# Incorporating AI in the Teaching of Requirements Tracing Within Software Engineering

Juan Ortiz Couder
*Department of Electrical Engineering and Computer Science*
*Embry-Riddle Aeronautical University*
Daytona Beach, Florida, United States of America
ortizcoj@my.erau.edu

William C. Pate
*Department of Electrical Engineering and Computer Science*
*Embry-Riddle Aeronautical University*
Daytona Beach, Florida, United States of America
patew@my.erau.edu

Daniel A. Machado
*Department of Electrical Engineering and Computer Science*
*Embry-Riddle Aeronautical University*
Daytona Beach, Florida, United States of America
machadd4@my.erau.edu

Omar Ochoa
*Department of Electrical Engineering and Computer Science*
*Embry-Riddle Aeronautical University*
Daytona Beach, Florida, United States of America
ochoao@erau.edu

***Abstract*—During the Software Development Lifecycle (SDLC), the first stage entails the Requirement Engineering phase. In this phase, engineers gather, analyze, and specify the requirements for a software system. Requirements play a crucial role in the SDLC as they establish the foundation for the entire system by defining the expected behaviors of the software system to be built. The resulting specifications are captured in a Software Requirement Specification (SRS) document. As part of the validation process, requirement specifications are traced. Requirement tracing involves linking the requirement to the artifacts where the customer requested the high-level requirement. Teaching proper requirements tracing can be challenging in a traditional classroom setting. It is essential to educate future software engineers on the proper process of developing an SRS document and of tracing requirements back to the originating artifact, which is also challenging due to the complexity and large scope of applying the complete requirements engineering process. Understanding how changes in customer needs can impact requirements is an imperative learning opportunity. In this work, we aim to incorporate the use of AI in the teaching of requirements tracing using Large Language Models. In this experiment, both GPT-3.5 and GPT-4 are provided the transcript of an interview between the customer and the engineering team, as well as the subsequent requirements elicited from that meeting and other customer provided artifacts. The GPTs are then instructed to determine which requirements can be traced back to the interview transcript. At the same time, the students (the requirements engineering team) conduct their own effort to trace requirements back to the original interview. The experiment was taken one step further to assess students' and the GPTs abilities to address requirements modifications. After another interview with the customer, where some needs were changed, some requirements were modified, and students, and GPTs were asked to trace the modified requirements to the new interview. The results proved that students are better than both GPT versions at tracing modified requirements, yet GPTs again identified requirements that students didn't trace back. The findings, illustrate that AI can help in the teaching of requirement tracing; these results suggest that while no AI model is currently capable of replacing real requirement engineers as they don't outperform students, it can be used as a tool to test the completeness of the requirement tracing process. We posit that GPT can be a tool for students to self-assess the degree to which their own requirements tracing is exhaustive.**

***Keywords—AI, Requirement Tracing, Education, Software Requirement Specification, Large Language Models***

## I. Introduction

Requirement tracing, also known as requirements traceability, is a crucial aspect of the Software Development Lifecycle (SDLC), which involves tracking and documenting the relationships between various elements of a software project, primarily requirements [1]. SDLCs ensure that the software being developed aligns with the initially defined requirements and helps in managing changes, testing, and quality assurance. Requirement tracing supports maintaining transparency and accountability throughout the SDLC [2], and connects every requirement with the artifact used to conceive the requirement. An artifact is a tangible document, file, or item that is generated during the requirement elicitation process [3]. By performing requirement tracing, the developers can visually show to the customer at what point during the requirement elicitation process the customer asked for any requirement specification, this is known as software validation [4]. Requirement tracing is usually performed after the requirements are written down, i.e., the engineers must go back and check every single document written for the instance at which every requirement was asked for [5].

Due to Large Language Models' (LLMs) Natural Language Processing (NLP) capabilities, can aid with requirement tracing, as they can understand and generating natural language texts [6]. They can analyze and extract information from the different artifacts used to generate the requirements. This paper aims to answer the following research question:

- *Can LLMs be used as a tool to teach students on requirements tracing and track their progress?*

This paper examines several scenarios in which different versions of GPT are given a list of requirements and a corresponding artifact that captures the source of requirements, and engineers a prompt to trace the requirements back to the artifacts.

