# Generating Specifications from Requirements Documents for Smart Devices Using Large Language Models (LLMs)

Rainer Lutze[1(✉)] and Klemens Waldhör[2]

[1] Dr.-Ing. Rainer Lutze Consulting, Wachtlerhof, Langenzenn, Germany
rainerlutze@lustcon.eu
[2] FOM University of Applied Sciences, Nuremberg, Germany
klemens.waldhoer@fom.de

**Abstract.** The current contribution of artificial intelligence based Large Language Models (LLMs) in supporting the requirements engineering and design phase of software centered systems is analyzed. The application domain is focused on smart devices and services for Ambient Assisted Living (AAL), e.g. programmable smartwatches. In the realm of AAL, programmable smartwatches offer significant potential to support elderly users in their daily activities. However, developing applications for these devices is resource demanding, algorithmically difficult and requires careful consideration of various factors, including user-specific needs (e.g. a diminished natural sensation of thirst) and technical constraints (restricted computational power due to limited battery capacity). Our approach first utilizes widespread LLMs like ChatGPT, BARD to automatically interpret and enrich product concept catalogues, targeting at comprehensive, consistent and tailored requirements specifications for the specific domain of application. Secondly, we analyze in which extent LLMs today can contribute to the original design phase by introducing suitable functional principles and reference architectures for fulfilling the services specified as requirements including the constraints on its operations. To demonstrate the challenges and perils of this approach, we will detail the process using the specific use case of automatic drinking detection and providing suitable advice for preventing dehydration for elderly users. We will discuss and differentiate the principal strength of the approach from its actual limitations by the presently available LLMs, which are expected to increase and elaborate their generative capabilities rather frequently.

**Keywords:** Large language models (LLMs) · LLMs for analyzing and enriching requirements definitions · LLMs for generating design specifications · smart devices and service · systems and software engineering

## 1 Background and Motivation

Generative artificial intelligence (AI) in the form of large language models (LLMs) is increasingly popular also for software engineering [1, 2]. For the software development life cycle (SDLC), major efficiency improvements will be expected from this technology. Currently, the majority of all LLM studies analyzes and has been implemented

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
M. Kurosu and A. Hashizume (Eds.): HCII 2024, LNCS 14684, pp. 94–108, 2024.
https://doi.org/10.1007/978-3-031-60405-8_7

for the *software development (coding, test) phase, maintenance, quality assurance and requirements engineering phase* (listed in decreasing frequency, e.g. Fig. 8, [1]). A typical example for this application scenario is the IBM Programmer's Assistant *Socrates* [3] based on OpenAI's Codex LLM, which also powers the GitHub Copilot LLM, for supporting Python *coding phases.* For the *software design phase* by usage of LLMs, [1] reports only a minimum of 1,29% of all analyzed papers. But, especially for long-lived systems and software, especially the design phase is essential for the documenting taken design decisions, in order the record the original scope of the system and to facilitate further evolution and refactoring of the software during its life cycle.

## 2 Prior Work

Sommerville [4] differentiates systems engineering ([4], chap. 19) from software engineering. Systems engineering targets at a complete sociotechnical system consisting of hardware computing devices, software delivering a set of services and the system owner, its users and their organizational and societal context. The engineering process has to establish a purposeful collection of interrelated components of different kind collaborating to fulfill defined requirements definitions. ([4], 19.1). This extended reach requires to consider for the properties of the system, e.g. reliability, not only the software, but also hardware as well as the envisioned operators of the system, users. One decisive feature of systems engineering is the central role of conceptual design within the engineering process. Conceptual design incrementally elaborates a vision of the required system and its features, simultaneously adapting the requirements definitions, deciding what needs to be procured for the realization of the system, what needs to be developed and also shaping the operational concept for the system. This systems engineering approach will be also the case in the chosen application domains of smart devices and services.

In contrast, for pure software engineering, in principle the system and software design phase typically follows a prior requirements analysis and definition phase ([4], 2.1.1). In practice and especially for up-to-date iterative software processes like agile development, software requirements engineering and software design will be intertwined and performed alternatingly. System models as essential design findings will be used as early as possible in order to clarify and illustrate the fulfillment of prominent requirements to stakeholders already during the initial requirements engineering phase.

