using the PESD framework. Chats were saved as JSON files, containing input prompts from users and the corresponding output answers from the model. These chats are analyzed based on four properties outlined in this chapter: validity, value, competence, and consistency. Validity assesses the relevance of the context in the chat, while value evaluates whether the model's outputs bring value to the user's needs. Competence gauges the model's ability to handle complex inputs with logic, and consistency assesses the ease of setting up and using the model.

As stated, the analysis comprises 36 chats that were individually analyzed. The analysis was conducted with the perspective that the target user was a junior or trainee-level developer. Due to data security concerns, we do not display these chats with the results. We summarized the results using mean values to simplify the findings with overall value. The summarized results are presented in Table 9, with more detailed information available in appendix Table 10.

TABLE 9 Results

LLM	**Validity** (High, mid, low, none)	**Value** (High, mid, low, none)	**Competence** (High, mid, low, none)	**Convenience** (High, mid, low, none)

Vicuna	High = 6
	Mid = 2
	Low = 1
	None = 0
	Overall: High
		High = 1
		Mid = 4
		Low = 3
		None = 1
		Overall: Mid
			High = 3
			Mid = 3
			Low = 3
			None = 0
			Overall: Mid
				High = 9
				Mid = 0
				Low = 0
				None = 0
				Overall: High

Code Llama	High = 4
	Mid = 4
	Low = 1
	None = 0
	Overall: Mid
		High = 0
		Mid = 4
		Low = 4
		None = 1
		Overall: Low
			High = 4
			Mid = 4
			Low = 1
			None = 0
			Overall: Mid
				High = 9
				Mid = 0
				Low = 0
				None = 0
				Overall: High

Dolphin	High = 2
	Mid = 4
	Low = 3
	None = 0
	Overall: Mid
		High = 0
		Mid = 2
		Low = 6
		None = 1
		Overall: Low
			High = 1
			Mid = 5
			Low = 3
			None = 0
			Overall: Mid
				High = 9
				Mid = 0
				Low = 0
				None = 0
				Overall: High

WizardLM	High = 2
	Mid = 4
	Low = 2
	None = 1
	Overall: Mid
		High = 0
		Mid = 1
		Low = 2
		None = 6
		Overall: None
			High = 1
			Mid = 3
			Low = 2
			None = 3
			Overall: Low
				High = 9
				Mid = 0
				Low = 0
				None = 0
				Overall: High

5 DISCUSSION

This chapter summarizes the research results and reflects them in relation to the prior research. Future implications and outlooks are discussed, and the practical implications are noted.

As evident from the results, the analysis of individual properties reveals the capabilities of the models. Among all the properties, convenience had the highest score across all the models, as the setup process was easy and straightforward with each model. Second was validity: all the models were able to interpret the context of inputs and questions, scoring at least at an intermediate level. The third property was competence, where all the models scored at an intermediate level. Their ability to comprehend the relative complexity of the inputs was impressive, although the outputs did not consistently present highly relevant value. The value property scored the least, as outputs varied from erratic to value-generating ones. The most accurate observation was that when the size and complexity of the target file increased, the inconsistency of outputs also increased.

Overall, the Vicuna model demonstrated the most success with the set of properties, averaging high in validity, mid in value, mid in competence, and high in convenience. Code Llama secured the second position in terms of model usage success, followed by Dolphin in the third position and WizardLM in the fourth and last position.

5.1 Revisiting research questions

Here we reflect the research questions with the empirical results.

RQ1: How to systematically evaluate LLM outputs in software maintenance?

> *Hypothesis*: It is feasible to construct a framework that facilitates objective comparison of outcomes.

Findings suggest that an effective framework was possible to be established. The effectiveness of the PESD framework was evaluated through a rigorous process with standardized prompt inputs. This way, we discovered facilitated comparison of prompt outputs, which provided insights into the properties of validity, value, competence, and convenience of four different LLMs. The PESD framework presents clear phases on how to systematically evaluate LLMs with explicit prompt design principles. Thus, the findings align with our hypothesis.

RQ2: How can generative artificial intelligence help software developers in system management and maintenance?

> *Hypothesis*: Generative artificial intelligence can help systematically when organizing system development points and management.

Findings suggest that, while maintenance analysis can be systematically conducted with small LLMs, the analysis itself is modest. Nevertheless, the models consistently performed well within their outputs, offering suggestions on how to modify and advance software maintenance within the inputted files. These suggestions were consistent across different models, averaging an intermediate score within differing models. Thus, the PESD framework enables researchers and developers to systematically gain knowledge on the management, maintenance, and development of case artifacts. The findings align with our hypothesis, although not as strongly as anticipated.

RQ3: Can generative artificial intelligence manage development to reduce technical debt?

> *Hypothesis*: Technical debt can be detected in the system using generative artificial intelligence, but a larger scope of technical debt cannot be identified.