The organization of this paper is as follows. Section II outlines the background knowledge required to understand the content of this paper, including ways to trace requirements and explains what LLMs are used for. Section III explains this research work's approach. Section IV displays the results of the experiment. Section V presents common uses for LLMs and

proposes how it would be possible to modify these uses for the different activities. Finally, section VI contains the authors' suggested future work and concluding statements.

## II. Background

Validation and Verification (V&V) are used within the SDLC to comprehensively determine that software is both correctly implemented and meets the specified requirements [7]. Although Verification and Validation bear similarities, they are inherently different and cater to distinct aspects of the software assurance process. Both must be distinctly understood to ensure the effective and accurate assessment of software products.

The IEEE Standard Glossary of Software Engineering Technology provides specific definitions for V&V. Verification is defined as the process of evaluating the system/components to determine if the product satisfies the given condition. This represents the need to continuously monitor the software's alignment with its design objective at every development stage. In contrast, validation is defined as the process of evaluating the system/components during or after development to determine if it satisfies requirements. This ensures that, beyond meeting design specifications, the software genuinely meets the end user's needs and the broader system objectives [8]. Software requirements are an agreed statement of what a proposed system must accomplish. Requirements can describe a system's functionality but also non-functional issues, constraints on the design, or implementation constraints [9].

For standard engineering projects, the stakeholders and subject experts will show their needs and requirements for a project. These requirements have to be elicited into requirements specifications by the development group to make these requirements understandable by the team members, represent all of the client's needs, and are testable. Stakeholder requirements must be elicited since they often do not account for all the intricacies or edge cases present when developing a large software system. Mismanaged requirements are initially small problems but can cause the complete failure of a software project [10, 11, 12, 13].

To prevent incorrect requirements, it has become standard practice to implement the process of eliciting and refining stakeholder needs into requirements. It is common belief that effective requirement elicitation is essential to a successful project [14]. Through requirement validation, incorrect requirements are found and removed [15]. Software requirements have also proven useful in helping to rank and organize functionalities of the system [16, 17].

For most software systems, there will be many requirements created during the elicitation phase. To ensure accountability between all groups, the process of requirement tracing is used. Requirement traceability describes the ability to follow a set requirement though its lifetime in a forward and backward manner [18]. There are many methods to implement requirement tracing. Most common approaches use a standardized matrix system [19], but implementations also exist using various techniques [20, 21] such as key phrases [22], scenarios [23], or ontologies [24].

NLP encapsulates a set of techniques used to achieve human-like language text processing. Generic goals of NLP are to paraphrase a text, translate it into another language, answer questions about its contents, and draw inferences from it. NLP has previously been able to satisfy the first three but was not able to routinely draw new inferences from a given text [25].

Building on the foundation of NLP, LLMs have recently begun an exponential upward trend in popularity. In 2020, Kaplan et al. produced a study that showcased and defined laws for the scaling properties between increasing parameter size and language replication ability [26].

Since then, LLMs have been researched extensively and have seen increasingly large performance results in tasks such as multi-step reasoning, instruction following, program execution, model calibration, and reading [27, 28].

## III. Approach

In the early stages of the Requirements Engineering phase of the SDLC, the engineering team and the customer meet to discuss and get more information about the project. During that phase, a series of models and artifacts are used to analyze the elicited customer requirements regarding what the project must provide to the different stakeholders. Finally, a series of requirement specifications are generated, which are documented within the SRS and serve as a contract on what the engineering team must deliver to the customer at the end of the project. Once the requirements specifications are written down, they must be traced back to the artifact that was used to elicit them. This is necessary to ensure the functionality delivered by the development team was requested by the customer at some point and to pinpoint exactly where and when it was asked for.

The approach used in this paper involved providing GPT with a transcript of a team of requirement engineers interviewing a customer. Additionally, GPT was provided with a list of the requirements specifications written by the developers and agreed to by the customer through the requirement elicitation process. Once GPT had the transcript and the list of requirements, GPT was prompted to determine how many of those final requirement specifications were asked for in the transcript. At the same time, a group of 20 software engineering master's students were asked to trace the same requirement specifications. A comparison of GPT's response was conducted against the requirement traceability matrix generated by the engineering team. Further, GPT was also asked to identify out of all the requirements in the requirement specification document, which ones it believed to have a prototype, which was provided, as the original artifact used to elicit those requirements. The response was analyzed against the requirement traceability matrix.