Systematically, the design phase is typically subdivided in an initial *system modelling* sub-phase followed by an *architectural design* sub-phase and a *detailed design* subphase ([4], chap. 5, 6). For the *system modelling* sub-phase, suitable *structural* and *behavioral* models fulfilling the entirety of requirements definitions will be identified and typically expressed by *UML class diagrams, collaboration*, *sequence* and/or *activity diagrams.* The following *architectural design* introduces a suitable *system structure,* its *distribution* and *decomposition* into components. In this sub-phase, proven domain specific reference architectures, e.g. *transaction processing systems*, which are known to solve similar tasks as stated in the requirements definition, are identified, instantiated, adapted and configured. UML *deployment* or – more fine-grained - *component diagrams* are the notation of choice for representing such system structure. The final detailed design subphase focusses on database design, interface design and component selection and design ([4], chap. 2.2.2).

The **design thinking** methodology puts a multidisciplinary team of experts and users in the center of a problem solving task. The focal point of design thinking is an iterative, rapid prototyping process with a strict orientation on the actual requirements of the envisioned target groups. Based on *empathy* with the envisioned users, typical cycle phases of the design thinking process, for which a plenitude of supporting tool sets are available, are: *problem framing, inspiration, ideation, prototyping, and testing/evaluating.* The typical application area of the design thinking methodology is the innovation of products and services within business and social contexts [5], not so much system and software. Especially, there is no specific concise approach for the *ideation phase* creating a design from elaborated requirements other than the combination of *divergent* and *convergent* thinking. In the systems and software engineering discipline, the methodology was elaborated to **user-centered design** by D. Norman [6]. Many current UI-design concept, like *personas, user stories, or use cases, …* build on this methodology. Use case diagrams have also made their way into the UML. Although, the inherent weakness of the ideation phase is still unsolved.

The **model based systems design** is characterized in ([7], def. 1.44) as: *"To design a system is to develop a model on the basis of which a real system can be built, developed, or deployed that will satisfy all its requirements."* The original Wymore 1993 model based systems engineering concept is strongly mathematically oriented and resulted in the Systems Modelling Language (SysML). From our point of view it is not as universally suited and widespread used than the Universal Modelling Language (UML) [8] for representing software designs.

The **domain-driven design** [9] methodology builds on a suitable, known systems resp. Software reference model catalogues for the specific application domain and its logics. The methodology selects and configures a specific instance from the catalogue as foundation for the software design, which fits best to the entirety of requirement specifications for the specific task. Originally emerging from *object-oriented design patters,* meanwhile more complex models like *digital twins* [10] or *state machines* constitute the core elements of the reference model catalogue.

**Concept design** [11] builds on an inventory of domain independent, reuseable concepts, which are characterized by an (1) expressive *purpose* addressing a specific user need, (2) *familiarity* to the users, and (3) *independence* from other concepts, so that they can be understood, designed and analyzed on at a time separately from other concepts. A concept is characterized by its dynamic behavior (functional principle") typically involving multiple objects and encapsulating its own data model. But, in essence, concepts are *teleological* entities rather than only *semantic* entities [11].

For the usage of LLMs for software engineering, White et al. [12] present a customized prompt pattern catalogue (for ChatGPT) in order to improve and focus the LLM output to the specific needs of different software engineering phases. (see [12], for example section IV.G for the design supporting *"architectural possibilities pattern").* Ahmad et al. [13] use a ChatGPT based bot for generating *recommendations for software architecture.* All these standard LLM based approaches are in prototype state today and suffer from the inherent risks resulting from the opaque data acquisition process: e.g. potential copyright infringements, data poisoning [14], privacy regulations violations by the LLM recommendations, …. On the other side, the built-up of LLMs based

exclusively on reliable sources is incredibly expensive, especially for SMEs, so actually there is no practical alternative than using widespread LLMs like OpenAI's ChatGPT™ and(or) Google's BARD™.

## 3 Goals, Methodology and Research Questions

The presented research aims to verify the effectivity of LLMs for generating systems and software design specifications, based on an analysis end enrichment of given domain specific brief product concept catalogues. Special importance will be put on considering the proven engineering processes for systems and software development for this design phase. Ideally, the design specifications will be expressed in diagrams of the worldwide standard Universal Modelling Language (UML), serving as a universal visual guide for developers. These diagrams are expected to be clear, well-structured, annotated and immediately understandable, facilitating the further development process. It can additionally record taken design decisions and argues about the presence of features within the generated UML diagrams.

