BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

GENERATIVE AI FOR AUTOMATED SOFTWARE PROJECT PLANNING

[1]Vighnesh Jaywant Alhat, [2]Prof. Rupali Wagh

[1]Student,[2]Professors
[1,2]Department of Artificial Intelligence and Data Science
[1,2]Ajeenkya DY Patil School of Engineering, Pune, Maharashtra, India
[1]vighanesh.alhat@dypic.in, [2]rupaliwagh@dypic.in

Abstract

Software project planning involves defining the scope, requirements, stakeholders' roles, task dependencies, duration of each task and potential risks. For the most part, these activities are carried out manually and highly depend on the experience of project managers. Fuzzy requirements, incorrect estimations and dependency issues can make a project fall behind schedule and decrease the quality of delivery. Generative AI, in particular, Large Language Models (LLMs), can solve this problem by analyzing a project's textual description, extracting requirements, proposing the architecture, decomposing the work into tasks and producing the planning documentation. However, LLMs can produce false statements, wrong estimates and contradictions and thus the need for manual oversight is still necessary. The paper describes a framework that involves the generation of planning data by LLMs and the validation of the result. A validation layer evaluates the completeness of requirement coverage, rationality of dependencies and feasibility of schedule before the output is shown to the project manager. Ten selected papers are discussed, the comparison of the existing approaches, research gaps and several metrics are presented.

Keyword : Generative AI, Large Language Models, Software Project Planning, Task Decomposition, Effort Estimation, Risk Analysis, Project Scheduling.

I. INTRODUCTION

Planning is involved in all software development projects – what it should achieve, how large it is, what technologies are used, task breakdown, estimated length, dependencies between tasks and possible risks. This process is efficient, yet it takes time to prepare an adequate plan and changing one requirement necessitates updating task lists, scheduling, estimates, and risk register. Also, managing planning artifacts separately makes them fall apart rather fast. An LLM can comprehend natural language input and provide a structured planning output based on a description of a project. The previous research studied the capabilities and limitations of LLMs for executing development tasks [6], [9], [10]. However, an apparently sensible plan may not work at all. An LLM may not notice dependency, suggest an unrealistically long or short length, wrong technology or even unsupported information. Thus, some restrictions and validation are required for automated planning.

Problem and Motivation. The process of making a comprehensive plan from an informal specification is a set of interrelated activities (requirement, constraint, technical choice, task, estimation, dependency, scheduling and risk) which take a lot of time and are quite inconsistent when performed manually. The idea

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

is not to automate the project managers but to give them an assistant who will generate the first draft which will then be checked and validated by the project manager with all requirements traced back to tasks, dependencies and estimations. In this paper, we (i) review recent work on GenAI and LLMs in the context of software engineering, (ii) propose our structured approach to generating plans from natural language specifications, (iii) provide a rule-based validation, (iv) define evaluation criteria, and (v) discuss limitations and future work.

Objectives. The objectives of the paper are (i) to perform literature review of GenAI and LLMs in software engineering, (ii) to formulate a plan generation framework that uses natural language inputs, (iii) to integrate rule-based validation, (iv) to formulate evaluation criteria, and (v) to define limitations.

Research Questions. RQ1: In what way can large language models help in performing the basic activities of planning in software projects? RQ2: How can deterministic rules guarantee consistency in the planning outputs of LLMs? RQ3: Which metrics will determine completeness, feasibility, and efficiency of the planning activity with the use of AI? RQ4: What are the limitations that need to be overcome before implementation of the approach?

Scope. The scope of the research concerns the planning phase of the early development process. No claim is made regarding the increase of accuracy in a deployed system; rather, the paper states the concept and provides literature review and evaluation framework.

II. BACKGROUND

Planning of software project includes identification of work that needs to be done to develop and deliver the product. The outcome of it is described below: scope statement, requirements specification, work breakdown structure, effort estimation, schedule, resource planning, risk register and technical architecture. All of them are interconnected with each other, for example, change in technology affects effort estimation and vice versa.