A common situation that is often encountered is when some needs are changed during the development of a product. As a result of the change in needs, some requirements must be changed, causing disruption in the development process. A

similar situation was used to test the students and GPT's capabilities of tracing modified requirements. After completing the initial tracing, some a scenario where the customer changed the requirements was conducted, causing modifications to the existing requirement specifications. After changing the requirements, both the teams and GPT were asked to trace the

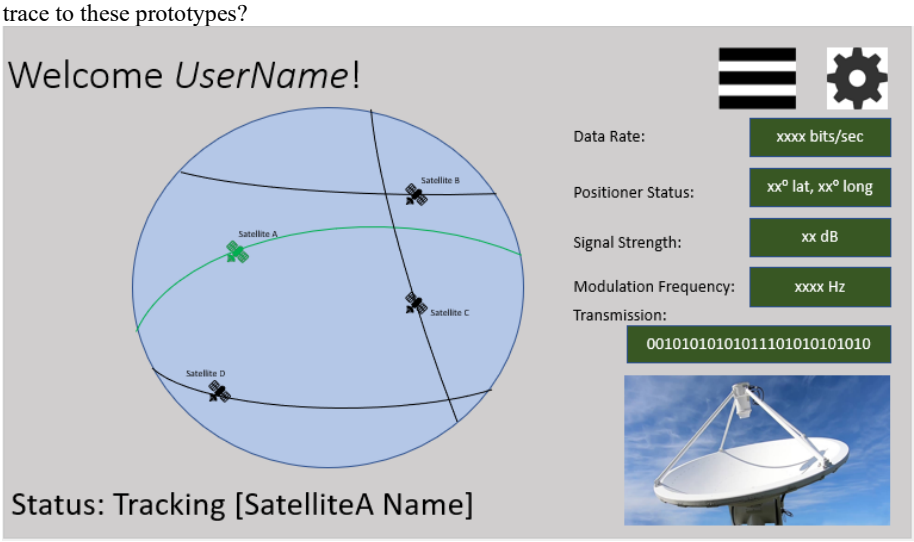
| # of prompt | Prompt used | # of requirements traced by GPT 3.5 | # of requirements traced by GPT 4.0. |
|---|---|---|---|
| 0 | I'm going to give you an interview verbatim between a customer and a software development team, a project description and a list of software requirements specifications and I want you to tell me how many of the requirements were asked by the customer<br><br>Question 1. Give brief description of system<br>Answer: The software will be the backbone of a ground station for satellite communications. The software controls the positioner of, so, I should first mention, the ground station is basically an antenna system that receives the signals from space, it can be a CubeSat, nano-satellite or spacecraft. So, basically, what it will do is you will have a positioner that moves antenna around to align to the satellite or the spacecraft and that positioner tracks the satellite/spacecraft as it moves around, you know, on the orbit and then as its tracking movement is receiving the data down to ground.<br><br>So, a few things that the software must do are know the orbit of the satellites with a few parameters and then the software must know when that orbit is visible for a particular location of the ground station. When the satellite is visible to the ground station, it then starts tracking from one side to the other. | | |
| 1 | These are the requirements. How many were asked for by the customer?<br><br>1. The system shall be able to determine the needed angular position (two angular axes) needed to track a spacecraft given the orbit parameters and time to perform the signal reception.<br>2. The system shall save a prioritized list, determined by user input, of spacecrafts to receive from.<br>3. The system shall let the user determine the priority through a numerical ranking.<br>4. The system shall not allow two or more satellites have the same priority.<br>5. The system shall compute when a tracked satellite will be visible for the ground station, given the ground station coordinates and the maximum reception angle from the horizon.<br>6. The system shall compute the expected frequency doppler shift for a given angular | 94 | 164 |
| 2 | How many of the requirements given were asked initially by the customer in the interview? | 159 | 255 |
| 3 | How many of these requirements are not explicitly mentioned in the transcript, but are requested by the customer? | 229 | 365 |
| 4 | I am going to give you a series of prototypes, how many of the provided requirements can you trace to these prototypes?<br>  | 287 | 379 |

*Table 1. Iteration of prompts and requirements specifications traced to interview and description.*

| GPT Version | # of requirements traced | # of requirements traced to interview or description | # of requirements traced to prototype | # of requirements total |
|---|---|---|---|---|
| 3.5 | 516 | 229 | 287 | 888 |
| 4 | 676 | 365 | 379 | 888 |
| Students | 888 | 399 | 490 | 888 |

*Table 2. Tracing breakdown for GPT-3.5, GPT-4 and, requirements engineering team.*

requirements specifications that had been changed. Finally, the results of both GPT versions used and those of the students were analyzed.