Findings suggest that with prompt engineering using the PESD framework within smaller LLMs, we do not obtain advanced information on technical debt within the case system. Identifying technical debt is typically challenging, and the analysis on singular files within our research timeframe makes it even more difficult. The environmental constraint of our case system's size became evident when conducting the empirical study. In the future, the usage of different LLMs with multiple users and a larger quantity of varying files could reveal technical debt over time. The findings align moderately with our hypothesis.

5.2 Implications and future research

The results of the empirical study suggest diverse implications for scientific and practical targets regarding LLM usage. Our findings propose that developers could leverage LLMs for their usage with the PESD framework as a standardized way to gather knowledge on case artifacts rather than singular prompts. Prompt engineering works best when analyzing singular functions or files in their development system, not whole systems. The PESD framework enables developers and researchers to use and evaluate their chosen LLMs and the outputs referred to their own use cases. Prompt design principles provide practical ways to implement personalized prompts with advanced outputs rather than relying on zero-shot learning prompts.

Software maintenance could be enhanced through the systematic application of the PESD framework when analyzing individual files. Moreover, the learning process for a new developer is improved by incorporating prompt engineering into novel software components unfamiliar to the developer. However, more extensive research, involving larger models, different systems, and a greater quantity of prompts, is necessary for significant advancements in larger-scale maintenance. In the future, organizations will harness the power of LLMs through specialized applications, leveraging from chatbots to automated code reviews, powered by entities such as OpenAI’s Assistant API (OpenAI, 2023). With these applications, organizations can advance their operations in customized situations such as onboarding to the codebase. This will require systematic approaches, ones that we have presented here as prompt design principles and a prompt engineering software development framework.

5.3 Limitations

Studying a system with AI used in the industry comes with limitations. As LLM-based application like ChatGPT have proposed several security vulnerabilities for organizations, the data leaks have lead organizations to prohibit the usage of these applications and begin to create their own applications (Bloomberg, 2023; Reuters, 2023). From this, organizations need to develop their LLMs with cloud solutions or internal server computers, both which can lead to cost-heavy solutions with larger LLMs requiring unique competence, especially with pretraining (Dwivedi et al., 2023; Kaddour et al., 2023). Thus, smaller LLMs could fit more properly with an organization's needs, especially when using the models only for smaller or general tasks. However, choosing smaller models leads to a significant loss in accuracy and prediction naturally.

LLMs are generally run in cloud environments or via GPUs due to their large computing requirements. However, CPUs can also be used when running LLMs, although it is certainly slower and more power-consuming than GPUs or applications running in cloud environments. Since the private nature of the target system led the study to use a local environment and smaller LLMs, lower computational power was utilized. Also, we only had a single researcher working on the empirical study with specific files; we could have acquired different and more versatile data if more than one researcher had analyzed model behavior on software maintenance.

With the case system being a backend system with specific programming languages and paradigms, we could have obtained different data if we had analyzed different systems containing components in the frontend or databases. Also, the laboratory server solution, with its notable configurations and installation processes, performs poorly when scaled for a larger number of developers, as the GPU quantity would rise rapidly. The NVIDIA T4 GPU that we used likewise had limitations in CUDA VRAM and computing power when allocating memory in its usage. Therefore, a cloud-based solution would be preferable in the future.

6 CONCLUSION

In conclusion, this thesis delves into the dynamic intersection of generative AI and software development, emphasizing the continuous evolution and adaptation in response to technological advancements.

The challenges faced in maintaining large and complex software systems demand innovative solutions. As a response to maintenance challenges, such as complexity, costs, and technical debt, this thesis focuses on the potential of generative AI to provide systematic refinements in software product maintenance.

Generative AI, particularly in the context of software onboarding and supporting developers, holds promise for introducing new concepts, proposing repair suggestions, and offering sophisticated examples related to the application area and software components. The thesis aims to contribute to the development of more efficient and systematic approaches to software maintenance via LLMs.

The empirical study conducted as part of this thesis seeked to uncover insights and innovative solutions for enhancing the maintenance of software systems through prompt engineering. As a result, we present PESD framework for systematic prompt engineering with design principles for singular prompts. With PESD framework, we conducted an evaluation with 36 different chats with 4 different LLMs. Findings suggest that the PESD framework produced consisted outputs with average of intermediate value. The PESD framework and prompt design principles contribute to the knowledge base as a framework and applied processes with contribution types aligned with the research questions.

As we move forward in the ever-evolving landscape of software development, the findings from this study contribute to the ongoing discourse on optimizing software processes. The potential applications of generative AI in this context open avenues for more efficient, cost-effective, and innovative software practices, ultimately advancing the field of software development.

REFERENCES

Akter, S., McCarthy, G., Sajib, S., Michael, K., Dwivedi, Y. K., D'Ambra, J., & Shen, K. N. (2021). Algorithmic bias in data-driven innovation in the age of AI. *International Journal of Information Management*, *60*, 102387. https://doi.org/10.1016/j.ijinfomgt.2021.102387

Barth, A., Chambers, M., Eigenbrode, S., & Fregly, C. (2023, June). *Generative AI project lifecycle*. Coursera. https://www.coursera.org/learn/generative-ai-with-llms/home/welcome

Berente, N., Gu, B., Recker, J., & Santhanam, R. (2021). *Managing artificial intelligence*.