Generative AI and LLMs. Generative AI generates new data depending on the learned patterns and given input data. In case of large language models trained on large volumes of textual data, it is possible to create document interpretation and summaries. In this suggested solution, LLM performs language-related tasks like requirements extraction, task suggestions, architecture suggestions and risks brainstorming. Its result depends on input data, model and prompts.

Task Decomposition and Scheduling. Task decomposition refers to breaking up a high-level goal into several tasks which are characterized by identification, description, requirements, effort estimation, role, dependencies, and criteria for acceptance. The next step is to arrange these tasks in time – to schedule them. If task i needs to precede task j,

$$S_j \geq S_i + D_i \quad (1)$$

Here, S_i and S_j denote the start times of the respective tasks, and D_i represents the duration of task i. Real-life scheduling may involve constraints on resources, calendars, and uncertain events.

Deterministic validation involves applying certain explicit rules in order to come up with the same output based on the same input. Such validation includes verification that there is a task for each requirement, task

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

dependencies, self-dependency of a task and schedule conflicts in connection with deadlines. These checks cannot evaluate ambiguous business requirements and architecture suitability.

III. LITERATURE REVIEW

This section provides a literature review of ten papers from broad surveys and research agendas to tools to collaboration and planning papers.

Zhang et al. provide a review of LLM applications in software engineering [1], which puts into context the problem of planning in AI-supported software engineering, as it involves reasoning about project-related constraints and objectives, unlike code generation. He, Treude and Lo give a review of LLM-based multi-agent systems [2], with agents being able to specialize in requirements, architecture, estimation and risks, yet coordination is no guarantee of agreement. Kharrufa et al. study the use of LLMs in team projects in software engineering [3], highlighting the fact that any generated plan should be in line with the team's workflow and with the assignment of responsibilities. Song, Agarwal and Wen analyze collaborative open-source development with the help of GitHub Copilot [4], and Pandey et al. analyze the efficiency of Copilot in practice [5]; both suggest measuring efficiency together with review effort, though coding results do not prove that generated schedules are accurate. Hou et al. present a systematic review of LLMs for software engineering [6], and Sauvola et al. discuss how generative AI may reshape development workflows [7].

For planning and research agendas, Bairi et al. present CodePlan, a combination of LLMs with planning on the repository scale [8]. This is to show how important component relations are in the same way that task relations are in a project. The research agendas and open questions from Nguyen-Duc et al. [9] and Fan et al. [10] put emphasis on evaluation, trust, context adaptation, and human oversight.

Synthesis. There are many studies regarding the use of LLMs in software engineering, although studies involving generation of code or collaboration cannot prove alone the ability of an LLM to generate a complete and viable plan. A plan involves the representation of all requirements, task interdependencies, dependencies among them, estimation and evaluation of estimates, and schedule that conforms to all constraints. Hence, the proposed framework separates the LLM generation process from the rule-based validation and human validation processes.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

TABLE I

SELECTED LITERATURE REVIEW

Year	Authors	Research Focus	Contribution	Relevance to This Study
2026	Zhang et al.	LLMs for software engineering	Literature review	Positions planning within AI-enabled SE activities.
2025	He, Treude, Lo	LLM-powered multi-agent systems	Literature review and future outlook	Advocates for specialist planning agents.
2024	Kharrufa et al.	LLMs in teamwork	Roles, effect, tool creation	Emphasizes collaboration and human supervision.
2024	Song et al.	GenAI in open-source collaboration	Experience of Copilot	Provides background for AI in collaborative processes.
2024	Pandey et al.	Copilot in real projects	Efficiency and challenges	Encourages assessment of efficiency and reviewing effort.
2024	Hou et al.	LLMs for software engineering	Literature review	Comprehensive and recurring issues.
2024	Sauvola et al.	Future of software development	GenAI and practices	Workflow inclusion and professions.
2024	Bairi et al.	CodePlan	Repository-oriented planning	Framework for planning within related components.
2023	Nguyen-Duc et al.	GenAI for SE	Research agenda	Ground for posing research questions.
2023	Fan et al.	LLMs for SE	Survey and open problems	Pioneering study on capabilities and limitations.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

IV. COMPARISON OF RESEARCH DIRECTIONS AND POSSIBLE IMPACT ON AUTOMATED PLANNING

Table II provides qualitative comparison of research directions and potential impact they would have on automated planning. It does not contain ranking of effectiveness.