Table 1 shows the prompts used and a brief example of the information given to GPT to trace the requirements to the interview and description and the number of requirements traced because of the prompts used. The whole transcript is not displayed as it would take several pages, same as for the requirements, and the prototypes.

## IV. Results

Table 2 displays the results of the activities conducted. Both GPT-3.5 and GPT-4 were given a total of 888 requirements to trace, which were the result of the requirement elicitation process. Out of those 888, the engineering team traced back 399 (44.93%) to the interview between the customer and engineering team, and to the customer project description, and traced a total of 490 (55.18%) back to the protypes shown to the customer. In some cases, a requirement could be traced back to both a prototype and interview or prototype and project description, this happens when the customer shows need for a functionality in the interview or description, and the prototype displays that functionality visually for the customer to agree on how it looks. GPT-3.5 was capable of tracing 229 back to the interview or description, and 287 to the prototype. As expected, GPT-4 performed significantly better than GPT-3.5, as it was capable of tracing 365 to the interview and 379 to the prototype. This increase in performance from one version to other displays how much impact the increase of parameters in the training of GPT has on the performance of the LLMs.

When comparing the performance of the requirements engineering team to both GPT versions, it can be seen that when it comes to tracing back to the given interview or description, GPT-4 performs almost as well as the engineering team, while GPT-3.5's performance is considerably worse. Furthermore, neither GPT versions perform nearly as well as the engineering team when it came to tracing back to the prototype. The reason for this could be potentially the lack availability of high-quality training data [29]. One interesting note regarding the tracing performed by LLMs and the engineering group was that, while the students traced more requirements than GPT-4 to the interview, GPT-4 traced some requirements the group of students weren't able to trace back to the interview. This raises the idea that LLMs could be used to validate the process followed by students and ensure the tracing process is exhaustive to ensure all requirements are traced to the right artifact.

As can be seen in Table 1, GPT was unable to trace many requirements after the original prompt, therefore, modifications had to be made. The biggest improvement within the prompt iterations was between the second and third prompt. It is possible that GPT assumes the user asks for explicitly mentioned requirements unless specified. This could explain why specifying that the requirements asked to trace do not have to be explicitly mentioned in the artifacts, as it increases the total number of requirements traced.

Most of the requirements were not explicitly mentioned in the interview. This is where the training of the language models come into play. The most novel version, GPT-4, can understand the context better than 3.5 and therefore is able to trace more requirements that have related words, but not identical, to the interview and where they were asked for by the customer [30]. The level at which the LLMs can understand what a requirement means and can find the relationship with the interview is the main reason why the newest version of GPT outperforms the earlier. GPT-4 also outperforms GPT-3.5 when it comes to understanding which requirements could have used a prototype as the artifact. However, seems like both GPT

| GPT Version | # of modified requirements traced | # of requirements traced to prototype | # of modified requirements |
|---|---|---|---|
| 3.5 | 35 | 17 | 53 |
| 4 | 42 | 21 | 53 |
| Students | 53 | 35 | 53 |

*Table 3. Tracing modified requirements breakdown for GPT-3.5, GPT-4 and, requirements engineering team.*

versions are unable to get a full understanding of everything that is displayed within the image, hence why the requirements engineering team outperforms both.

Following the tracing of the requirements, the customer changed some of the needs for the system. As a result of these changes, a total of 53 requirements had to be modified. Once these modifications were completed, both GPT versions and the requirements engineering team were asked to trace the modified requirements back to the interview. As an addition, GPTs and the team were also prompted to trace the modified requirements to the interview too, to see which of the requirements were traced to both artifacts. Table 3, shows the modified requirements and how effective GPT-3.5, GPT-4 and the requirements engineering team were at tracing them back. GPT-4 outperforms GPT-3.5 once again at tracing the modified requirements back to the interview, tracing 42 and 35 respectively, and prototypes, 21 to 17. However, the performance of both GPT versions fall short to the engineering teams which traced all 53 modified requirements to the interview, finding 35 which were also traced to the prototype.

