Supporting Software Maintenance with Dynamically Generated Document Hierarchies

Katherine R. Dearstyne
Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
kdearsty@nd.edu

Alberto D. Rodriguez
Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
arodri39@nd.edu

Jane Cleland-Huang
Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
JaneClelandHuang@nd.edu

***Abstract*—Software documentation supports a broad set of software maintenance tasks; however, creating and maintaining high-quality, multi-level software documentation can be incredibly time-consuming and therefore many code bases suffer from a lack of adequate documentation. We address this problem through presenting HGEN, a fully automated pipeline that leverages LLMs to transform source code through a series of six stages into a well-organized hierarchy of formatted documents. We evaluate HGEN both quantitatively and qualitatively. First, we use it to generate documentation for three diverse projects, and engage key developers in comparing the quality of the generated documentation against their own previously produced manually-crafted documentation. We then pilot HGEN in nine different industrial projects using diverse datasets provided by each project. We collect feedback from project stakeholders, and analyze it using an inductive approach to identify recurring themes. Results show that HGEN produces artifact hierarchies similar in quality to manually constructed documentation, with much higher coverage of the core concepts than the baseline approach. Stakeholder feedback highlights HGEN's commercial impact potential as a tool for accelerating code comprehension and maintenance tasks. Results and associated supplemental materials can be found at https://zenodo.org/records/11403244.**

***Index Terms*—Requirements, Hierarchy, Documentation, LLM**

I. Introduction

Software documentation supports a broad set of software maintenance tasks such as impact analysis, change analysis, requirements validation, safety assessment, and new developer onboarding [1], [2], [3], [4], [5], yet, creating and maintaining consistent multi-level software documentation and its associated trace links is incredibly time-consuming [6], [7], [8]. The process of documenting, defining, and maintaining documentation that describes the implemented system is often viewed as overly burdensome by developers and stakeholders. This perception leads to the documentation process being ignored, delayed, or inadequately sustained [9], [10], [11], especially in startups and small companies where speed is often prioritized over comprehensive requirements engineering processes [12], [13]. Consequently, despite the many benefits of a systematic software documentation process, many code bases suffer from a lack of adequate documentation [14].

While there have been advancements in automating certain types of software documentation, such as API specifications or the continuous deployment of embedded software documentation ([15], [16]), efforts to automate the generation of comprehensive, multi-layered artifacts describing system features remain underexplored. With the advancements of large language models (LLMs) and their generative capabilities, there is now a path towards generating multi-layered, just-in-time software documentation; however, the challenge is in ensuring that the documentation correctly represents the underlying code base, is readable, understandable, well formatted, and properly organized so that it is useful to practitioners maintaining software systems [17], [18]. In pursuit of this goal, we present HGEN, an automated pipeline that generates multi-layer hierarchy of documentation, comprised of artifacts such as low-level design descriptions, as well as sub-system and system-level requirements formatted according to the norms of the currently adopted life-cycle process. HGEN not only constructs these artifacts but also generates trace links that connect them into a meaningful hierarchy, providing well organized documentation, designed to effectively support diverse software maintenance activities. We provide examples to the generated documentation for two open source datasets[1].

This paper first describes the HGEN process, providing a simple running example taken from the open-source gaming domain. We then report results from two studies evaluating HGEN in which we first assessed the quality of the HGEN generated hierarchy for three different projects, and then used it to generate documentation for nine industrial pilot projects using our partners' project data. In the first study, we recruited a key developer from each of the three projects to compare HGEN's generated documentation against their own project's manual documentation and against an off-the-shelf LLM baseline. For each project, we systematically evaluated the quality of the documentation by assessing the individual quality of each generated artifact, the overall coverage of concepts, and the relationships between layers. In the second study, we used HGEN to generate documentation for source code provided by our partners and then performed a think-

[1] Example of generated documentation:

Dronology: app.safa.ai/demo?version=a05d072b-163c-4ba5-a248-0683d1e2dda5&to=/project

JOC: app.safa.ai/demo?version=99965515-cbc1-43e9-b834-4815f22bd2e6&to=/project.

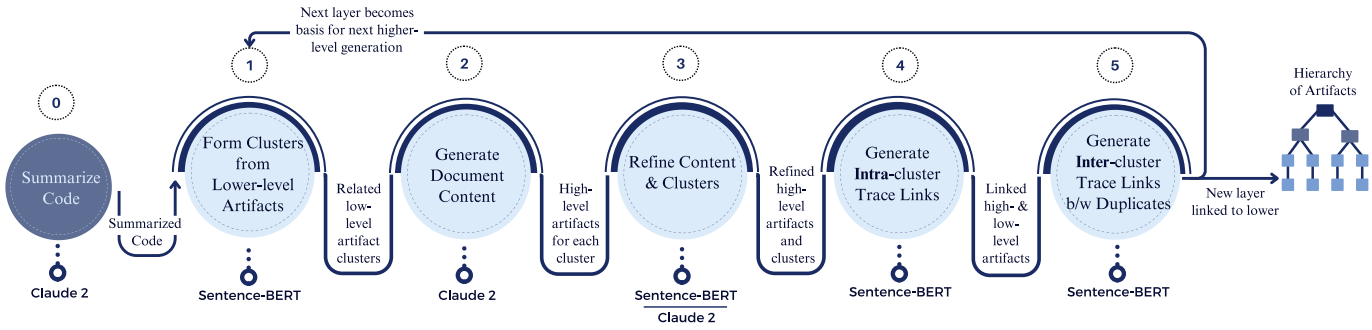

Fig. 1. The HGEN Process utilizes a pipeline to produce each layer of the documentation hierarchy. The lowest layer accepts source code as input and generates a natural language summary (Step 0). Steps 1-5 form a pipeline, which is used to generate each subsequent layer, thereby incrementally constructing a hierarchy of progressively higher-level artifacts formatted according to the norms of the current software development process. For each step in the pipeline, we show the underlying AI models used to support the transformation of lower-level to higher-level documentation.

aloud study in which we collected their feedback and analyzed it using an inductive approach.

The remainder of this paper is laid out as follows. Section II presents our process for generating the software artifact hierarchy. Section III describes the design on our experiment leading Section IV to present our quantitative evaluation of the HGEN process and Section V to present the qualitative feedback. Section VI presents related work on generating software documentation, code summarization, and code understanding, while Section VII discuss threats to the validity of our study. Finally, Section VIII summarizes the overall benefits of our approach and describes future work.

II. HGEN Process

The HGEN process represents a pipeline for generating hierarchies of software artifacts. As depicted in Figure 1, it includes five stages where Stage 1 accepts a set of *lower-level* artifacts as inputs, Stages 2-4 perform internal processing, and Stage 5 produces the *higher-level* artifacts that constitute a new layer of documentation. This new layer is then passed as inputs into Stage 1 to restart the process for creating the next layer. The stages for generating a single layer of documentation are therefore as follows:

Stage 1. Accept a set of lower-level natural language artifacts and perform clustering on them to identify related features and/or functionality. In the special case of the lowest-level, where the inputs are source code artifacts, perform an additional pre-processing stage (Stage 0) to generate a natural language summarization of the code. This summary serves as a proxy for the source code throughout the remaining steps. Upon the conclusion of this stage, a set of clusters of lower-level artifacts is generated as output.

Stage 2. For each cluster identified in Stage 1, generate a natural language description using the targeted artifact format (e.g., user story, feature description etc). This serves as the body of the new layer of documentation.

Stage 3. Refine the content of any artifacts that contain overlapping information to improve clarity, conciseness, and ensure each artifact focuses distinctly on one specific feature or functionality.

Stage 4. Connect these refined artifacts to the lower-level input artifacts by dynamically generating trace links.

Stage 5. Leverage the overall perspective provided by the relationships established in Stage 4 to detect and remedy redundant artifacts, and to produce the final set of output artifacts for the current layer.

Stage 6. If a higher-layer of artifacts is desired, pass these output artifacts as inputs to the next layer. Continue this process until all targeted layers have been generated.