Berthelot, D., Carlini, N., Goodfellow, I., Papernot, N., Oliver, A., & Raffel, C. A. (2019). MixMatch: A Holistic Approach to Semi-Supervised Learning. *Advances in Neural Information Processing Systems*, *32*. https://proceedings.neurips.cc/paper_files/paper/2019/hash/1cd138d0499a68f4bb72bee04bbec2d7-Abstract.html

Bloomberg. (2023, May 1). *Samsung Bans ChatGPT, Google Bard, Other Generative AI Use by Staff After Leak*. Bloomberg. https://www.bloomberg.com/news/articles/2023-05-02/samsung-bans-chatgpt-and-other-generative-ai-use-by-staff-after-leak?in_source=embedded-checkout-banner

Boehm, B. W. (1976). Software Engineering. *IEEE Transactions on Computers, 25*(12), 1226–1241. https://doi.org/10.1109/TC.1976.1674590

Bogner, J., Fritzsch, J., Wagner, S., & Zimmermann, A. (2018). Limiting technical debt with maintainability assurance: An industry survey on used techniques and differences with service- and microservice-based systems. *Proceedings of the 2018 International Conference on Technical Debt*, 125–133. https://doi.org/10.1145/3194164.3194166

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., … Liang, P. (2022). *On the Opportunities and Risks of Foundation Models* (arXiv:2108.07258). arXiv. https://doi.org/10.48550/arXiv.2108.07258

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., … Amodei, D. (2020). Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, *33*, 1877–1901.

Bull, C., & Kharrufa, A. (2023). *Generative AI Assistants in Software Development Education* (arXiv:2303.13936). arXiv. https://doi.org/10.48550/arXiv.2303.13936

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., … Fiedel, N. (2022). *PaLM: Scaling Language Modeling with Pathways* (arXiv:2204.02311). arXiv. https://doi.org/10.48550/arXiv.2204.02311

Cunningham, W. (1993). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*, *4*(2), 29–30. https://doi.org/10.1145/157710.157715

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding* (arXiv:1810.04805). arXiv. https://doi.org/10.48550/arXiv.1810.04805

Douglas Heaven, W. (2023, May 12). *The open-source AI boom is built on Big Tech's handouts. How long will it last?* MIT Technology Review. https://www.technologyreview.com/2023/05/12/1072950/open-source-ai-google-openai-eleuther-meta/

Dwivedi, Y. K., Kshetri, N., Hughes, L., Slade, E. L., Jeyaraj, A., Kar, A. K., Baabdullah, A. M., Koohang, A., Raghavan, V., Ahuja, M., Albanna, H., Albashrawi, M. A., Al-Busaidi, A. S., Balakrishnan, J., Barlette, Y., Basu, S., Bose, I., Brooks, L., Buhalis, D., … Wright, R. (2023). Opinion Paper: "So what if ChatGPT wrote it?" Multidisciplinary perspectives on opportunities, challenges and implications of generative conversational AI for research, practice and policy. *International Journal of Information Management*, *71*, 102642. https://doi.org/10.1016/j.ijinfomgt.2023.102642

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). DevOps. *IEEE Software*, *33*(3), 94–100. https://doi.org/10.1109/MS.2016.68

European Commission. Joint Research Centre. (2020). *Estimating investments in general purpose technologies: The case of AI investments in Europe.* Publications Office. https://data.europa.eu/doi/10.2760/506947

Feng, S., Park, C. Y., Liu, Y., & Tsvetkov, Y. (2023). *From Pretraining Data to Language Models to Downstream Tasks: Tracking the Trails of Political Biases Leading to Unfair NLP Models* (arXiv:2305.08283). arXiv. https://doi.org/10.48550/arXiv.2305.08283

Finnegan, M. (2023, May 23). *Microsoft's M365 Copilot AI assistant gets third-party app integrations*. Computerworld. https://www.computerworld.com/article/3696979/microsoft-s-m365-copilot-ai-assistant-gets-third-party-app-integrations.html

Frantar, E., Ashkboos, S., Hoefler, T., & Alistarh, D. (2023). *GPTQ: Accurate Post-Training Quantization for Generative Pre-trained Transformers* (arXiv:2210.17323). arXiv. https://doi.org/10.48550/arXiv.2210.17323

Freitag, C., Berners-Lee, M., Widdicks, K., Knowles, B., Blair, G. S., & Friday, A. (2021). The real climate and transformative impact of ICT: A critique of estimates, trends, and regulations. *Patterns*, *2*(9), 100340. https://doi.org/10.1016/j.patter.2021.100340

Fung, B. (2023, August 6). *The big bottleneck for AI: A shortage of powerful chips | CNN Business*. CNN. https://www.cnn.com/2023/08/06/tech/ai-chips-supply-chain/index.html