## V. Related Work

This section will discuss related work that leverages LLMs for different software development activities other than requirement tracing, along with current methods to perform validation for software systems and any methods that aim to automate this validation process.

When it comes to requirements validation, there have been several works that aim to improve the field. Some research focuses on already existing techniques that are common in the industry, such as inspections, requirements prototyping, reviews, viewpoint-oriented requirements validation, and use-case modeling. Some of these techniques are used to elicit requirements like prototypes and viewpoint-modeling, while inspections and reviews tend to be performed during the development of the software [31]. Other research made to improve requirement validation was done to suggest that the requirement validation process needed to consider the consistency, completeness and correctness or the requirements [32]. The authors proposed new methods of validating and managing the consistency of requirements to make the generation and management of requirements more streamline.

Following one step further on the research for requirements validation, requirements traceability was investigated as an area that could greatly improve the efficiency of requirement validation methods as traceability would serve as an easily visualized tool which the customer and developer can look back now a requirement was asked for. An idea that combined with requirement tracing could potentially become a breakthrough in the field were ontologies. Ontologies are sets of entities in a specific domain that shows their properties and relations between them [33, 34]. Requirement traceability matrices serve is a management tool that helps visualize where each requirement and artifact align, meaning that it helps show which artifact was used to generate each requirement. Ontologies were suggested as another method to visualize their relations [35, 36]. In essence, both ontologies and requirement traceability matrices are used for the same purpose, however, ontologies were initially created with the purpose of automation, which would speed up the process of manual generation of a requirement traceability matrix.

There are exiting tools used for requirement tracing such as DOORS [37], Rational Rose, or XDE among others [38]. These tools require the user to enter UML diagrams, code, requirements, and interfaces manually to allow for in-place traceability of artifacts. For large systems, this becomes a time-consuming task, which is why the use of LLMs as a tool to automate requirement tracing would simplify the current tracing process and make it much faster.

In the context of automation of the requirement traceability, an interesting approach was suggested that requirement traceability could be performed using Unified modeling Language (UML) models [39]. UML models were generated from different requirement elicitation models like Use case diagrams and scenarios, which would then turn into Activity diagrams and later Sequence diagrams. Since use case descriptions explain the actors involved, the pre- and post-conditions and action flows, they are used to describe the uses the different actors have for the system. Later, the activity and sequence diagrams aid to ensure the traceability between analysis and the design of the system. Machine learning models were also used to automate the requirement tracing process. Reinforcement learning was used to generate links between requirements and artifacts [40]. It was proven that reinforcement learning models could generate links documents with common words located close to each other faster than manually doing so, even though it was only capable of linking 73% of the concepts. In addition to machine learning, some deep learning has been applied to the requirement tracing process. Recurrent Neural Networks (RNNs) and pretrained LLMs were used showing that both models struggled to link between artifacts, having the highest performance of 72% [41]. The authors suggest that the main reason for this low performance could be the amount of data used for the pretraining of the LLMs.

On the other hand, LLMs are still in their infancy stage. It is quite unknown how much potential they have but are already being used in several fields, not only in regular life to perform simpler tasks, but also related to software development. One of these main uses is code explanation. GPT-3 has been used in the past as a tool to explain code given to it [42]. This way, a developer could focus on developing code and use an external tool to write the explanation for other developers or stakeholders to look at. LLMs have also been used for educational purposes. LLMs have been used to explain code for students in a way they could easily understand what a code is doing [43]. LLMs have also been used to walk students through their own code to double check its behavior [44]. This way students can make sure that their code is working as intended before submitting it to the instructor.

One of the key problems LLMs might encounter when generating code is that they write the code and assume it is correct, when indeed it is not. If you give them that same code, it can detect that the code doesn't work as intended. An iterative method that keeps asking LLMs if the code works to detect the errors in the code previously generated until the input and the output is the same was proposed as a possible solution [45]. Multi-program synthesis is very similar to iterative methods but instead of asking itself, it asks the user for feedback as to how the user would rate the code returned. Each time the user interacts with the model by giving feedback on how to improve the code or what aspects to change, the model improves the code to make it more tailgated to the user's needs [46]. However, using only one model to generate code could cause it to fall constantly into the same mistakes as it might not know where the error is in the first place. That is why using an ensemble of LLMs outputs was used to see if it could make an improvement in the quality of the code generated by LLMs [47]. The outputs from different GPT versions and InstructGPT were used to carry out this experiment. In 20-shot and 50-shot learning, the ensemble method outperformed individual LLMs by over 20% and 15% respectively.