The end result is a hierarchy of software artifacts, referred to as an artifact tree. Given the transformation that occurs during the generation of a single layer, we made numerous design decisions concerning the stages of the pipeline, the tasks assigned to each stage, and the algorithmic solutions for accomplishing each task. Each stage, and the overall sequence of stages, was designed as the result of trial-and-error in which we evaluated various techniques and their combinations. We followed a robust process based on the Design Science methodology, in which each design iteration included problem investigation, design and validation, and implementation and evaluation [19]. In earlier iterations, validation was performed internally by the researchers, while in later iterations it was performed by external Software Practitioners. The final outcomes of their evaluation are reported in Section III of this paper.

To support our description of the HGEN process, we've chosen a small, straightforward code repository from a CS101 project as a running example. We refer to this as HERO throughout the remainder of this paper. HERO is not associated with the paper's authors and is openly accessible at https://github.com/gbaman/QUB-CSC1011-Module-Hero-Game. Due to space constraints, we do not detail the intermediate stages of the design that led to the finalized process, and focus instead on describing the end result in the following sections. Overall, our process is designed to be model-agnostic and therefore we do not present a detailed empirical comparison of results based on different LLM model types in this paper, and discuss this decision further in Section VII. We now outline each stage in our HGEN pipeline.

Hero.java: This code provides the framework for a user to take control of a hero character within a digital game. Upon initialization, the hero is placed in a starting location on a virtual map. Lists of crimes for the hero to address and playable characters they can select are automatically generated. The user is then able to view the hero's character details and current status. As the user navigates the hero through the game world and engages in activities to resolve crimes, their total action value increases. Periodically checking this action value triggers different game states - once a threshold is reached, the user achieves victory and the gameplay loop restarts from the main menu. The user can also adjust their hero's action directly to progress the story at will. Throughout, the code integrates the hero character with the overall game system to immerse the user in an interactive experience where they guide the actions and challenges of their virtual protagonist.

Fig. 2. As part of our running example, HGEN summarizes the HERO source code in the preprocessing Stage 0.

A. Stage 0: Code Summarization

The lowest level of the documentation hierarchy starts with source-code, and therefore a pre-processing step is applied to summarize the code into natural language. This step serves two key purposes. First, it allows us to transform source code into natural language comparable to the input artifacts of all other documentation layers. Second, the summary has higher information density and less redundancy than raw code. This enables a larger amount of information to be conveyed within a single context window of the LLM, thereby enhancing its capacity to comprehend a broader scope of the system. Summarization tasks are best performed using generative models; therefore, we opted to use Anthropic's Claude 2.0 model, which returns similar results to OpenAI's GPT-4 [20] and has a large context window of 100-k tokens [21]. We prompted Claude to summarize the source code by (i) initially outlining the functionality provided to the user by the code, and (ii) then creating a polished summary that explains how the code supports the described user behavior. The resulting summarized output becomes the starting input for HGEN, representing the initial tier in the documentation hierarchy. The summary dynamically generated by HGEN for $Hero.java$ in the HERO code-base is depicted in Figure 2.

B. Stage 1: Form clusters from Lower-level Artifacts

We adopted a multi-technique clustering approach with the following internal steps, labeled C1-C8.

C1. *Preprocessing:* We start by converting the natural language artifacts into embeddings using the Sentence-BERT transformer. This choice is driven by the model's capacity to encode entire sentences rather than relying on word-level encoding, as well as its consistent performance across a diverse range of tasks. As a result, Sentence-BERT is used for all transformations to embeddings throughout the remainder of this paper.

C2. *Multi-Technique Clustering:* Early experimentation showed various unsupervised clustering algorithms each had their own unique strengths and limitations. We therefore ultimately adopted a consensus-based approach and included five different techniques to achieve diversity of cluster size, outlier detection, and geometric considerations [22]. These techniques were OPTICS [23], Spectral [24], Agglomerative [25], Affinity Propagation [26] and K-means [27]. In this step, each technique was used to individually cluster the vector representations of the input artifacts, producing a diverse set of candidate clusters.

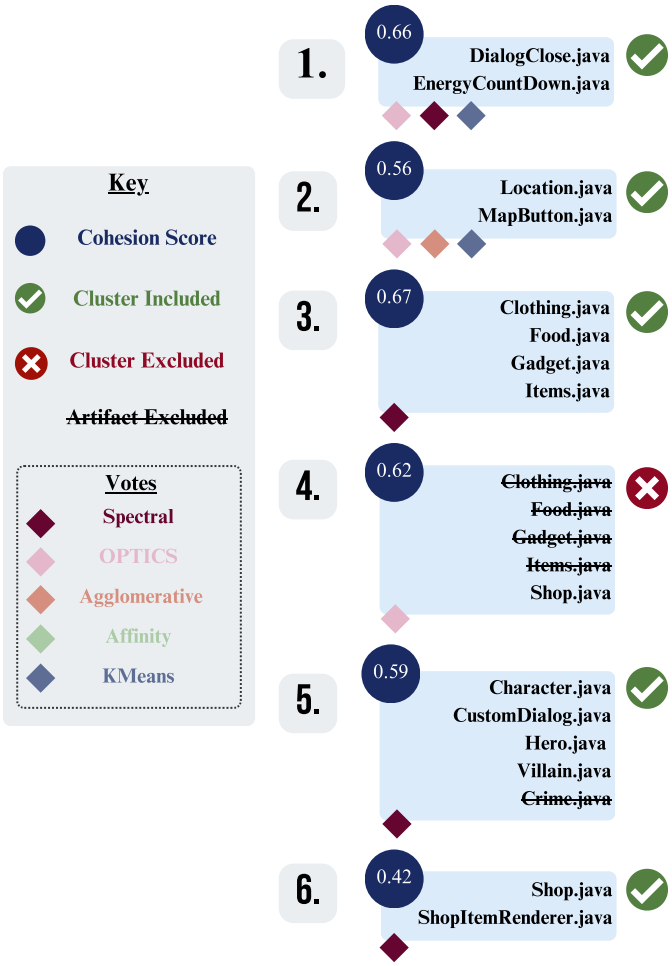

Fig. 3. In Stage 1, HGEN uses multiple clustering algorithms to produce and initial set of clusters from the source code summaries, and then performs a series of filtering, ranking, and cleansing steps on the clusters.

C3. *Filter by Size* The set of generated clusters were highly diverse but included overlapping and redundant clusters. We therefore applied the following filtering steps, starting by eliminating two types of clusters:

- *Singletons:* Temporarily set aside singleton clusters containing only one artifact.
- *Large Clusters:* Discard large clusters containing five or more artifacts, as these tend to inhibit the LLM's ability to identify and extract finer details. While the decision to remove large clusters limits the potential for constructing higher-level abstractions across larger artifact groups, we partially address this later in Step 3, by allowing clusters to re-form.

C4. *Cluster Scoring:* We assign an importance value to each remaining cluster as follows:

$$importance = (\alpha \cdot \log(s) + h) \cdot v \quad (1)$$

where:

- h is the cohesion score for the cluster,
- v is the voting score from the five clustering techniques,
- s represents the cluster size,
- α is the weight applied to the cluster size.

The voting score (v) is computed by counting the number of times the exact cluster, with the same input artifacts, appears in the candidate pool.

Cohesion (c) is computed by averaging the cosine similarity of each artifact's embedding to all its neighbors within a cluster as follows:

$$cohesion = \frac{2}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j \neq i}^{N} \cos(\theta_{ij}) \quad (2)$$

Finally, the size (s) metric is used to compensate for the tendency for smaller clusters to have higher cohesion. It is computed as the log of the number of input artifacts, weighted by a small constant ($alpha$). The use of logarithm moderates the impact of larger clusters, ensuring that their contribution to the importance score grows at a decreasing rate and prevents them from disproportionately dominating the score due to their size alone.