### 3.1 Research Questions

The following four specific research questions have been analyzed by our work:

Q1 To which extend is a domains specific understanding, analysis and enrichment of product concept catalogues currently possible by widespread LLMs, targeting at a consistent, comprehensive and complete requirements specification, customized to the domain of application?
Q2 What is the significance of additional information (either by specific prompt control, documents) for automatically enriching the product concept catalogues given to the LLMs?
Q3 What is the significance of additional information (either by specific prompt control, documents) for supporting the original design process to be performed by the LLMs?
Q4 To which extend is generation of diagrammatic UML design specifications possible by the current capabilities of widespread LLMs?

### 3.2 Methodology Applied

We performed an experiment consisting of two main phases. For the first main phase for analyzing and enriching given brief product concept catalogues, we used a group of undergraduate computer science students, in order to minimize the potential risk of getting biased assessments about the generative power of current mainstream LLMs (ChaptGPT4.0, BARD). The students had no prior domain expertise in AAL and programmable smartwatch app design experience for the chosen field of application. With this setup of the experiment, we deliberately wanted to avoid an incidental domain specific know-how transfer from the students into the generated specifications other than by direct injection through the LLMs itself. Different student groups were advised to actively use the generative AIs ChatGPT4 or BARD, or – as a control group – only use classical Internet search, without the help of Generative AIs. As an additional reference

in assessing the student work results, we used our proven design and implementation work in the field of application [15–17].

The second main phase of the experiment, aiming to verify the conceptual design capabilities of the LLMs based on the results of the first main phase, was carried out by the authors of this paper itself, due to the rather unambiguous (negative) results which do not allow ambiguous assessments.

In the following paragraphs *italic text* indicates answers provided by ChatGPT or BARD.

## 4 Experiment

### 4.1 First Main Phase: Generating Requirements Specifications

**Experimental Background**
The primary aim of this first main phase was to ascertain whether specifications generated by Large Language Models (LLMs) yield results comparable to those produced by human efforts. LLMs used are: ChatGPT 4 (https://chat.openai.com) and BARD (https://bard.google.com). Owing to time constraints, the focus of the experiment was narrowed to the initial stages of the development cycle, specifically the requirements engineering phase. The documents produced for this part of the experiment can be found in [19].

**Experimental Design.** In an initial stage, a preliminary test with the given assignment (detailed below) of product concept specifications was conducted. The primary objective of this pre-test was to assess the clarity and comprehensibility of the assignment instructions. Participants were divided into three distinct groups: the first group was tasked with manually writing the requirements specification without any external aid, while the second and third group utilized ChatGPT and BARD for the same task.

As this was a preliminary test, the results were not subjected to in-depth analysis. However, it was observed that the solutions generated by the ChatGPT and BARD group were significantly more sophisticated than those of the manual group. The latter's submission predominantly comprised a basic table of items, lacking in structured organization. It is important to note that, unlike participants in the main experiment, these students were in their third semester of a Business Informatics program and had limited experience in software development. Specifically, they had not yet taken a course in software engineering. The pre-test was conducted as part of a lecture on Big Data and Data Science.

The main experiment of this phase consisted of three stages:

In the initial first stage of the experiment, students were allocated into three distinct groups: a manual group (M), a ChatGPT 4.0 group (C), and a BARD group (B). Each group received an identical assignment, presented in German. The manual group's task was to draft the specification document using their software development skills and web resources, explicitly excluding the use of any AI software. In contrast, the remaining two groups employed ChatGPT and BARD, respectively, for their tasks. These groups were required to meticulously record the prompts they used and the corresponding outputs generated by the AI systems, with a strict rule against altering the text.

The main experiment involved six male students, with each group comprising two students. These students were in their fifth semester of a Bachelor's program in Business Informatics, possessing adequate knowledge and practical experience in software engineering and development. The theme of the course was Big Data and Data Science. Notably, they were currently enrolled in a software engineering course. The experiment was conducted as part of a lecture on web technology. All participating students were full-time employees at a prominent German telecommunication company, working in various software-related departments. They were allotted 60 min to complete the task.

Following the exercise, a focused discussion was held. During this session, students were specifically prompted to contemplate any overlooked or implicit requirements of the use case and to propose enhancements to their documents. The exercise concluded with a comprehensive discussion, where the students critically evaluated the specification documents and assigned ratings to each solution based on their performance and accuracy.