Regarding requirements specifically, there has been research done on different applications of LLMs for them. The use of LLMs during the requirement elicitation process was investigated as a potential application. GPT-3.5 was used to elicit requirements and then compared its results to the requirements given by different experts from academia and industry. GPT generated requirements had high Abstract, Atomic, Consistent, Correct and Understandable scores but yielded lower scores in Unambiguity and Feasibility categories [48]. LLMs show promise for their use in the Requirements Engineering activities. Calculating the completeness of requirements using LLMs was also researched. BERT models were used to identify relevant but missing terminology within requirements [49]. Results showed that BERT could identify where there could be an instance of incompleteness but not always point out what the incompleteness is caused by. Other BERT models were used to standardize requirements. This standardization aims for the requirement specification to use the same structure when being written, following industry standards "The system shall…". To account for different types of requirements, such as design, functional, behavioral or performance requirements, the model generated several different templates to be used for each of those categories [50]. Our research, like the previous ones mentioned in this paragraph, aims to apply LLMs for requirements related purposes, however, this work focuses on tracing the origin of the requirement to the artifact used to generate it.

## VI. Conclusion

This paper showed that GPT-3.5 and GPT-4 can be used as additional tools to check generated requirement traceability matrixes, helping to validate requirements specifications. However, due mainly to the weak image processing capabilities from GPT, it is not a tool that can be used to validate requirements, as it will not get understanding of the prototypes, hence not tracing all the generated requirements to the artifact used to generate it.

The students used for this experiment outperformed both GPT versions. One of the potential reasons was the quality of the prompts used. It is possible that with different prompts that asked GPT to carry out the same task, its performance would have improved.

There is a significant performance improvement when using GPT-4 compared to GPT-3.5 as the former used a much larger training dataset. This resulted in GPT-4 being able to track many more requirements than the earlier version. Not only that, but GPT-4 was also able to figure out more requirements that were related to prototypes or other sort of visual artifacts, making them impossible to trace back to the interview transcript. One of the main reasons for the difference in performance might be a difference in context understanding that there is between GPT-4 and GPT-3.5. Since the input window for GPT-3.5 was not big enough to accept whole batches of requirements at once, they had to be broken down GPT-4's input window was large enough to accept whole lists.

Undoubtedly, AI can help in the teaching of requirement tracing; the results suggest that while no AI model is currently capable of replacing real requirement engineers as they don't outperform students, it can be used as a tool to test the completeness of the requirement tracing process as it was capable of tracing some of the requirements students weren't able to trace. This paper shows the promise that GPT can be a tool for students to self-assess the degree to which their own requirements tracing is complete. Furthermore, LLMs could be used as an assessment tool to evaluate student's performance in different software engineering related activities, such as software design and modeling, by tracking the models and ensuring they don't contradict each other.

## VII. Future Work

This research focused on validation and verification of requirement specifications. In the future, we would like to implement this method and build a class around leveraging LLMs in *all* steps of the requirements engineering process. Thus, future research should explore the efficacy of LLMs in eliciting requirements as well as generating requirements specifications that meet the customer's needs. New technologies are developed every day, and it is necessary to adapt to be able to utilize them efficiently. Breaching the gap between what is taught in the classroom and what is used in the industry is pivotal to student's success. By designing a class that uses the method proposed in this paper, students could be better prepared to use new technologies after their education is complete instead of having to adapt to them while on the job.

## VIII. References

[1] K. E. Wiegers and J. Beatty, Software requirements, Pearson Education, 2013.

[2] P. A. Laplante and M. H. Kassab, Requirements Engineering for Software and Systems, CRC press, 2022.

[3] R. Tsuchiya, K. Nishikawa, H. Washizaki, Y. Fukuzawa, Y. Shinohara, K. Oshima and R. Mibe, "Recovering transitive traceability links among various software artifacts for developers," *IEICE TRANSACTIONS on Information and Systems,* vol. 102, no. 9, pp. 1750-1760, 2019.