C5. *Cluster Ranking:* Clusters are then ranked in descending order by importance score. Fig. 3, depicts several clusters generated from the code base arranged by their respective importance scores. While clusters 1 and 2 have lower cohesion scores than cluster 3, their overall score places them higher in the ranking.

C6. *Cluster Cleansing:* Artifact outliers that deviate by 1.5 standard deviations or more from the average similarity to their neighbors are removed from their clusters to eliminate dissimilar artifacts from the cluster. For example, Crime.java is removed from cluster 5 in Fig.3.

C7. *Cluster Selection:* Next, we iterate through the clusters in order of their importance to determine whether to select them for the final set. At each iteration, we consider the cluster possessing the next highest importance score (termed the $FocusCluster$), alongside the set of clusters already chosen for inclusion (referred to as the $InclusionSet$). Given the initial prevalence of overlapping or redundant clusters in our consensus-based approach, we first assess whether the $FocusCluster$ contains artifacts not already present in the $InclusionSet$. After removing any artifacts shared with clusters in the $InclusionSet$, we admit the $FocusCluster$ only if it maintains a size of two or more artifacts and has a cohesion score greater than or equal to the top 75% of clusters. This process is exemplified in Cluster 4 from Fig. 3. This cluster contains four artifacts already included in Cluster 3, so each of these artifacts are removed, leaving the cluster with only one unique artifact. As a result, it fails to meet the size threshold and is therefore excluded from the final cluster set.

C8. *Handle Orphans* Finally, we check for artifacts not assigned to a cluster. For each orphaned artifact, we identify its most similar cluster by computing the average cosine similarity between the orphan and all members of each cluster. If the similarity is close to the cluster's overall cohesion score (within 0.1), it indicates that the orphan can be added without reducing the overall cohesion of the cluster. As a result, the orphan is incorporated into that cluster. Finally, any unplaced orphans are retained as singleton clusters.

C. Stage 2: Generate Documentation Content

Prior studies have shown the importance of well-formatted documentation [17]; therefore this stage focuses on formatting the generated artifacts according to the stakeholder's needs. For example, a user might wish to generate an agile documentation hierarchy composed of source code (lowest layer), user stories (middle layer), and epics (top layer); or they might wish to generate a traditional hierarchy composed of source code, design specifications, and multiple layers of requirements. In this stage we prompt the LLM to format the output of the desired artifacts by specifying (i) the output artifact type, (ii) the desired format of the artifact, and (iii) the targeted number of document artifacts to be generated from the current cluster.

The artifact type is specified by the user, while the format can either be predefined by the user or generated by the LLM in a separate context window. In this study, we used the latter approach to increase the degree of automation. Given that most LLM's pre-training data contains examples of diverse common artifact types, we can simply prompt Claude to generate a standardized format for the artifact type. For instance, Claude generated the following user story template: "As a [type of user], I want to [action or goal] so that [reason or benefit]".

Finally, we define the number of high-level artifacts (`n_targets`) to be generated for each cluster by considering two factors. First, *cohesion* measures the extent to which an artifact focuses on a single topic. Seemingly, clusters with low cohesion typically encapsulate more topics and require more higher-level elements. We therefore compute "concept diversity" as the inverse of the cohesion score (cf Eq. 2), normalized so that the maximum "concept diversity' for the project equals 1. Second, the amount of information within a cluster's artifacts plays a key role in determining the number of higher-level artifacts required. We estimate the *information density* of the cluster by comparing the size of its artifacts to the average size of all artifacts of the same type. Finally, we calculate the number of targeted artifacts (i.e., `n_targets`) as the product of "concept diversity" and "information density". To promote the emergence of a tree-like documentation structure, we impose a constraint that `n_targets` must be greater than 50% and less than 100% of the current cluster's artifact count.

Returning to the HERO example, we determine that Cluster 5 (see Figure 3) requires three higher level artifacts. The four code files have a total of 730 LOC (lines of code), while the average file in this layer has 109 LOC. We estimate information density to be approximately 6.7 ($730/109$), reflecting the complex game logic contained within these core character-

related classes. Normalized concept diversity is computed as 0.56, leading to `n_targets` being three artifacts i.e., by computing and truncating $0.56\times$ 6.7. Figure 4 shows the three subsequent user stories generated for Cluster 5 in our example.

[US1] Customize Character Name and Image: As a player, I want to be able to customize my character's name and image so that I can personalize my gameplay experience.

[US2] View Character Inventory and Money: As a player, I want to be able to view my character's inventory and money so that I can make informed decisions when interacting with the game world.

[US3] Progress Character Through Story: As a player, I want to be able to commit crimes and take heroic actions that will progress my character through the game's story and scenarios.

Fig. 4. User stories generated from HERO source code during Stage 2. All 3 user stories were produced from the same cluster of source artifacts (see Cluster 5 from Figure 3).

D. Stage 3: Refine Content & Clusters

Because automatic clustering may not always match human judgment, some level of conceptual overlap across clusters is inevitable, resulting in duplicated content in the generated artifacts. Stage 3 addresses this issue by reducing duplicated content and refining the artifact clusters through three steps, labeled D1-D3.

D1. *Duplicate Identification*: To identify potential duplicates, we cluster the generated artifacts using the algorithm described in Section II-B. This creates groups of similar artifacts that are currently spread across different clusters. The most cohesive clusters are those most likely to contain duplicated content and thus are identified as *duplicate clusters*. For example, in HERO, US1 (Fig. 4) is a generated artifact which is detected as similar to other character-related user stories from different clusters (US4, US5) (see Figure 5).

[US1] Customize Character Name and Image

[US4] Customize Character Identity: As a player who wants an immersive role playing experience, I want to be able to customize a character with a name and choose to be a hero or villain so that I can define my virtual identity in the world

[US5] Character Entity Templates for Game Testing: As a game developer, I want the system to allow defining character entities via reusable templates and validated testing so that playable characters can be reliably generated with consistent expected behaviors for use in game scenarios.

Fig. 5. Generated user stories for HERO that were clustered together in Stage 3. Although each user story originated from a different cluster in Stage 2, they were clustered together in Stage 3 due to their shared focus on character customizations.

D2. *Duplicate Content Identification*: In this step, our aim is to determine what source artifacts led to the overlapping content so they can be re-clustered together. We identify the source artifacts contributing to the overlap by selecting those with the highest semantic similarity to the parent. Then, a new cluster is formed containing the selected source artifacts for each generated artifact in the duplicate cluster.

D3. *Re-generation*: At this stage, each duplicate cluster has identified the source artifacts containing the overlapping content. Now, our goal is to give the LLM a chance to regenerate new artifacts based on this focused context. Given the set of source artifacts, we repeat Stage 2 in order to generate a fresh set of artifacts centered around the core theme. These new artifacts replace those in the duplicate clusters. In our example, the overlapping artifacts were re-generated as shown in 6, where each artifact is now focused on a more distinct topic.

[US1*] Character Customization: As a player, I want to name and customize the appearance of my character so that I can roleplay a unique persona in the game world.

[US4*] Play as Hero or Villian: As a player, I want the option to play as either a hero or villain so that I can experience different perspectives when interacting with the game systems.

Fig. 6. Refined User Stories for HERO during Stage 3

E. Stage 4: Generate ***Intra***-cluster Trace Links

Once the new artifacts have been generated, we need to connect them via trace links to the current layer. Given that a single cluster can produce multiple higher-level artifacts, we cannot assume every lower-level artifact in a cluster should link to each of the resulting higher-level ones. Consequently, we create trace links only between artifacts that demonstrate strong semantic similarity using standard automated tracing techniques. First, we generate embeddings for the higher-level artifacts and use these to calculate their cosine similarity with each low-level artifact from their originating clusters. We scale each cluster's scores using min-max scaling so that the highest score is adjusted to 1. We consider the variability in scores across different clusters, and only generate links where the similarity score is within two standard deviations of the maximum normalized score. Typically, this results in a cutoff of approximately 0.8. However, if no links are generated for a lower-level artifact, we establish a link with the higher-level artifact that has the greatest similarity.