TABLE II

QUALITATIVE COMPARISON OF RESEARCH DIRECTIONS

Directions	Focus	Planning Application	Consideration for Framework
LLM surveys	Abilities in software engineering	Identify tasks for automation	Context, not accuracy in planning.
Multi-agent systems	Specialized agents divided into separate tasks	Split requirements, architecture, estimation, risk	Divide the work among the agents and check.
AI development tools	Programming and everyday tasks	Define tasks and technical solutions	Evaluate planning independently from coding benefits.
Repository-based planning	Mutual dependencies in related components	Represent mutual dependencies in tasks	Make sure to preserve task-component dependencies.
Human-AI collaboration	Human roles and practices	Management reviews and modifies plans	Approval process stays within humans.
Deterministic validation	Explicit and repeatable validation	Check dependencies, fields, scheduling rules	Combine validation with human evaluation.

The proposed framework is an integration level that is not meant to replace project management tools and estimation techniques. Its distinctive feature lies in the fact that automatically created plan becomes a candidate which needs to be validated: the task list can be written by an LLM while the dependencies references and testing of the timeline can be performed by a program. Whether this increases the quality of plans and eliminates unnecessary inconsistencies requires further verification.

V. PROPOSED METHODOLOGY

The method converts the description of a project into planning artifacts through generation, validation, and review.

Input collection. The system takes the natural language description, along with the goals, deliverables, functional and non-functional requirements, constraints on budget and deadline, skills of the team members, and technologies and methods involved. Absence of input should be followed by a question or an assumption but never by implicit agreement about any default value.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Requirement processing. The LLM provides a set of structured requirements with identifiers, description, category, priority, source, and acceptance criteria. All of the outputs are trackable back to the description provided initially, thus assisting the reviewer to understand the rationale for generation of this requirement.

Technical planning. Based on the inputs and constraints, the LLM gives recommendations about the technology stack, architecture style, main modules, integration methods, data storage, and test approach. All of the assumptions made are explicitly stated. Recommendations are separated from requirements and are never considered the best.

Task decomposition. The requirements and deliverables are decomposed to work items having unique identification number, description, requirement, acceptance criteria, effort estimate, level of skills required, predecessors and risk. This decomposition is hierarchical and follows the hierarchy of epics, activities, tasks, and subtasks based on the size of the project.

Effort estimation. LLM estimation is an uncertain forecast which is calibrated Marked with a reason for validation and adjustable by a reviewer. To evaluate the planning, we use the estimation error of task i given by

$$E_i = |A_i - P_i| / \max(A_i, \epsilon) \times 100 \quad (2)$$

where A_i is actual effort, P_i is estimated effort and ϵ is a small positive constant used to avoid division by zero. Should be interpreted with caution when actual effort is zero.

Dependencies and schedule. The system creates a dependency graph, detects invalid links and cycles, and generates a schedule taking into account constraint (1), deadlines, resources and calendars. With insufficient information, the schedule is called provisional.

Risk identification. The LLM identifies risks from scope, architecture, dependencies and constraints, for example, unrelated requirements, problems of integration, inadequate resources, security issues and schedule pressure. For each risk record, a description, likelihood, impact, mitigation, owner and status are stored; scores are not objective probabilities unless calibrated with respect to evidence.