[4] R. W. Adrion, M. A. Branstad and J. C. Cherniavsky, "Validation, verification, and testing of computer software," *ACM Computing Surveys (CSUR),* vol. 14, no. 2, pp. 159-192, 1982.

[5] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel and F. G. Oliveira Neto, "Requirements engineering challenges and practices in large-scale agile system development," *Journal of Systems and Software,* vol. 172, p. 110851, 2021.

[6] P. Vaithilingam, T. Zhang and E. L. Glassman, "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models," in *CHIEA '22: Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems*, 2022.

[7] D. Wallace and R. Fujii, "Software verification and validation: an overview," *IEEE Software,* vol. 6, no. 3, pp. 10-17, 1989.

[8] "610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology," IEEE, 1990.

[9] J. W. Brackett, "Software Requirements," Carnegie-Mellon Univ Pittsburgh PA Software Engineering Institution, 1990.

[10] A. M. E. O. K. F. M. Hussain, "The Role of Requirements in the Success or Failure of Software Projects," in *International Review of Management and Marketing*, 2016.

[11] The Standish Group, "The CHAOS Report," The Standish Group International, Inc., 1994.

[12] R. N. Charette, "Why Software Fails," *IEEE Spectrum,* vol. 49, no. 9, pp. 42-49, 2005.

[13] E. Bjarnason, K. Wnuk and B. Regnell, "Requirements are slipping through the gaps — A case study on causes & effects of communication gaps in large-scale software development," in *IEEE 19th International Requirements Engineering Conference*, 2001.

[14] S. Sankhwar, V. Singh and D. Pandey, "Requirement Engineering Paradigm," *Global Journal of Multidisciplinary Studies,* vol. 3, no. 3, pp. 290-297, 2014.

[15] A. Aggarwal and R. Kaur, "An Experimental Analysis of Software Requirement Traceability and Enhancement of Quality of Traceability Process using Quality Assurance Interface," *International Journal of Innovative Technology and Exploring Engineering (IJITEE),* vol. 8, no. 8S3, pp. 515-518, 2019.

[16] B. T.E. and T. T.A., "Software requirements: Are they really a problem?," in *International Conference on Software Engineering*, 1976.

[17] J. Karlsson, "Software Requirements Prioritizing," in *International Conference on Requirements Engineering*, 1996.

[18] O. Gotel and A. Finkelstein, "An analysis of the requirements traceability problem," in *Proceedings of IEEE International Conference on Requirements Engineering*, 1994.

[19] R. Wieringa, "An Introduction to Requirements Traceability," Vrije Universiteit, 1995.

[20] A. Egyed, "A scenario-driven approach to trace dependency analysis," *IEEE Transactions on Software Engineering,* vol. 29, no. 2, pp. 116-132, 2003.

[21] R. Watkins and M. Neal, "Why and how of requirements tracing," *IEEE Software,* vol. 11, no. 4, pp. 104-106, 1994.

[22] J. Jackson, "A Keyphrase Based Traceability Scheme," Racal Radar Defence Systems Ltd, 1991.

[23] M. F. Bashir and M. A. Qadir, "Traceability Techniques: A Critical Study," in *IEEE International Multitopic Conference*, 2006.

[24] Z. Li, M. Chen, L. Huang, V. Ng and R. Geng, "Tracing Requirements in Software Design," in *Proceedings of the 2017 International Conference on Software and System Processes*, 2017.

[25] E. D. Liddy, "Natural Language Processing," School of Information Studies - Syracuse University, 2001.

[26] J. Kaplan, S. McCandlish, T. Henighan, T. B. C. B. Brown, R. Child, S. Gray, A. Radford, J. Wu and D. Amodei, "Scaling Laws for Neural Language Models," arXiv:2001.08361, 2020.

[27] J. Wei, Y. Tay, R. Bommasani, C. Raffel, B. Zoph, S. Borgeaud, D. Yogatama, M. Bosma, D. Zhou, D. Metzler, E. H. H. T. Chi, O. Vinyals, P. Liang, J. Dean and W. Fedus, "Emergent Abilities of Large Language Models," arXiv:2206.07682, 2022.

[28] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals and L. Sifre, "Training Compute-Optimal Large Language Models," arXiv:2203.15556, 2022.