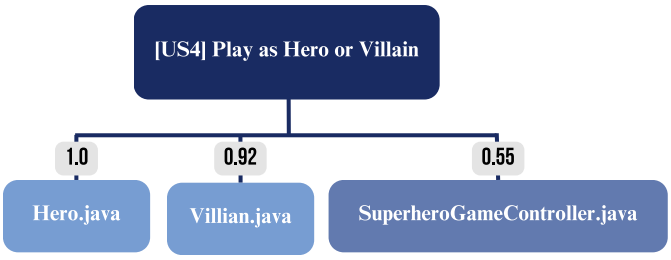

Fig. 7. Example of trace link generations for US4 from the HERO example.

For example, the trace links established for US4 (Figure 6) are depicted in Figure 7. Following scaling, both $Hero.java$ and $Villain.java$ attain high similarity scores and are consequently linked to the user story. Although $SuperheroGameController.java$ receives a considerably lower score, its similarity to US2 exceeds its scores with all other user stories, allowing it to trace to US2 as well.

*F. Step 5: Generate **Inter**-cluster Trace Links between Duplicates*

In this step, we identify any remaining artifacts with overlapping content, which also aids in detecting potential trace links between clusters. First, we compute the cosine similarity between each pair of generated artifacts, marking pairs with similarity scores more than two standard deviations above the mean as potential duplicates. For each duplicate pair, designated as A and B, we consider whether any of B's trace links should also trace to A and vice verse. Trace links are formed if the similarity score between B and the child is of a similar strength between A and the child (i.e., within a difference of 0.1).

If two pairs of highly similar artifacts result in trace links with identical child artifacts, it signifies that the pair are likely duplicates. In such cases, we remove one of the duplicates, as the risk of losing crucial information is significantly reduced.

An illustration of this re-tracing process can be seen in Figure 8. Artifacts that were originally traced to each user story are shown in blue. After re-tracing, each user story gains an additional trace link, represented in white. Notably, US3 possesses one trace link ($Crime.java$) not linked to US4; however, in this example, both US3 and US4 are retained after this stage, as US3 is linked to $Crime.java$, while US4 is not.

III. Experiment Design

Evaluating the effectiveness of documentation hierarchies is complex because there is no single ground-truth solution [28], [29], [30]. While it is tempting to use automated assessment techniques such as BLEU, METEOR, ROUGE, CIDEr, and SPICE, to detect overlapping terms across documents, Hu et al., showed that the metrics do not align with human judgment about the quality of documentation [31]. Therefore, our evaluation primarily leveraged human judgment. In our first study, we recruited a knowledgeable project stakeholder to systematically compare HGEN's performance against their own project documentation in three different projects. While in the second study, we conducted nine industry pilot studies using HGEN and elicited general feedback on its performance. In this section we describe the first study.

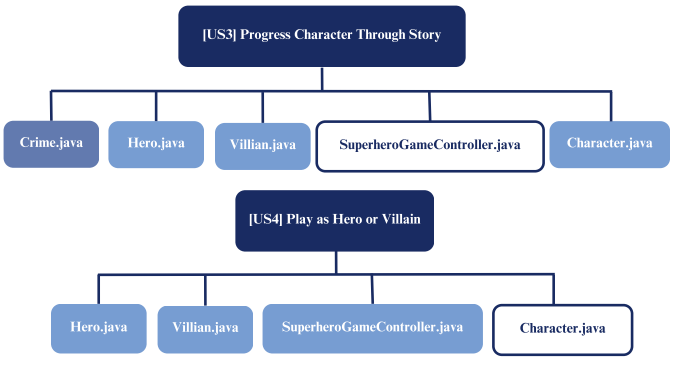

Fig. 8. Trace links selected for US3 and US4 after they were flagged containing overlapping content in Stage 5. Both user stories gain 1 additional trace link from the other (shown in white).

TABLE I
THE FIRST STUDY EVALUATED THE QUALITY OF MANUALLY CONSTRUCTED DOCUMENTATION, HGEN, AND A BASELINE APPROACH. TYPES AND NUMBERS OF ARTIFACTS ARE DEPICTED FOR EACH PROJECT. OPEN-SOURCE DATASETS ARE ANNOTATED WITH AN ASTERISK (*).

Dronology * [32]
Dronology is an Open-Source small Unmanned Aerial System (sUAS) written in Java. It provides a platform for controlling and coordinating multiple sUAS to support search-and-rescue, surveillance, and scientific data collection missions.

Functional Requirements		
Manual: 4	Base: 7	HGEN: 13
Design Definitions		
Manual: 12	Base: 4	HGEN: 25
Code Files		
27		

SAFA [33], [34]
SAFA is a software documentation management platform that leverages live traceability to build a knowledge graph and support change impact analysis. Our industry collaborators provided access to closed-source client-side source code.

Features		
Manual: 10	Base: 9	HGEN: 25
Functional Requirements		
Manual: 35	Base: 9	HGEN: 47
Code Files		
49		

Jack of Clubs *
Jack of Clubs is a re-creation of Ace of Spades, a voxel-based first-person shooter game. The creator gave us access to the manually created user stories and epics, and we are releasing them to the public as part of this paper.

Epics		
Manual: 3	Base: 4	HGEN: 5
User Stories		
Manual: 7	Base: 10	HGEN: 21
Code Files		
36		

A. Projects

The three projects are summarized in Table I. Our three projects represent diverse domains and cover both traditional and agile development processes. Each project included source code and at least two types of natural language artifacts, organized into layers, and connected by trace links. The Dronology is a UAV project that includes 423 Java files, 211 Design Definitions, and 99 Requirements respectively. SAFA is a software safety tool and includes 242 Vue source-code files, 262 functional requirements, and 101 features. Finally, Jack of Clubs is a first-person shooter game and includes 36 C++ files, 7 user stories, and 3 epics. Each project had an available Senior Developer with prior experience in developing documentation, to serve as the project expert.

B. Techniques under Comparison

Our study involved three different treatments including (i) the HGEN generated documentation, (ii) a baseline LLM approach, and (iii) the project documentation previously developed manually by project personnel. The HGEN documentation was generated for the three projects following the steps

TABLE II
EVALUATION GUIDELINES FOR ASSESSING DOCUMENTATION QUALITY.

Group	Metric	Metric Description
Language	Readability	How easily a user can understand the information provided.
	Appropriate-ness	Language appropriateness with respect to the artifact's technical level.
Content	Conciseness	How brief, yet clear the system is described.
	Importance	Considers the importance of the information provided.
Effective-ness	Usefulness	Considers how useful this documentation is to the project expert.
	Helpfulness	Consider how helpful the documentation is for understanding its children.

outlined in Section II of this paper, while the manually constructed project documentation was provided by the original project stakeholders as referenced in Table I.

We also created a baseline LLM approach for comparison purposes. We started with the identical set of summarized code as the HGEN processs and used the same LLM (Claude 2.0) to generate comprehensive documentation for each artifact type used in HGEN. As with HGEN, we repeat the process for each layer, and connect artifacts across layers by generating embeddings (using Sentence-BERT) for both the lower and higher-level artifacts. Trace links are established between artifact across the two layers if the normalized cosine similarity score is greater than 0.7. The major difference between the BASELINE and HGEN is the use of clustering techniques in HGEN as well as the later refinement steps.

Due to the size of Dronology and SAFA, we extracted a subset of source-code files and their generated documentation so that the project experts could conduct a more in-depth analysis of each artifact. In each case, the project expert identified the most critical source files, and the study included those files and their linked documentation.

IV. QUANTITATIVE COMPARISON OF THE DOCUMENTATION QUALITY

To evaluate the quality of the generated documentation we addressed the following research questions.