In a subsequent follow-up evaluation, the documents produced were further assessed by a group of business informatics students in the fifth semester. These students, who were enrolled in a software engineering course, possessed some experience with requirement analysis, gained either through professional experience or the course itself. The evaluation teams comprised two or three students each. Among these, two groups were tasked with assessing the manually created version of the documents, while one group evaluated the ChatGPT-generated version, and another assessed the version produced by BARD. Ratings should be assigned using a school grading scale, where 1 is the highest (best) grade and 5 is the lowest (worst) grade. Initially, each group familiarized themselves with the assignment task, followed by a thorough reading of the documents assigned to them. This process was allocated a total time frame of approximately 30 min. Upon completion of their evaluations, each group presented their findings, detailing their ratings along with the rationale behind their assessments. This structure ensured a comprehensive and reasoned analysis of the documents.

**Assignment Text.** The text was composed in German to mitigate any potential misunderstandings arising from language barriers. Deliberately, it was crafted to be somewhat vague, mirroring the common initial phase of a project. This approach was based on the presumption that the customer might not have a clear understanding of the specifics of their envisioned idea. Concurrently, it was hypothesized that the development team lacked substantial expertise in Ambient Assisted Living (AAL) or smartwatch software development.

*You have received the following product concept catalogue from your client:*

*The client needs a smartwatch app that can monitor whether and how much their care-dependent individuals are drinking. If the person drinks too little, they should receive a reminder on their smartwatch. If this reminder is ignored, the care staff must be informed. These reminders should be designed in such a way that care-dependent individuals can easily recognize and respond to them. Also, all data should be centrally managed. In the future, it is planned to monitor additional activities such as walking, sleeping, etc. Additionally, algorithmic sketches should be presented on how the drinking recognition can be implemented*

*This is all the client has told you. Your task is now to analyze the scenario, determine requirements, use cases, identify missing aspects, and create a specification and requirement document*

*Since your CEO has heard that LLMs are the future, and the company has not yet been recognized in this field of care, he wants you to work with LLMs for the first draft. However, since he is not sure if all LLMs work equally well, two teams should use different LLMs. And as a precaution, he commissions another team to analyze the scenario in a traditional way*

**Limitations of the Experiment.** It is essential to acknowledge a fundamental limitation of the experiment: the participants were students rather than full-time developers with extensive experience in specifying and designing software. Additionally, these participants lacked experience in designing software for smartwatches and Ambient Assisted Living (AAL) systems. It's worth considering that Large Language Models (LLMs) could potentially assist non-experts in creating specifications, which could actually highlight the utility of LLMs in bridging expertise gaps in software development.

A noteworthy limitation, particularly concerning ChatGPT and BARD, is the usage of different prompts by various groups. For a more detailed and accurate comparison of the outcomes from both Large Language Models (LLMs), it would be essential to utilize an identical sequence of prompts. However, this level of uniformity in prompt usage was not within the scope of this experiment.

**Discussion of the Experiment**

*Discussion of the Manual RE Analysis.* The requirements specification document produced by this group, inspired by IEEE830 and its successor ISO/IEC/IEEE 29148:2011 as taught in the SE course, comprises four chapters: an introduction, a general description, individual requirements, and technical restrictions, spanning one and a half pages. The six-line introduction outlines the project's goal and application area. The general description covers functionality, user management, restrictions, constraints, and dependencies. The core section lists eleven requirements, expanding on the assignment text with more detail. A typical example is the following:

*Function 3 - Visual and Acoustic Signal: The app notifies the person in care through an audio signal and a colored pop-up on the display, so they know that the app has successfully recorded the entry*

*Function 8 - Adjustment of Intervals According to Weather Conditions: depending on the current weather, the app adjusts the time intervals between reminders, as one needs to be notified more frequently in higher temperatures (requires access to the current location of the person and a weather API)*

For instance, Function 8, not originally in the customer's product concept catalogue, fits well within the project context. The final chapter lists non-functional requirements, like general technical limitations. However, the document resembles a brainstorming output, lacking specific aspects crucial for smartwatch software development, such as

battery management. This reflects a typical outcome from developers new to a domain, lacking deep insight into implementation challenges. It's a useful starting point, but requires further refinement in subsequent sessions.