Ghahramani, Zoubin, & Google. (2023, May 10). *Introducing PaLM 2*. Google. https://blog.google/technology/ai/google-palm-2-ai-large-language-model/

GlobalData. (2023, August 29). ChatGPT dominates LLM mentions on social media in 2023, Bards distant second, reveals GlobalData. *GlobalData, Business Fundamentals*. https://www.globaldata.com/media/business-fundamentals/chatgpt-dominates-llm-mentions-on-social-media-in-2023-bards-distant-second-reveals-globaldata/

Goldberg, S. B., Flemotomos, N., Martinez, V. R., Tanana, M. J., Kuo, P. B., Pace, B. T., Villatte, J. L., Georgiou, P. G., Van Epps, J., Imel, Z. E., Narayanan, S. S., & Atkins, D. C. (2020). Machine learning and natural language processing in psychotherapy research: Alliance as example use case. *Journal of Counseling Psychology*, *67*(4), 438–448. https://doi.org/10.1037/cou0000382

Gregor, S., & Hevner, A. (2013). Positioning and Presenting Design Science Research for Maximum Impact. *MIS Quarterly*, *37*, 337–356. https://doi.org/10.25300/MISQ/2013/37.2.01

Haenlein, M., & Kaplan, A. (2019). A Brief History of Artificial Intelligence: On the Past, Present, and Future of Artificial Intelligence. *California Management Review*, *61*(4), 5–14. https://doi.org/10.1177/0008125619864925

Hartford, E., & Hugging Face. (2023a, April 28). *Ehartford/WizardLM-7B-Uncensored · Hugging Face*. https://huggingface.co/ehartford/WizardLM-7B-Uncensored

Hartford, E., & Hugging Face. (2023b, September 18). *Ehartford/dolphin-2.1-mistral-7b · Hugging Face*. https://huggingface.co/ehartford/dolphin-2.1-mistral-7b

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Howard, J., & Ruder, S. (2018). *Universal Language Model Fine-tuning for Text Classification* (arXiv:1801.06146). arXiv. https://doi.org/10.48550/arXiv.1801.06146

Hugging Face. (2023a, September 22). *Quantization*. https://huggingface.co/docs/optimum/concept_guides/quantization

Hugging Face. (2023b, October 17). *Open LLM Leaderboard—A Hugging Face Space by HuggingFaceH4*. https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard

Jacobson, I., Sutherland, J., Kerr, B., & Buhnova, B. (2022). Better Scrum through Essence. *Software: Practice and Experience*, *52*(6), 1531–1540. https://doi.org/10.1002/spe.3070

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., & Fung, P. (2023). Survey of Hallucination in Natural Language Generation. *ACM Computing Surveys*, *55*(12), 248:1-248:38. https://doi.org/10.1145/3571730

Kaddour, J., Harris, J., Mozes, M., Bradley, H., Raileanu, R., & McHardy, R. (2023). *Challenges and Applications of Large Language Models* (arXiv:2307.10169). arXiv. https://doi.org/10.48550/arXiv.2307.10169

Kalyan, K. S., Rajasekharan, A., & Sangeetha, S. (2021). *AMMUS: A Survey of Transformer-based Pretrained Models in Natural Language Processing* (arXiv:2108.05542). arXiv. https://doi.org/10.48550/arXiv.2108.05542

Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., & Shah, M. (2022). Transformers in Vision: A Survey. *ACM Computing Surveys*, *54*(10s), 200:1-200:41. https://doi.org/10.1145/3505244

Khashabi, D., Lyu, S., Min, S., Qin, L., Richardson, K., Welleck, S., Hajishirzi, H., Khot, T., Sabharwal, A., Singh, S., & Choi, Y. (2022). *Prompt Waywardness: The Curious Case of Discretized Interpretation of Continuous Prompts* (arXiv:2112.08348). arXiv. http://arxiv.org/abs/2112.08348

Koskinen, J. (2004, February 2). *Software Maintenance Costs*. Archive.Ph. https://archive.ph/oBlIr

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), Article 7553. https://doi.org/10.1038/nature14539

Lester, B., Al-Rfou, R., & Constant, N. (2021). *The Power of Scale for Parameter-Efficient Prompt Tuning* (arXiv:2104.08691). arXiv. https://doi.org/10.48550/arXiv.2104.08691

Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, *101*, 193–220. https://doi.org/10.1016/j.jss.2014.12.027

Li, Z., Liang, P., & Avgeriou, P. (2014). Chapter 9—Architectural Debt Management in Value-Oriented Architecting. In I. Mistrik, R. Bahsoon, R.