- **RQ1. Artifact Quality: How does the quality of individual machine-generated artifacts compare to that of expert-made artifacts?** We addressed this RQ by asking our project experts to evaluate the language, content, and effectiveness of each artifact as proposed by Hu et al., [31]. *Language* included readability and appropriateness, *content* included conciseness and importance, and *effectiveness* included usefulness and helpfulness. The definitions provided to our human assessors are summarized in Table and provided in our supplemental material.
- **RQ2. Coverage: To what extent are concepts that appear in the original documentation** *covered by* **the generated documentation?** To answer this RQ we measured concept coverage [35], [36] and evaluated the extent to which concepts appearing in the original documentation appeared in the generated documentation. Examples of concepts in HERO might be the ability to select different characters or purchase specific items at the shop.
- **RQ3. Relationships: How effectively does the machine-generated documentation build appropriate parent-child relationships inherent in multi-layer data structures?** Relationships are represented by trace links, and we therefore evaluated them using standard traceability metrics of recall and precision for the generated artifact tree [6]

A. RQ1. Artifact Quality

Project experts comparatively evaluated artifacts in their own project for (a) the manually constructed documentation, (b) HGEN, and (c) the baseline approach using a qualitative rubric associated with each quality in Table IV. Evaluations were performed against a Likert scale ranging from 1-5, where 1 signified low quality and 5 represented high quality. They were allowed to move freely between the different types of artifacts during this process.

Analysis of results: Due to non-normal score distributions, we employed the Mann–Whitney U test, which is non-parametric, to test if there was a notable difference in score distributions between each of the three treatments. Further, to account for multiple comparisons in our study—18 tests across six metrics and three documentation groups (Human-made, *Baseline*, *HGEN*)—we use the Holm-Šidák method to adjust our p-values to control for error rates, ensuring a reliable statistical analysis when comparing documentation quality across groups. Table III reports the mean scores for the six quality attributes assigned by the project experts across all three projects, as well as the adjusted p-values. It highlights instances where the null hypothesis is rejected ($p < 0.05$), suggesting that scores from one distribution tend to have higher scores than the other.

Discussion of results: The comparison between the human constructed documentation and the baseline approach returned comparable scores on all metrics except *Readability*, which was higher for the baseline approach. The same comparison between human and HGEN documentation showed that HGEN returned higher quality scores across four of the six metrics: *Readability*, *Appropriateness*, *Usefulness*, and *Helpfulness*. On the other hand in a direct comparison of HGEN versus the Baseline method, the only significant difference observed was for *Usefulness*, with HGEN's higher score indicating that project experts thought its documentation was more useful than the baselines. These results confirm the findings of previous studies that LLMs are able to produce software documentation of comparable quality to humans [37]. Notably, the main difference between HGEN and baseline is in the way HGEN constructs the hierarchy and not in the way it generates individual documents.

B. RQ2: Coverage

We asked each expert to evaluate concept coverage by identifying concepts in the manual documentation and checking whether they were adequately reflected in the generated

TABLE III
RESULTS OF MANN–WHITNEY U TESTS. CASES IN WHICH THE NULL HYPOTHESIS IS REJECTED (I.E., WHERE ($p < 0.05$)) ARE HIGHLIGHTED AND DEPICT CASES WHERE ONE TREATMENT OUTPERFORMED THE OTHER WITH RESPECT TO THE QUALITY ATTRIBUTE.

Human vs. Baseline			
Metric	**Human Mean**	**Baseline Mean**	**Corrected P-value**
Readability	3.90	**4.44**	**0.002445**
Appropriateness	3.70	4.00	0.214476
Conciseness	4.30	4.44	0.596927
Importance	4.10	4.35	0.191984
Usefulness	3.51	3.67	0.794294
Helpfulness	3.82	3.88	0.990378
Human vs. HGEN			
Metric	**Human Mean**	**HGEN Mean**	**Corrected P-value**
Readability	3.90	4.38	**0.000104**
Appropriateness	3.70	4.16	**0.001493**
Conciseness	4.30	4.32	0.618787
Importance	4.10	4.33	0.095525
Usefulness	3.51	4.14	**7.09e-06**
Helpfulness	3.82	4.21	**0.002445**
Baseline vs. HGEN			
Metric	**Baseline Mean**	**HGEN Mean**	**Corrected P-value**
Readability	4.44	4.38	0.990378
Appropriateness	4.00	4.16	0.462604
Conciseness	4.44	4.32	0.990378
Importance	4.35	4.33	0.990378
Usefulness	3.67	4.14	**0.020646**
Helpfulness	3.88	4.21	0.172422

TABLE IV
PERCENTAGE OF CONCEPTS FROM ORIGINAL DOCUMENTATION CAPTURED PER HGEN VERSION, AS DETERMINED BY THE PROJECT EXPERTS (E1-E3).

Project	**ID**	**Baseline**		**HGEN**	
		% Covered	**Covered by**	**% Covered**	**Covered by**
Dronology	E1	6.3%	9.1%	87.5%	28.9%
SAFA	E2	37.8%	38.9%	84.4%	43.1%
JOC	E3	50.0%	35.7%	100%	38.5%

documentation. We then computed the proportion of concepts from the original documentation that were addressed in each of the generated documentations.

Analysis of results: Results are reported in Table IV in the column labeled (% Covered). We also report the percentage of artifacts in the generated documentation that included these concepts (Covered by). The "Covered by" percentages for both the Baseline and HGEN documentation suggest that many artifacts focus on concepts that were not highlighted in the manual version.

Discussion of results: HGEN demonstrates a notable increase in concept coverage compared to the baseline approach across all three projects, capturing twice as many concepts in both JOC and SAFA, and an impressive 80% increase in the case of Dronology. Additionally, a larger portion of the HGEN-generated documentation, as indicated by the "Covered by" metric, centers on core concepts from the manual documentation, particularly in the case of Dronology. Given that the experts did not identify duplicate artifacts, it appears that both the Baseline and HGEN uncovered project aspects not emphasized in the manual documentation. Matched with the increase in 'helpfulness' returned by experiments for RQ1, it appears that the additional information could be helpful to project stakeholders.

TABLE V
TRACEABILITY ACCURACY METRICS FOR GENERATIVE APPROACHES

Project	**Approach**	**mAP**	**Precision**	**Recall**	**# Orphans**
Dronology	Baseline	84.5%	47.2%	89.5%	17
	HGEN	94.0%	56.3%	93.4%	0
SAFA	Baseline	91.9%	49.2%	100%	28
	HGEN	94.5%	54.3%	98.4%	9
JOC	Baseline	95.5%	67.3%	74.5%	11
	HGEN	96.7%	81.4%	80.2%	1

C. RQ3: Relationships

To evaluate the quality of relationships within each generated hierarchy, we developed a basic tracing tool which visualized the generated documentation tree and allowed project experts to approve or decline existing links, adding new links if needed. Each expert performed this task twice -- once for HGEN and once for the Baseline approach, resulting in their version of a ground truth solution for each generative technique. We then evaluated the generated trees for HGEN and Baseline against the modified ground truth version for each one and computed mean average precision (mAP), precision, and recall for each project using standard formulas [6]. To compute mAP, which assesses the extent to which correct links appear at the top of a ranked list, we ordered the links according to their original cosine similarity scores. In addition, we assessed the number of orphan artifacts generated by each approach, as this aspect had emerged as a key distinguishing factor through discussions with the three experts. Table IV-C presents these results.

Discussion of results: The relatively high mAP scores (80% to 95.5%) and recall scores (47.2% - 81.4%) indicate that Sentence-BERT was able to capture a range of semantic similarities between artifacts. This is likely attributable to the LLM's use of lower-level artifacts for generating higher-level ones, thereby creating a shared vocabulary. However, we also observed a significantly higher number of orphans in the Baseline approach versus HGEN, which could have lowered recall whilst increasing precision. HGEN's lower orphan count suggests that it's enhanced clustering techniques enabled it to capture the concepts in the low-level artifacts at higher levels of abstraction, identifying concepts that might otherwise have been overlooked.

D. Qualitative Feedback

We also asked each expert a number of open-ended questions including having them describe the most and least valuable characteristics of the documentation. A full list of these questions can be found in the paper's supplemental section. We briefly summarize their feedback.