*Discussion of the ChatGPT RE Analysis.* Overall about four pages requirements specification document was produced with four prompts. The first prompt asked for an analysis of the scenario. Here ChatGPT started with the main function, esp. drinking detection. Basically ChatGPT takes the description as a starting point and enhances it with features. functional requirements were derived as follows:

*Functional Requirements*

*• Hydration Monitoring: Recording the daily fluid intake of the user*

*• Alert System: Notifications when the recommended drinking volume is not met*

*• Notification of Care Personnel: Automatic information to the care staff if warnings are ignored*

*• User Interaction: Simple and intuitive interface for people in need of care*

When asked for a requirements and functional specification (roughly corresponding to the German terms "Lasten- und Pflichtenheft" which were used in the prompt) - ChatGPT came up with five main points starting from goals, functional, non-functional, technical requirements and a time plan (not further detailed). The Functional Specification gave an overview of phases as part of the project management. Basically the six steps mentioned followed some kind of waterfall model. These next prompts dealt with identifying use cases. The final prompt deals with missing aspects of the analysis. ChatGPT comes up with ten items starting from integration into the health care system to user acceptance. It explicitly mentions battery management as a missing item. A typical example is the following:

*• Integration with Existing Health Systems: It has not been specified how the app should interact with existing health or care management systems. Compatibility and integration with such systems could be crucial for efficiency*

*• Hardware Requirements: The technical requirements for the smartwatch (e.g., battery life, sensors) have not been precisely defined. This is critical to ensure that the hardware can support the software requirements*

The analysis of ChatGPT, as evidenced by the four prompts, is notably more advanced, informative, and well-structured in its overall presentation.

However, it falls short in delving into detailed aspects, particularly in addressing low battery management. The analysis lacks specific suggestions or considerations on how to effectively tackle this issue. In a separate session ChatGPT was asked how to deal with low battery situations. Although it came up with several useful suggestion it did not propose just to use a second smartwatch in that case. If explicitly asked if this would help it came up with a set of useful guidelines how to integrate a second smartwatch.

*Discussion of the BARD RE Analysis.* BARD produced a requirements specification document similar to that of ChatGPT. However, for BARD's document, there was a specific request to adhere to the DIN 69901-5 standard.

Requirements looked like that:

*The requirements can be specified as follows:*

*Hydration Monitoring: The app must record and store the fluid intake of the care-dependent individuals. Various sensors can be used for this purpose, such as an accelerometer to detect the opening and closing of a bottle, or a moisture sensor to measure the amount of water consumed*

*Hydration Alerts: If the fluid intake falls below a certain level, the app must generate a notification for the care-dependent individual. This alert must be designed to be easily understandable and followable*

Similarly, use cases were developed. In response to inquiries about overlooked elements, BARD provided the following suggestions:

*The following aspects have not yet been considered:*

*Security: The app must be secure and the data of the care-dependent individuals must be protected*

*Data Privacy: The app must ensure the privacy of the care-dependent individuals' data*

*Usability: The app must be user-friendly and take into account the needs of the care-dependent individuals*

Interestingly, BARD did not address battery management in its suggestions.

Asked to generate Requirement and Functional Specifications, it produced a well-structured document that serves as a solid foundation for further refinement.

Overall, the documents generated by BARD were quite similar to those produced by ChatGPT. However, ChatGPT appeared slightly more advanced in highlighting technical limitations, which could also be attributed to the use of varying prompts by the respective groups.

**Evaluation of the Requirements Within the Course.** Table 1 provides a comparative overview of ratings for three requirement documents. These documents were created using for-mentioned different methods: manual preparation, ChatGPT 4.0, and BARD. The evaluation process for the manually prepared document involved group discussions among the students, followed by the assignment of a final overall rating.

The students showed a preference for the solutions generated by the Large Language Models (LLMs), particularly favoring BARD. A notable factor influencing this preference could be BARD's tendency to produce more extensive documents, esp. by providing a template for further improvements. To further assess ChatGPT's evaluative skills, it was tasked with analyzing the three requirement documents. Intriguingly, ChatGPT's assessment aligned closely with the students' rankings.