Kazman, & Y. Zhang (Eds.), *Economics-Driven Software Architecture* (pp. 183–204). Morgan Kaufmann. https://doi.org/10.1016/B978-0-12-410464-8.00009-X

Lientz, B. P., Swanson, E. B., & Tompkins, G. E. (1978). Characteristics of application software maintenance. *Communications of the ACM*, *21*(6), 466–471. https://doi.org/10.1145/359511.359522

LMSYS Org. (2023, March 30). *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%* ChatGPT Quality | LMSYS Org*. https://lmsys.org/blog/2023-03-30-vicuna

LMSYS Org, & Hugging Face. (2023, November 22). *Lmsys/vicuna-7b-v1.3 · Hugging Face*. https://huggingface.co/lmsys/vicuna-7b-v1.3

Lo, L. S. (2023). The CLEAR path: A framework for enhancing information literacy through prompt engineering. *The Journal of Academic Librarianship*, *49*(4), 102720. https://doi.org/10.1016/j.acalib.2023.102720

Maeder, A. (2023, January 26). *AI/ML unleashes the full potential of 5G-Advanced*. Nokia. https://www.nokia.com/blog/aiml-unleashes-the-full-potential-of-5g-advanced/

Mäkitalo, N., Taivalsaari, A., Kiviluoto, A., Mikkonen, T., & Capilla, R. (2020). On opportunistic software reuse. *Computing*, *102*(11), 2385–2408. https://doi.org/10.1007/s00607-020-00833-6

Manning, C. D. (2015). Computational Linguistics and Deep Learning. *Computational Linguistics*, *41*(4), 701–707. https://doi.org/10.1162/COLI_a_00239

March, S. T., & Smith, G. F. (1995). Design and natural science research on information technology. *Decision Support Systems*, *15*(4), 251–266. https://doi.org/10.1016/0167-9236(94)00041-2

Melo, A., Fagundes, R., Lenarduzzi, V., & Santos, W. B. (2022). Identification and measurement of Requirements Technical Debt in software development: A systematic literature review. *Journal of Systems and Software*, *194*, 111483. https://doi.org/10.1016/j.jss.2022.111483

Meta. (2023a, February 24). *Introducing LLaMA: A foundational, 65-billion-parameter language model*. https://ai.facebook.com/blog/large-language-model-llama-meta-ai/

Meta. (2023b, July 18). Meta and Microsoft Introduce the Next Generation of Llama. *Meta*. https://about.fb.com/news/2023/07/llama-2/

Meta. (2023c, August 24). Introducing Code Llama, an AI Tool for Coding. *Meta*. https://about.fb.com/news/2023/08/code-llama-ai-for-coding/

Meta, & Hugging Face. (2023, November 22). *Codellama/CodeLlama-7b-Instruct-hf · Hugging Face*. https://huggingface.co/codellama/CodeLlama-7b-Instruct-hf

Min, B., Ross, H., Sulem, E., Veyseh, A. P. B., Nguyen, T. H., Sainz, O., Agirre, E., Heintz, I., & Roth, D. (2023). Recent Advances in Natural Language Processing via Large Pre-Trained Language Models: A Survey. *ACM Computing Surveys*. https://doi.org/10.1145/3605943

Mökander, J., Schuett, J., Kirk, H. R., & Floridi, L. (2023). Auditing large language models: A three-layered approach. *AI and Ethics*. https://doi.org/10.1007/s43681-023-00289-2

Nadkarni, P. M., Ohno-Machado, L., & Chapman, W. W. (2011). Natural language processing: An introduction. *Journal of the American Medical Informatics Association*, *18*(5), 544–551. https://doi.org/10.1136/amiajnl-2011-000464

oobabooga. (2023). *Text generation web UI* [Python]. https://github.com/oobabooga/text-generation-webui (Original work published 2022)

OpenAI. (2022, November 30). *Introducing ChatGPT*. https://openai.com/blog/chatgpt

OpenAI. (2023, June 11). *New models and developer products announced at DevDay*. https://openai.com/blog/new-models-and-developer-products-announced-at-devday

OWASP. (2023, August 26). *OWASP Top 10 for Large Language Model Applications | OWASP Foundation*. https://owasp.org/www-project-top-10-for-large-language-model-applications/

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A Design Science Research Methodology for Information Systems Research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Perkusich, M., Chaves e Silva, L., Costa, A., Ramos, F., Saraiva, R., Freire, A., Dilorenzo, E., Dantas, E., Santos, D., Gorgônio, K., Almeida, H., & Perkusich, A. (2020). Intelligent software engineering in the context of agile software development: A systematic literature review. *Information and Software Technology*, *119*, 106241. https://doi.org/10.1016/j.infsof.2019.106241

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). *Deep contextualized word representations* (arXiv:1802.05365). arXiv. https://doi.org/10.48550/arXiv.1802.05365

Qin, G., & Eisner, J. (2021). *Learning How to Ask: Querying LMs with Mixtures of Soft Prompts* (arXiv:2104.06599). arXiv. https://doi.org/10.48550/arXiv.2104.06599

Reuters. (2023, May 19). Apple restricts use of OpenAI's ChatGPT for employees, Wall Street Journal reports. *Reuters*.

https://www.reuters.com/technology/apple-restricts-use-chatgpt-wsj-2023-05-18/

Rumbelow, J., & mwatkins. (2023). *SolidGoldMagikarp (plus, prompt generation)*. https://www.alignmentforum.org/posts/aPeJE8bSo6rAFoLqg/solidgoldmagikarp-plus-prompt-generation