The experts acknowledged the quality of the baseline's individual artifacts, which, in the case of E3, was identified as more readable than the project expert's own documentation. However, both E1 and E2 identified that the baseline version

lacked comprehensiveness, clarity, and accurate prioritization of information compared to the manual documentation. Furthermore, its sparse generations resulted in the creation of "redundant" parents highlighting the *Baseline's* tendency to establish 1-1 relationships between its initial and final layer generations.

All experts preferred the HGEN version over the baseline method. E2 and E3 stated that HGEN tended to produce more detailed and comprehensive information that provided "helpful" relationships and dependencies, with links generally "making sense."

Despite their preference for HGEN, the experts noted some shortcomings. E2 mentioned that HGEN missed some obvious links between clusters and did not reflect the same organizational structure for conceptualizing code as they had used. This difference in structure lead both E1 and E2 to favor their own documentation over HGEN's, although E2 acknowledged HGEN's was "more useful for teaching someone about my system," suggesting that preference depended on the documentation's intended use. Meanwhile, E3 appreciated HGEN's detailed information and preferred it over their own.

V. Industrial Pilot Studies

We now present the results of our industrial pilot studies which focused purely on the HGEN solution.

A. Study Method

The nine pilots were conducted in seven different companies on eight unique projects as depicted in Table VI. Seven of them used data provided by the company, and two (marked with an asterisk) used open-source project data [38]. For each project we applied HGEN to the source code to generate documentation.

To assess the effectiveness of the documentation, we conducted interviews with nine industrial partners. During these interviews, partners were prompted to share their candid thoughts on the documentation's quality, usefulness, and suggestions for potential improvements. After obtaining permission, we recorded the sessions and utilized an AI tool to automatically transcribe the recordings. Two of the paper's authors independently extracted all relevant quotes, used an inductive approach to encode each quote, and then worked together to discuss the code and to sort them into categories. We did not assess inter-rater agreement as this activity was performed collaboratively.

B. Analysis of Feedback

Our analysis identified three clear themes and six subcategories associated with documentation quality, prospective use cases, and recommendations for improvements.

1) Documentation Quality: Two sub-themes emerged for documentation quality. The first addressed *information accuracy and coverage* and focused on whether the generated documentation conveyed crucial information about the source code without errors. Feedback was generally positive. Participant P9 expressed strong satisfaction, stating, "everything I can read corresponds exactly to the reality," while P1 remarked, "I feel like they are a good representation of what we are doing." However, one participant, P2, noted a discrepancy where an external tool was mentioned despite not being implemented in the code. On the other hand they pointed out that the tool was referenced in the code as a prerequisite, which likely misled the LLM. P4 stated that they couldn't find any errors at all, and confirmed that it captured all essential information, stating that "I couldn't identify any aspect missing from it."

The second theme focused on *clarity and structure*, including readability and understandably of the documentation. P4 observed that the documentation was well written, and was "probably better than I could do." P8 specifically praised the hierarchical organization of the documentation, expressing appreciation for the fact that it provided "a summary at every level of depth...[and] every level of extraction."

2) Prospective Use cases: Participants also focused on how HGEN could benefit their respective companies, with three specific use cases emerging, all of which are highly pertinent to software maintenance.

Five participants (P2, P6, P7, P8, P11) highlighted HGEN's potential for *comprehending complex systems* lacking sufficient documentation, especially in scenarios where the original authors are unavailable. Of these, P7 underscored its use when "nobody knows anything about [the code]" stating that "you run it through your system, and then it's a lot more potentially clear, and people can understand what was going on." P11 felt that it would be particularly advantageous for "tackling some pretty legacy stuff." Participants P6 and P7 enthusiastic about HGEN's time-saving capabilities, with P6 stating that without documentation, understanding a system might take a month, whereas with the HGEN documentation "I have an idea maybe within days. So it's definitely a big help on that one."

Four participants (P1, P2, P3, P6) highlighted the value of HGEN for expediting the onboarding of new developers. P2 stated that they would no longer need "to spend tons of time giving the engineer an overview of the code base", and P3 echoed this sentiment, stating that it was "a whole lot better than me setting an intern down with some piece of code that I got and telling him. 'Hey do your best buddy. We'll talk to you in four months'".

Finally, two participants (P7, P11) discussed the use of HGEN for regulatory compliance. P7 observed that generating documentation would simplify the process, stating, "instead of going to the code and trying to figure out what to provide to them" (i.e., regulators), "[HGEN] would be somewhat easier." P11 said that HGEN would have assisted them in a previous government project, in which they were required to maintain an "enormous" and comprehensive list of requirements, which was challenging to maintain. They believed that utilizing HGEN might have made the task more manageable.

3) Potential Improvements: Two participants (P1, P3) recommended new features aimed at enhancing HGEN's utility. P1 noted that the generated documentation failed to include details from external libraries, suggesting that relevant parts of these libraries would provide context for the LLM. P3

TABLE VI
HGEN WAS USED IN NINE INDUSTRIAL PILOT PROJECTS FROM MULTIPLE DOMAINS WITH THE SOURCE CODE LAYER WRITTEN IN DIVERSE PROGRAMMING LANGUAGES.

ID	Category	Use Case			ID	Lang.	Input Files
		RE.	LG	OB			
C1	Enterprise	●	●		P1	C#	1,049
		●	●		P2	C++	181
C2	Automotive*				P3	C++ *	242 *
C3	SAAS		●		P4	TS / JS	264
C4	IT Services	●			P5	Java	197
C5	Aerospace		●		P6	C	345
C6	Education			●	P7	Java	643
C7	Automotive*				P8	*	*
C8	IT Services		●		P9	Go / TS	65

*Pilot conducted using Open-Source Systems
RE=Reverse Engineering, LG=Legacy Documentation, OB=Onboarding
JS= Java Script, TS = Type Script

proposed adapting HGEN for instances where high-level documentation already exists, suggesting that generated documentation could bridge the gap between this high-level overview and the code. Finally, P1 was intrigued by the potential for extending HGEN to provide incremental support for documentation alongside code development.

C. Discussion of Results

The feedback provided by project stakeholders highlighted several benefits and potential applications of HGEN as well as some places for improvement. Most importantly, it demonstrated that HGEN was capable of generating high quality documentation hierarchies for an extremely wide range of software projects. However, these results are based on a pilot study, and therefore feedback is based on stakeholders perspectives of HGEN's utility rather than on its adoption in practice. Nevertheless, this is an important first step in validating HGEN for deployment on industrial projects.

VI. Related Work

Our work is informed by the groundwork laid by prior research in the areas of automated document generation [18]. We discuss most closely related work in three specific areas.

First, there is a large body of work in automating code-level documentation and API specifications for various frameworks [16], [15], [39], [40], [41], [42], [43], [44], [45]. Our research focuses on generating hierarchical, multi-level documentation for higher-level system abstraction. We build on the emerging results showing that LLMs can generate a variety of software requirements and documentation. Dvivedi et al. showed that LLM-based models often surpass human documentation in inline, function, and file level code documentation [46]. Likewise, Bencheikh and Höglund's demonstrated that LLMs can generate software requirements [37], while Xie et al. use it to generate code specifications [47]. Our work aims to enhance the existing capabilities of LLMs to create diverse documentation types at multiple abstraction levels. In the private sector, companies like swimm.io have investigated documentation automation, but claim that full automation is infeasible due to the difficulty of integrating business logic, design decisions, and other external elements [48]. While this is important, our work demonstrates the benefits of automation as a component of the documentation process.

Research in source code understanding has predominantly focused on generating detailed, low-level explanations. Notable works include Srihara et al.'s file-level code summarizations using natural language generators [49], and methods by Robillard, Burden, Moreno, among others, for creating concise method and class summaries [14], [50], [51], [52]. Others have explored parameter-level comments [53] and context-enhanced summaries [54], [55], including method functionality, purpose, and usage [54]. HGEN, in contrast, targets higher-level, language-agnostic documentation, leveraging LLMs to summarize code across most major programming languages.