**Table 1.** Evaluation of experiment 1 (Big Data Course)

| | Manual | ChatGPT | BARD |
|---|---|---|---|
| Criteria | | | |
| Extent | 0 | + | ++ |
| Structure | + | + | + |
| Number Requirements | + | ++ | ++ |
| New Requirements | + | ++ | + |
| Student Evaluations | | | |
| Criteria Evaluation | 3 | 6 | 6 |
| Overall Ranking | III | II | I |
| ChatGPT Evaluations | | | |
| Evaluation | ++ | ++ | ++ |
| School Grade | 3 | 2 | 1 |

0 = neutral, + = good, ++ = very good; 1 = very good = 5 very bad

*Follow-up evaluation of the requirements within the software engineering course.* The outcomes of the follow-up evaluation are presented in the Table 2 below

**Table 2.** Follow-up evaluation of experimenT 1 (SE course)

| RE Source | Structure | Content | Details | Size | Overall |
|---|---|---|---|---|---|
| Manual I | 2 | 1 | 3 | 3 | 2,75 |
| Manual II | 3 | 1 | 4 | 2 | 3 |
| ChatGPT | 2 | 4 | 3 | 4 | 3 |
| BARD | 4 | 2 | 2 | 3 | 3 |

The evaluation revealed that all requirements specifications received broadly similar ratings. A common critique of the LLM-generated content was its lengthiness and perceived verbosity, though some students appreciated this detail as providing a solid foundation for further development. Notably, it was observed that technical implementation details were absent in the specifications, but it's important to note that this was not the primary focus of these documents.

Overall, the students didn't show a distinct preference for any particular source of the specifications. Interestingly, they assumed all the documents were generated by LLMs. The groups who evaluated the manually created texts were surprised to learn that their assigned documents were actually human-produced. This suggests that the LLMs performed impressively, blurring the lines between human and machine-generated content in terms of quality and style.

### 4.2 Experiment Second Main Phase: Generating Design Documents

The central goal of the experiment in this main phase was to support the conceptual design of an algorithm capable of monitoring the fluid ingestion behavior of elderly people. To assess the depth of a proposal that could be produced by a LLM, we employed ChatGPT to create several prompts. For an extensive description, refer to [20]. The initial prompt requested an approach for accomplishing this task:

> *"The task is to develop a smartwatch app which is able to detect drinking. The smartwatch comprises various sensors. Please advise which approach could be used."*

ChatGPT initially outlined potential sensors, then proposed methods for their utilization.

> *"5. Machine Learning and Data Processing:*

- *Use machine learning techniques to process and analyze the sensor data.*
- *Collect a large dataset of sensor readings during drinking and other activities for training the ML models.*
- *Implement real-time data processing to allow the app to detect drinking as it happens."*

The response was fundamentally accurate, particularly in its recommendation of ML-based methods. It also addressed several other aspects, including privacy concerns. Following that, we explored the feasibility of using neural networks for this purpose. ChatGPT delineated the necessary steps for such a solution. Notably, its insights on network architecture were intriguing.

> "3. Neural Network Design:

- *Model Architecture: For time-series data, recurrent neural networks (RNNs) like LSTM (Long Short-Term Memory) or GRUs (Gated Recurrent Units) are effective as they can capture temporal dependencies. Convolutional Neural Networks (CNNs) can also be used to identify patterns in sensor data."*

ChatGPT recognized the issue as a time-series problem and suggested a range of appropriate architectural solutions. Given our extensive experience in developing applications for Ambient Assisted Living (AAL), we could confidently assess the relevance of these suggestions. All proposed architectures were practical and had been previously implemented. When asked which architecture was most promising, ChatGPT presented six options and provided selection criteria, ultimately recommending:

> *"A hybrid CNN-LSTM model is often a strong candidate for this kind of task. It combines the feature extraction capabilities of CNNs with the sequence modeling strengths of LSTMs, making it well-suited for analyzing complex sensor data patterns associated with human movements like drinking."*

This recommendation seemed highly promising. Although we have previously utilized LSTMs, the integration with CNNs had not been explored. Consequently, we requested a Python implementation from ChatGPT, resulting in a straightforward model combining CNN and LSTM layers. Although it lacked detailed explanations for certain parameters, such as "model.add(LSTM(50, return_sequences = True))", it provided a basic framework for training.

Further prompts inquired about alternative algorithmic approaches for analyzing sensor data linked to drinking. We had previously employed simpler models like logistic regression. ChatGPT suggested several viable methods.

Although, and mentioned as an example, as well ChaptGPT and BARD failed to offer specific designs for fulfilling the central requirements of *hydration monitoring* and *hydration alerts* (see above, chapt. 4.1). The critical issue in this subject is to determine the individual nominal value, how much fluid an elderly person should ingest daily. In general, a rough estimation of 30 ml per kg bodyweight per day can be used [18], or the much more elaborated model [17], chap. 3.1. In each case, a couple of additional corrective factors must be considered for calculating the correct nominal value: actual exposure of the elderly person to higher temperatures, excessive physical activities, actual medical conditions like fewer, diarrhea or vomiting, heart or renal insufficiencies. From this plenitude of factors, only the possibility to correct the nominal value by excessive physical activity detected by the smartwatch was proactively offered by ChatGPT or BARD. For a professional solution, this is not sufficient with respect to the state of the art.