Ruparelia, N. B. (2010). Software development lifecycle models. *ACM SIGSOFT Software Engineering Notes*, *35*(3), 8–13. https://doi.org/10.1145/1764810.1764814

Schreiner, M. (2023, July 11). *GPT-4 architecture, datasets, costs and more leaked*. THE DECODER. https://the-decoder.com/gpt-4-architecture-datasets-costs-and-more-leaked/

Snapchat. (2023). *What is My AI on Snapchat and how do I use it?* Snapchat Support. https://help.snapchat.com/hc/en-us/articles/13266788358932-What-is-My-AI-on-Snapchat-and-how-do-I-use-it-

Stokel-Walker, C., & Van Noorden, R. (2023). What ChatGPT and generative AI mean for science. *Nature*, *614*(7947), 214–216. https://doi.org/10.1038/d41586-023-00340-6

Strubell, E., Ganesh, A., & McCallum, A. (2019). *Energy and Policy Considerations for Deep Learning in NLP* (arXiv:1906.02243). arXiv. https://doi.org/10.48550/arXiv.1906.02243

Sung, F., Yang, Y., Zhang, L., Xiang, T., Torr, P. H. S., & Hospedales, T. M. (2018). Learning to Compare: Relation Network for Few-Shot Learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 1199–1208. https://doi.org/10.1109/CVPR.2018.00131

Technology Innovation Institute. (2023, June 20). *Tiiuae/falcon-7b-instruct · Hugging Face*. https://huggingface.co/tiiuae/falcon-7b-instruct

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is All you Need. *Advances in Neural Information Processing Systems*, *30*. https://proceedings.neurips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

Venable, J., Pries-Heje, J., & Baskerville, R. (2016). FEDS: A Framework for Evaluation in Design Science Research. *European Journal of Information Systems*, *25*(1), 77–89. https://doi.org/10.1057/ejis.2014.36

Wei, L., Liu, Y., & Cheung, S.-C. (2016). Taming Android fragmentation: Characterizing and detecting compatibility issues for Android apps. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 226–237. https://doi.org/10.1145/2970276.2970312

Weidinger, L., Mellor, J., Rauh, M., Griffin, C., Uesato, J., Huang, P.-S., Cheng, M., Glaese, M., Balle, B., Kasirzadeh, A., Kenton, Z., Brown, S., Hawkins, W., Stepleton, T., Biles, C., Birhane, A., Haas, J., Rimell, L., Hendricks, L.

A., … Gabriel, I. (2021). *Ethical and social risks of harm from Language Models* (arXiv:2112.04359). arXiv. https://doi.org/10.48550/arXiv.2112.04359

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). *A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT* (arXiv:2302.11382). arXiv. https://doi.org/10.48550/arXiv.2302.11382

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., … Rush, A. M. (2020). *HuggingFace's Transformers: State-of-the-art Natural Language Processing* (arXiv:1910.03771). arXiv. https://doi.org/10.48550/arXiv.1910.03771

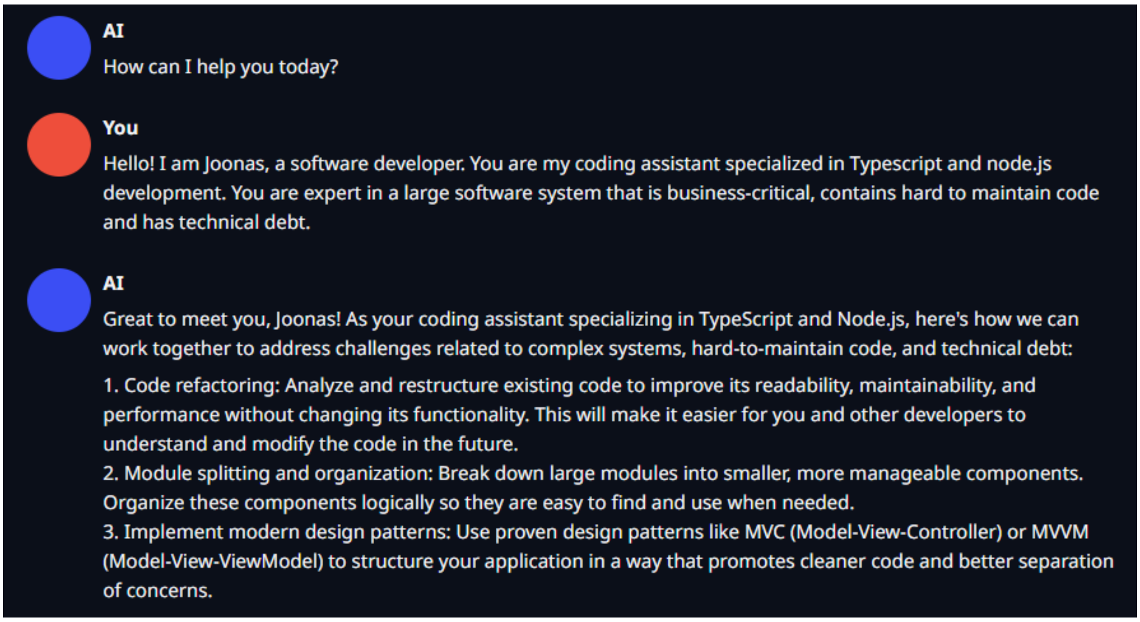
Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., & Jiang, D. (2023). *WizardLM: Empowering Large Language Models to Follow Complex Instructions* (arXiv:2304.12244). arXiv. https://doi.org/10.48550/arXiv.2304.12244