The pursuit of ubiquitous software traceability serves as a foundational inspiration, drawing from extensive prior research dedicated to the development and refinement of automatic trace link generation across various software domains [56], [57], [7], [8], [58]. Despite substantial progress in software traceability [59], [60], challenges in achieving high accuracy remain, especially in data-scarce areas [6], [61]. Recently, LLMs like GPT3, GPT4, and Claude have demonstrated potential in improving trace link accuracy in such scenarios [62], [20], [21], [63]. Our approach integrates advances from BERT-based models for trace link generation and LLMs for documentation generation, with a novel clustering strategy to increase trace accuracy.

VII. Threats to Validity

Our work includes several important threats to validity. With respect to construct validity, our first study focused on three projects only. However, we partially mitigated the threat to generalizability by selecting projects from diverse domains, which encompassed both open-source and closed-source code for different sized project. We then applied HGEN to industrial project data, and the feedback from project experts indicated that it was effective across all domains. However, our pilot studies were based on feedback elicited from an interactive demonstration, and while this provides valuable insights, an additional study is needed of its use over time in industrial projects. In a threat to external validity, the HGEN pipeline is quite complex, developed following much trial and error and its replication is complex. To mitigate this, we ensured repeatability in the hierarchy generation process by minimizing randomness and maintaining a closely-deterministic approach in our pipeline. Multiple runs were conducted to confirm consistent results. Further, we have prepared a fully functioning system that is accessible via a web application.[1] We also provide all study materials in the supplemental materials.

VIII. Conclusion

This paper proposes HGEN, an LLM-based approach to automatically generate hierarchies of requirements documentation from source code. HGEN builds upon the recent successes of LLMs by engineering a process that addresses some of the deficiencies identified with the unaided models.

Our evaluation, designed to target three critical aspects of the documentation, supports existing literature that LLM-generated documentation can match or exceed the quality of that written by humans. We also show that HGEN is able to capture meaningful relationships across varied artifact levels and can identify nearly all of the core concepts found in expert-produced documentation, showing a considerable enhancement over the baseline LLM. These results indicate that HGEN could substantially reduce the time and effort needed for comprehensive documentation creation, thereby aiding in software maintenance tasks. Moreover, HGEN presents potential for automating additional aspects of requirements engineering, paving the way forward towards ubiquitous documentation and traceability.

IX. Acknowledgements

The research described in this paper was partially supported by the USA National Science Foundation (NSF) under grant numbers 1909007 and 1901059.

References

[1] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: A survey," in *Proceedings of the 2002 ACM Symposium on Document Engineering*, ser. DocEng '02. New York, NY, USA: Association for Computing Machinery, 2002, p. 26–33. [Online]. Available: https://doi.org/10.1145/585058.585065

[2] T. Lethbridge, J. Singer, and A. Forward, "How software engineers use documentation: the state of the practice," *IEEE Software*, vol. 20, no. 6, pp. 35–39, 2003.

[3] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?" in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 255–265.

[4] O. Gotel and A. Finkelstein, "Contribution structures (requirements artifacts)," in *Second IEEE International Symposium on Requirements Engineering, March 27 - 29, 1995, York, England, UK*. IEEE Computer Society, 1995, pp. 100–107. [Online]. Available: https://doi.org/10.1109/ISRE.1995.512550

[5] S. Maro, J. Steghöfer, and M. Staron, "Software traceability in the automotive domain: Challenges and solutions," in *Software Engineering and Software Management, SE/SWM 2019, Stuttgart, Germany, February 18-22, 2019*, ser. LNI, S. Becker, I. Bogicevic, G. Herzwurm, and S. Wagner, Eds., vol. P-292. GI, 2019, pp. 61–62. [Online]. Available: https://doi.org/10.18420/se2019-14

[6] J. Cleland-Huang, O. Gotel, J. H. Hayes, P. Mäder, and A. Zisman, "Software traceability: trends and future directions," in *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, J. D. Herbsleb and M. B. Dwyer, Eds. ACM, 2014, pp. 55–69. [Online]. Available: https://doi.org/10.1145/2593882.2593891

[7] A. D. Lucia, A. Marcus, R. Oliveto, and D. Poshyvanyk, "Information retrieval methods for automated traceability recovery," in *Software and Systems Traceability*. Springer, 2012, pp. 71–98.

[8] M. Rath, J. Rendall, J. L. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: automatically augmenting incomplete trace links," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 834–845.

[9] B. Fluri, M. Wursch, and H. C. Gall, "Do code and comments co-evolve? on the relation between source code and comment changes," in *14th Working Conference on Reverse Engineering (WCRE 2007)*, 2007, pp. 70–79.

[10] P. Mäder, P. L. Jones, Y. Zhang, and J. Cleland-Huang, "Strategic traceability for safety-critical projects," *IEEE Softw.*, vol. 30, no. 3, pp. 58–66, 2013. [Online]. Available: https://doi.org/10.1109/MS.2013.60

[11] P. Rempel, P. Mäder, T. Kuschke, and J. Cleland-Huang, "Mind the gap: assessing the conformance of software traceability to relevant guidelines," in *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, P. Jalote, L. C. Briand, and A. van der Hoek, Eds. ACM, 2014, pp. 943–954. [Online]. Available: https://doi.org/10.1145/2568225.2568290

[12] N. Paternoster, C. Giardino, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, vol. 56, no. 10, pp. 1200–1218, Oct. 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584914000950

[13] C. Giardino, N. Paternoster, M. Unterkalmsteiner, T. Gorschek, and P. Abrahamsson, "Software Development in Startup Companies: The Greenfield Startup Model," *IEEE Transactions on Software Engineering*, vol. 42, no. 6, pp. 585–604, Jun. 2016, conference Name: IEEE Transactions on Software Engineering. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/7360225

[14] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: Understanding the decisions of open source contributors," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 127–136. [Online]. Available: https://doi.org/10.1145/1882291.1882312

[15] "Home." [Online]. Available: https://www.openapis.org/

[16] D. van Heesch, "Doxygen, a tool for generating documentation from annotated source code (ver.1.9.7)," May 2023. [Online]. Available: https://www.doxygen.nl/

[17] E. Aghajani, C. Nagy, M. Linares-Vásquez, L. Moreno, G. Bavota, M. Lanza, and D. C. Shepherd, "Software documentation: the practitioners' perspective," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 590–601. [Online]. Available: https://doi.org/10.1145/3377811.3380405

[18] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, and M. Linares-Vásquez, "On-demand developer documentation," in *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 2017, p. 479–483. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8094446/

[19] R. J. Wieringa, *Design Science Methodology for Information Systems and Software Engineering*. Springer, 2014. [Online]. Available: https://doi.org/10.1007/978-3-662-43839-8

[20] OpenAI, "GPT-4 Technical Report," Mar. 2023, arXiv:2303.08774 [cs]. [Online]. Available: http://arxiv.org/abs/2303.08774

[21] A. Askell, Y. Bai, A. Chen, D. Drain, D. Ganguli, T. Henighan, A. Jones, N. Joseph, B. Mann, N. DasSarma, N. Elhage, Z. Hatfield-Dodds, D. Hernandez, J. Kernion, K. Ndousse, C. Olsson, D. Amodei, T. Brown, J. Clark, S. McCandlish, C. Olah, and J. Kaplan, "A general language assistant as a laboratory for alignment," 2021.

[22] "2.3. Clustering." [Online]. Available: https://scikit-learn/stable/modules/clustering.html

[23] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "OPTICS: ordering points to identify the clustering structure," *SIGMOD Rec.*, vol. 28, no. 2, pp. 49–60, Jun. 1999. [Online]. Available: https://dl.acm.org/doi/10.1145/304181.304187

[24] S. X. Yu and J. Shi, "Multiclass Spectral Clustering."

[25] K. Sasirekha and P. Baby, "Agglomerative Hierarchical Clustering Algorithm- A Review," vol. 3, no. 3, 2013.