Lastly, we asked about implementing this on a smartwatch, leading to the suggestion of using TensorFlow Lite, suitable for mobile phones and smartwatches.

In the final refinement step, we asked ChatGPT developing an Android implementation using the TensorFlow Lite model, which would generate an alert if the wearer did not drink within an hour. ChatGPT delivered a basic, yet functional, code. While not ready for production, the experiment's one-hour timeframe made the results notably impressive, considering previous implementations took weeks.

This experiment demonstrated that ChatGPT can significantly expedite the development of a functional prototype app, potentially within a day. While an experienced programmer with relevant domain knowledge might achieve similar timelines, it's remarkable that ChatGPT enables those without extensive expertise in AAL or smartwatch programming to develop solutions so swiftly. In conclusion, for certain stages of the software development cycle, LLMs like ChatGPT can be immensely beneficial in accelerating development.

## 5 Conclusions

In the following we will discuss our findings with regard to our research questions Q1–Q4.

Q1: The use of LLMs in generating elaborated requirements specifications from given brief product concept catalogues represents a significant qualitative and efficiency-improving advancement in the development of smartwatch applications for AAL.

Q2: Currently, LLMs can only generate recommendations and lists of potentially useful reference architectures and possible functional principles for the wanted solution. But, the core competence of conceptual design, to **select,** propose and justify the *best suited solution* to fulfill the entirety of specified requirements is still missing. The effectiveness of the proposed solutions increases with the specificity of the prompt, especially regarding architectural or algorithmic challenges. This was evident in our experiment, where the choice of an appropriate neural network architecture yielded better outcomes when detailed prompts were used.
Q3: The response here mirrors that of Q2. The efficacy of providing the user with additional information largely hinges on the prompts used. Generally, the supplementary information provided tends to be of a relatively high level. However, it frequently lacks in addressing specific, detailed aspects.
Q4: Consequently, it is still not possible for LLMs to generate a graphical UML depiction of this favored solution. Therefore, a domain experienced human systems architect is still essential for this systems design phase. The assistance ChatGPT and BARD can actually offer is of no essential help for this specific task, because the LLM generated lists of potential technology candidates is expected to be already and always on the mind of a real human domain expert. It's important to note that in later stages of the development process, such as generating class diagrams from more detailed implementation specifications, LLMs prove to be quite helpful. Moreover, as the process progresses towards producing source code, the support and effectiveness of LLMs become even more pronounced.'

One of the authors, Waldhör, is actively involved in a research project that integrates smart home technology, smartwatches, and robotics within the framework of Ambient Assisted Living (AAL) at home. In this context, the application of Large Language Models (LLMs) in the final phase of implementation – particularly in generating source code from REST interface specifications and integrating these interfaces with sensor data transmission – has demonstrated significant utility. This effectiveness is notably evident when compared to traditional methods that rely on resources such as GitHub.

Unfortunately, the original target of generating similar professional *design documents,* especially UML diagrams for representing the design results, is still out of scope by the current enhancements of LLM technology. The situation is set to evolve once Large Language Models (LLMs) gain the capability to a) generate XML-based descriptions for UML, such as XMI (XML Metadata Interchange), and b) integrate with graphical software systems. This advancement will significantly enhance the utility and application of LLMs in the field of UML modeling.

Finally, it is crucial to acknowledge that these findings 1) are specific to the selected application domain and 2) apply only to the Large Language Models (LLMs) as they existed at the time of writing this paper. If applied to broader application domains, such as ERP systems, the results might vary, with LLMs potentially offering more refined specifications. While generalizing these conclusions to other application domains and projecting them into the near future is not entirely implausible, each potential use case must be carefully evaluated on its own merits.