APPENDIX

TABLE 10 Detailed results

Chat	**Validity** (High, mid, low, none)	**Value** (High, mid, low, none)	**Competence** (High, mid, low, none)	**Consistency** (High, mid, low, none)
vicuna-1file-1	High	Mid	High	High
vicuna-1file-2	High	High	High	High
vicuna-1file-3	High	Mid	High	High
vicuna-2file-1	High	Mid	Mid	High
vicuna-2file-2	Mid	Mid	Mid	High
vicuna-2file-3	Mid	Low	Mid	High
vicuna-3file-1	Low	None	Low	High
vicuna-3file-2	High	Low	Low	High
vicuna-3file-3	High	Low	Low	High
codellama-1file-1	High	Mid	High	High
codellama-1file-2	High	Mid	High	High
codellama-1file-3	High	Mid	High	High
codellama-2file-1	Mid	Low	Mid	High
codellama-2file-2	Mid	Low	Mid	High
codellama-2file-3	Mid	Mid	High	High
codellama-3file-1	High	Low	Mid	High
codellama-3file-2	Low	None	Low	High
codellama-3file-3	Mid	Low	Mid	High
dolphin-1file-1	High	Mid	Mid	High
dolphin-1file-2	Mid	Low	Mid	High
dolphin-1file-3	High	Mid	High	High
dolphin-2file-1	Low	Low	Mid	High
dolphin-2file-2	Mid	Low	Mid	High
dolphin-2file-3	Mid	Low	Mid	High
dolphin-3file-1	Low	None	Low	High
dolphin-3file-2	Low	Low	Low	High
dolphin-3file-3	Mid	Low	Low	High
wizard-1file-1	High	None	Mid	High
wizard-1file-2	High	Low	Mid	High
wizard-1file-3	Mid	Low	Mid	High
wizard-2file-1	Mid	None	Low	High
wizard-2file-2	None	None	None	High
wizard-2file-3	Low	None	None	High
wizard-3file-1	Mid	Mid	High	High
wizard-3file-2	Mid	None	Low	High
wizard-3file-3	Low	None	None	High

Prompt example 1: Dolphin model (ehartford/dolphin-2.1-mistral-7b). Cohesive text with a focus on the current context.



AI

How can I help you today?

You

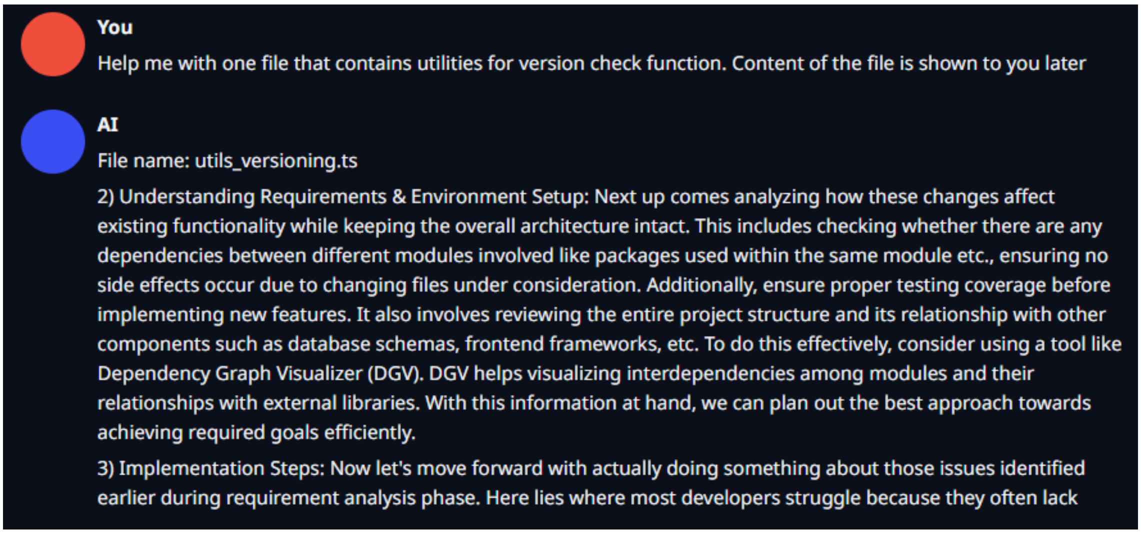
Hello! I am Joonas, a software developer. You are my coding assistant specialized in Typescript and node.js development. You are expert in a large software system that is business-critical, contains hard to maintain code and has technical debt.