[26] D. Dueck, "Affinity Propagation: Clustering Data by Passing Messages," Thesis, Sep. 2009, accepted: 2009-09-24T14:35:03Z. [Online]. Available: https://tspace.library.utoronto.ca/handle/1807/17755

[27] D. Arthur and S. Vassilvitskii, "k-means++: the advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, ser. SODA '07. USA: Society for Industrial and Applied Mathematics, Jan. 2007, pp. 1027–1035.

[28] F. Bachmann, L. Bass, P. Clements, D. Garlan, J. Ivers, R. Little, R. Nord, and J. Stafford, "Documenting software architectures: Organization of documentation package," *Software Engineering Institute*, 2001.

[29] A. Ferrari, S. Gnesi, and G. Tolomei, "Using clustering to improve the structure of natural language requirements documents," in *Requirements Engineering: Foundation for Software Quality: 19th International Working Conference, REFSQ 2013, Essen, Germany, April 8-11, 2013. Proceedings 19*. Springer, 2013, pp. 34–49.

[30] S. Yeganefard and M. Butler, "Control systems: Phenomena and structuring functional requirement documents," in *2012 IEEE 17th International Conference on Engineering of Complex Computer Systems*, 2012, pp. 39–48.

[31] X. Hu, Q. Chen, H. Wang, X. Xia, D. Lo, and T. Zimmermann, "Correlating automated and human evaluation of code documentation generation quality," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 4, p. 1–28, Oct. 2022.

[32] J. Cleland-Huang, M. Vierhauser, and S. Bayley, "Dronology: An incubator for cyber-physical system research," *CoRR*, vol. abs/1804.02423, 2018. [Online]. Available: http://arxiv.org/abs/1804.02423

[33] A. D. Rodriguez, T. Newman, K. R. Dearstyne, and J. Cleland-Huang, "SAFA: A tool for supporting safety analysis in evolving software systems," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*. ACM, 2022, pp. 165:1–165:4. [Online]. Available: https://doi.org/10.1145/3551349.3559535

[34] J. Cleland-Huang, A. Agrawal, M. Vierhauser, and C. Mayr-Dorn, "Visualizing change in agile safety-critical systems," *IEEE Softw.*, vol. 38, no. 3, pp. 43–51, 2021. [Online]. Available: https://doi.org/10.1109/MS.2020.3000104

[35] H. Lawson and J. N. Martin, "On the use of concepts and principles for improving systems engineering practice," in *Proceedings of the 18th Annual International Council on Systems Engineering (INCOSE) International Symposium*, Utrecht, The Netherlands, June 5-19 2008.

[36] Systems Engineering Body of Knowledge (SEBoK). (2024) Concept. [Online]. Available: https://sebokwiki.org/wiki/Concept_(glossary)

[37] L. Bencheikh and N. Höglund, "Exploring the efficacy of chatgpt in generating requirements: An experimental study," Aug. 2023. [Online]. Available: https://gupea.ub.gu.se/handle/2077/77957

[38] Autowarefoundation, "Autowarefoundation/autoware.universe." [Online]. Available: https://github.com/autowarefoundation/autoware.universe

[39] "Welcome — Sphinx documentation." [Online]. Available: https://www.sphinx-doc.org/en/master/

[40] "API Documentation & Design Tools for Teams | Swagger." [Online]. Available: https://swagger.io/

[41] D. Kramer, "Api documentation from source code comments: A case study of javadoc," in *Proceedings of the 17th Annual International Conference on Computer Documentation*, ser. SIGDOC '99. New York, NY, USA: Association for Computing Machinery, 1999, p. 147–153. [Online]. Available: https://doi.org/10.1145/318372.318577

[42] "Django." [Online]. Available: https://www.djangoproject.com/

[43] "Spring Boot." [Online]. Available: https://spring.io/projects/spring-boot

[44] "FastAPI." [Online]. Available: https://fastapi.tiangolo.com/

[45] "Node.js." [Online]. Available: https://nodejs.org/en

[46] S. S. Dvivedi, V. Vijay, S. L. R. Pujari, S. Lodh, and D. Kumar, "A comparative analysis of large language models for code documentation generation," 2023.

[47] D. Xie, B. Yoo, N. Jiang, M. Kim, L. Tan, X. Zhang, and J. S. Lee, "Impact of large language models on generating software specifications," no. arXiv:2306.03324, Oct. 2023, arXiv:2306.03324 [cs]. [Online]. Available: http://arxiv.org/abs/2306.03324

[48] O. Rosenbaum, "Will internal documentation be replaced? An AI discussion," Jul. 2023. [Online]. Available: https://swimm.io/blog/will-internal-documentation-be-replaced-an-ai-discussion

[49] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 43–52. [Online]. Available: https://doi.org/10.1145/1858996.1859006

[50] M. Nassif, A. Hernandez, A. Sridharan, and M. P. Robillard, "Generating unit tests for documentation," vol. 48, no. 9, 2022, pp. 3268–3279. [Online]. Available: https://doi.org/10.1109/TSE.2021.3087087

[51] H. Burden and R. Heldal, "Natural language generation from class diagrams," in *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation*, ser. MoDeVVa. New York, NY, USA: Association for Computing Machinery, 2011. [Online]. Available: https://doi.org/10.1145/2095654.2095665

[52] L. Moreno, J. Aponte, G. Sridhara, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 23–32.

[53] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Generating parameter comments and integrating with method summaries," in *2011 IEEE 19th International Conference on Program Comprehension*, 2011, pp. 71–80.

[54] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," *IEEE Transactions on Software Engineering*, vol. 42, no. 2, pp. 103–119, 2016.

[55] ——, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: Association for Computing Machinery, Jun. 2014, p. 279–290. [Online]. Available: https://dl.acm.org/doi/10.1145/2597008.2597149

[56] *Software and Systems Traceability*. London: Springer London, 2012. [Online]. Available: http://link.springer.com/10.1007/978-1-4471-2239-5

[57] G. Antoniol, J. Cleland-Huang, J. H. Hayes, and M. Vierhauser, "Grand challenges of traceability: The next ten years," *arXiv:1710.03129 [cs]*, Oct. 2017, arXiv: 1710.03129. [Online]. Available: http://arxiv.org/abs/1710.03129

[58] M. Rahimi and J. Cleland-Huang, "Evolving software trace links between requirements and source code," in *Proceedings of the 10th International Workshop on Software and Systems Traceability, SST@ICSE 2019, Montreal, QC, Canada, May 27, 2019*, J. Steghöfer and N. Niu, Eds. IEEE / ACM, 2019, p. 12. [Online]. Available: https://doi.org/10.1109/SST.2019.00012

[59] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained bert models," *arXiv:2102.04411 [cs]*, Feb 2021, arXiv: 2102.04411. [Online]. Available: http://arxiv.org/abs/2102.04411

[60] J. Lin, A. Poudel, W. Yu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Enhancing automated software traceability by transfer learning from open-world data," no. arXiv:2207.01084, Jul 2022, arXiv:2207.01084 [cs]. [Online]. Available: http://arxiv.org/abs/2207.01084

[61] J. Guo, J. Cheng, and J. Cleland-Huang, "Semantically enhanced software traceability using deep learning techniques," in *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. Buenos Aires: IEEE, May 2017, p. 3–14. [Online]. Available: http://ieeexplore.ieee.org/document/7985645/

[62] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Understanding the Challenges of Deploying Live-Traceability Solutions," Jun. 2023, arXiv:2306.10972 [cs]. [Online]. Available: http://arxiv.org/abs/2306.10972

[63] A. Rodriguez, K. Dearstyne, and J. Cleland-Huang, "Prompts matter: Insights and strategies for prompt engineering in automated software traceability," in *Proceedings of the 11th International Workshop on Software and Systems Traceability, SST@RE 2023, Hanover, Germany*, J. Steghöfer and N. Niu, Eds. IEEE, 2023.