[29] P.-P. Vázquez, "Are LLMs ready for Visualization?," *arXiv preprint arXiv:2403.06158,* 2024.

[30] C. Xiao, S. X. Xu, K. Zhang, Y. Wang and L. Xia, "Evaluating reading comprehension exercises generated by LLMs: A showcase of ChatGPT in education applications," in *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, 2023.

[31] H. A. Bilal, M. Iluas, Q. Tariq and M. Hummayun, "Requirements Validation Techniques: An Empirical," *International Journal of Computer Applications,* vol. 148, no. 14, 2016.

[32] M. Kamalrudin and S. Sidek, "A Review on Software Requirements Validation and Consistency Management," *International journal of software engineering and its applications,* vol. 9, no. 10, pp. 39-58, 2015.

[33] S. Staab and R. Studer, Handbook on ontologies, Springer Science & Business Media, 2010.

[34] M. Uschold and M. Gruninger, "Ontologies: Principles, methods and applications," *The knowledge engineering review,* vol. 11, no. 2, pp. 93-136, 1996.

[35] M. S. Murtazina and T. V. Avdeenko, "An ontology-based approach to support for requirements traceability in agile development," *Procedia Computer Science,* vol. 150, pp. 628-635, 2019.

[36] V. Adithya and G. Deepak, "OntoReq: an ontology focused collective knowledge approach for requirement traceability modelling," in *European, Asian, Middle Eastern, North African Conference on Management & Information Systems*, 2021.

[37] J. Wegener and U. Herold, "Requirements and test case tracing," in *Embedded Real Time Software and Systems (ERTS2012)*, 2012.

[38] J. Lin, C. C. Lin, J. Cleland-Huang, R. Settimi, J. Amaya, G. Bedford, B. Berenbach, O. B. Khadra, C. Duan and X. Zou, "Poirot: A distributed tool supporting enterprise-wide automated traceability," in *14th IEEE International Requirements Engineering Conference (RE'06)*, 2006.

[39] K. Yoshino and S. Matsuura, "Requirements traceability management support tool for UML models," in *Proceedings of the 2020 9th International Conference on Software and Computer Applications*, 2020.

[40] H. Sultanov and J. H. Hayes, "Application of reinforcement learning to requirements engineering: requirements tracing," in *2013 21st IEEE International Requirements Engineering Conference (RE)*, 2023.

[41] J. Lin, Y. Liu, Q. Zeng, M. Jiang and J. Cleland-Hunag, "Traceability transformed: Generating more accurate links with pre-trained bert models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021.

[42] D. Zan, B. Chen, F. Zhang, D. Lu, B. Wu, B. Guan, W. Yongji and J.-G. Lou, "Large language models meet nl2code: A survey," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2023.

[43] S. MacNeil, A. Tran, A. Hellas, J. Kim, S. Sarsa, P. Denny, S. Bernstein and J. Leinonen, "Experiences from using code explanations generated by large language models in a web software development e-book," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*, 2023.

[44] S. Sarsa, P. Denny, A. Hellas and J. Leinonen, "Automatic generation of programming exercises and code explanations using large language models," in *Proceedings of the 2022 ACM Conference on International Computing Education Research-Volume 1*, 2022.

[45] J. Zamfirescu-Pereira and B. Hartmann, "Iterative Disambiguation: Towards LLM-Supported Programming and System Design," 2023.

[46] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," *arXiv preprint arXiv:2203.13474.,* 2022.

[47] X. Wang, S. Li and H. Ji, "Code4Struct: Code Generation for Few-Shot Event Structure Prediction," in *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada, 2023.

[48] K. Ronanki, C. Berger and J. Horkoff, "Investigating ChatGPT's Potential to Assist in Requirements Elicitation Processes," *arXiv preprint arXiv:2307.07381,* 2023.

[49] D. Luitel, S. Hassani and M. Sabetzadeh, "Improving Requirements Completeness: Automated Assistance through Large Language Models," *arXiv preprint arXiv:2308.03784,* 2023.

[50] A. Tikayat Ray, B. F. Cole, O. J. Pinon Fischer, A. P. Bhat, R. T. White and D. N. Marvis, "Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models," *Systems,* vol. 11, no. 7, p. 352, 2023.