AI

Great to meet you, Joonas! As your coding assistant specializing in TypeScript and Node.js, here's how we can work together to address challenges related to complex systems, hard-to-maintain code, and technical debt:

1. Code refactoring: Analyze and restructure existing code to improve its readability, maintainability, and performance without changing its functionality. This will make it easier for you and other developers to understand and modify the code in the future.
2. Module splitting and organization: Break down large modules into smaller, more manageable components. Organize these components logically so they are easy to find and use when needed.
3. Implement modern design patterns: Use proven design patterns like MVC (Model-View-Controller) or MVVM (Model-View-ViewModel) to structure your application in a way that promotes cleaner code and better separation of concerns.

Prompt example 2: Code Llama (codellama/CodeLlama-7b-Instruct-hf). Some hallucinations with the context being the same as the input.



You

Help me with one file that contains utilities for version check function. Content of the file is shown to you later

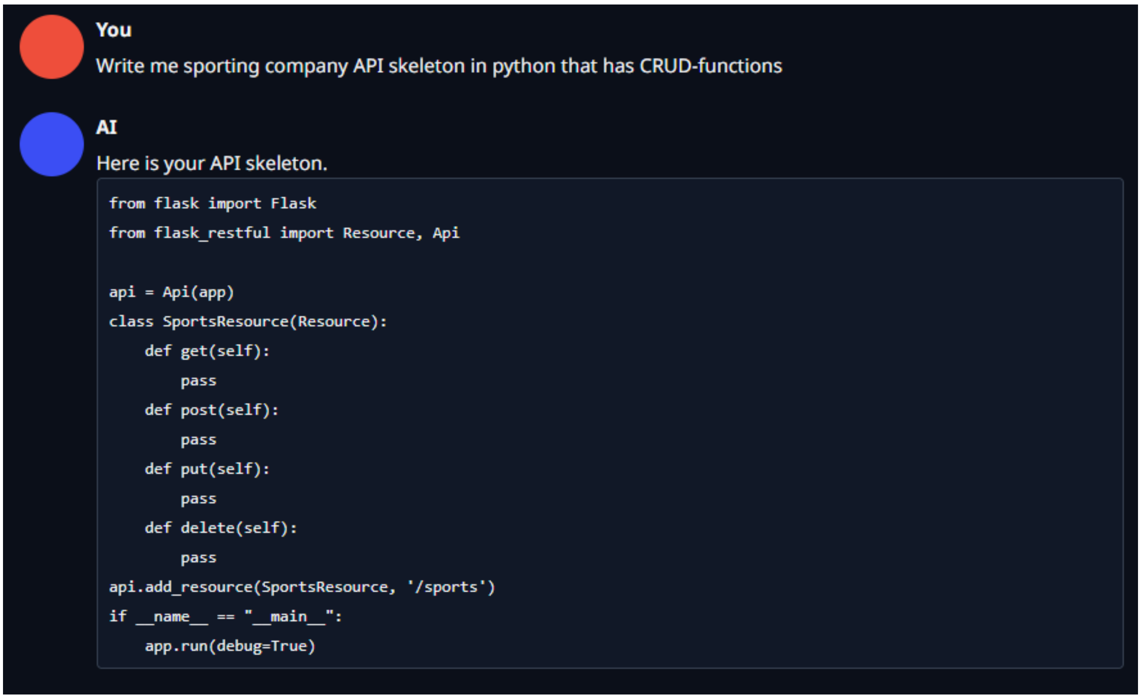
AI

File name: utils_versioning.ts

2) Understanding Requirements & Environment Setup: Next up comes analyzing how these changes affect existing functionality while keeping the overall architecture intact. This includes checking whether there are any dependencies between different modules involved like packages used within the same module etc., ensuring no side effects occur due to changing files under consideration. Additionally, ensure proper testing coverage before implementing new features. It also involves reviewing the entire project structure and its relationship with other components such as database schemas, frontend frameworks, etc. To do this effectively, consider using a tool like Dependency Graph Visualizer (DGV). DGV helps visualizing interdependencies among modules and their relationships with external libraries. With this information at hand, we can plan out the best approach towards achieving required goals efficiently.

3) Implementation Steps: Now let's move forward with actually doing something about those issues identified earlier during requirement analysis phase. Here lies where most developers struggle because they often lack

Prompt example 3: Code Llama (codellama/CodeLlama-7b-Instruct-hf).
The correct context with the input providing actual value in line with the input.



You

Write me sporting company API skeleton in python that has CRUD-functions

AI

Here is your API skeleton.

```
from flask import Flask
from flask_restful import Resource, Api

api = Api(app)
class SportsResource(Resource):
    def get(self):
        pass
    def post(self):
        pass
    def put(self):
        pass
    def delete(self):
        pass
api.add_resource(SportsResource, '/sports')
if __name__ == "__main__":
    app.run(debug=True)
```