## References


1. Hou, X, Zhao, Y., Liu, Y., et al.: Large language models for software engineering: a systematic literature review, research report, Huazhong University of Science and Technology, Wuhan, China, arXiv:2308.10620v4, p. 62, September 2023
2. Zhang, Q., Fang, C., Xie, Y., et al: A survey on large language models for software engineering, China, arXiv:2312.15223v1, p. 57, December 2023
3. Ross, S.I., Martinez, F., Houde, S., Muller, M., Weisz, J.D.: The programmer's assistant: conversational interaction with a large language model for software development. In: ACM 28th International Conference on Intelligent User Interfaces (IUI '23), 27–31 March, Sydney, Australia (2023). https://doi.org/10.1145/3581641.3584037
4. Sommerville, I.: Software Engineering, 10th updated edn. Pearson Education Limited, Harlow, England (2017)
5. Brown, T.: Design thinking. Harv. Bus. Rev. **86**(6), 84–92(2008)
6. Norman, A., Draper, W.: User Centered System Design – New Perspectives on Human-Computer Interaction, CRC Press, Boca Raton (1986)
7. Wymore, A.W.: Model Based Systems Engineering: An Introduction to the Mathematical Theory of Discrete Systems and to the Tricotyledon of System Design, CRC Press, Boca Raton (1993)
8. Bennett, S., Skelton, J., Lunn, K.: UML, 2nd edn. Schaum's Outlines, McGrawHill Education – Europe, New York (1991)
9. Evans, E.: Domain Driven Design – Tackling Complexity in the Heart of Software, Addison-Wesley, MA, USA, 8th printing (2006)
10. Lutze, R.: Digital twins – a determining engineering pattern. In: IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC), Edinburgh, United Kingdom, pp. 1–9 (2023). https://doi.org/10.1109/ICE/ITMC58018.2023.10332426
11. Wilczynski, P., Gregiore-Wright, T., Jackson, D.: Concept centric software development – an experience report. In: ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward! '23), 25–27 October, Cascais, Portugal (2023). https://doi.org/10.1145/3622758.3622894
12. White, J., Hays, S., Fu, Q., Spencer-Smith, J., Schmidt, D.C.: CHatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design, Vanderbilt University, Nashville, TN, USA, arXiv:2303.07839v1, p. 14, March 2023
13. Ahmad, A., Waseem, M., Liang, P., Fadimeh, M., Aktar, M.S., Mikkonen, T.: Towards human-bot collaborative software architecting with ChatGPT. In: ACM International Conference on Evaluating and Assessment in Software Engineering (EASE '23), June 14–16 2023, Oulu, Finland, pp. 279–285 (2023). https://doi.org/10.1145/3593434.3593468
14. Schneider, B, Stuber, M.: IntelligenteWerkzeuge im Software Engineering. Informatik J. (12), 21–292021. https://opus.hs-furtwangen.de/frontdoor/deliver/index/docId/7709/file/IntelligenteWerkzeugeimSoftwareEngineering.pdf
15. Lutze, R., Waldhör, K.: The application architecture of smartwatch apps – analysis, principles of design and organization. In: Mayr, H.C., Pinzger, M. (eds.) INFORMATIK 2016. LNI, vol. P259, pp. 1865–1878. Springer, Bonn (2016). ISBN 978-3-88579-653-4, ISSN 1617-5468, https://cs.emis.de/LNI/Proceedings/Proceedings259/1865.pdf
16. Lutze, R.: Practicality of automatic monitoring sufficient fluid intake for older people. In: IEEE 10th International Conference on Healthcare Informatics (ICHI), 11–14 June, Rochester, MN, USA, pp. 330–336 (2022). https://doi.org/10.1109/ICHI54592.2022.00054
17. Lutze, R., Waldhör, K.: Practicality aspects of automatic fluid intake monitoring via smartwatches. In: Kurosu, M., Hashizume, A. (eds.) H HCII 2023. LNCS, vol. 14014, pp. 67–86. Springer, Cham (2023). https://doi.org/10.1007/978-3-031-35572-1_5

18. Hall, J.E., Guyton, A.C.: Textbook on Medical Physiology, 14th edn. Elsevier Publishing Inc., Philadelphia, PA, USA (2020)
19. Waldhör, K., et al.: Experiment: Vergleich von Manueller vs. LLM basierten Analyse eines AAL Software Scenarios [Experiment: Comparison of Manual vs. LLM-based Analysis of an AAL Software Scenario] RETexte_konsolidiert_v1.pdf (2024)
20. Waldhör, K.: Development of a drinking detection smartwatch app for android: a comprehensive documentation of ChatGPT's role in the process (2024). http://www.waldhor.com/chi24/chatgptdrinkappdocumentation.pdf