Validation and approval. Before being finalized, a validation layer ensures that required fields are specified, requirements are linked to tasks, dependency IDs are provided, the graph does not have cycles, constraints on schedule, resources and deadlines are met, and any conflicting or lack of information is identified. After that, the project manager accepts, modifies or rejects the recommendations. Human approval is important since rules cannot fully assess business appropriateness and organizational priorities. Fig. 1 illustrates the general structure.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

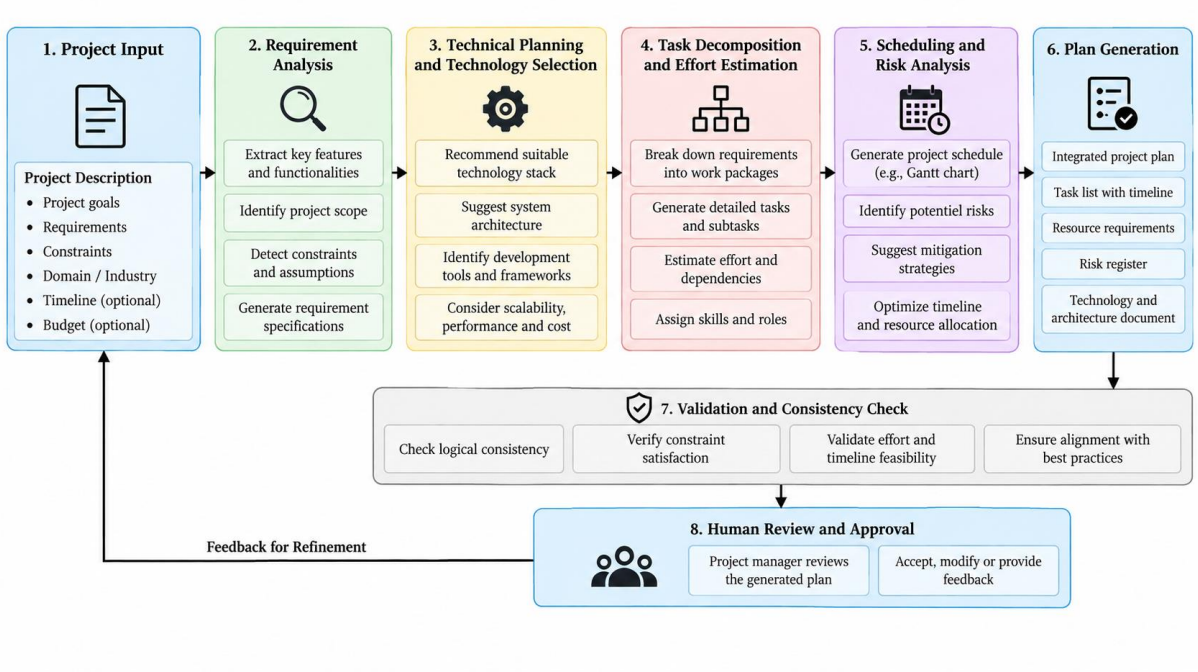


Fig. 1. Architecture of the proposed AI-assisted software project planning system.

VI. PROPOSED SYSTEM ARCHITECTURE

Separating input, generative reasoning, validation, output, approval and storage makes it easy to verify individual stages and identify how planning decisions were made.

Input layer. Collects all data needed, such as description, requirements, goals, constraints, documents and resource information, in a structured form.

LLM processing layer. Proposes candidates for requirements, technical decisions, tasks, estimates, scheduling and risks through the use of templates and structured output formats. Its output will be clearly distinguished from user input or verification information.

Planning and validation layer. Validates candidate artifacts based on explicitly stated constraints (task dependencies, coverage of requirements, scheduling feasibility). Failure results in an actionable warning, e.g., naming the task that has an undefined dependency target.

Output layer. Shows validated or tentative artifacts: requirement descriptions, technology and architecture, tasks with acceptance criteria, estimates and dependencies, scheduling, risk registers and a planning document, and remaining assumptions.

Interface and approval layer. Helps managers and teams to examine and modify the plan, make corrections to estimations, provide feedback, and either approve or reject the recommendations. A history of changes allows comparing the automatically generated plan and the approved plan.

Data storage layer. Stores requirements, artifacts, changes approved by humans, historical data, and metadata using access controls. The history used for estimation is tracked; in fact, past projects may be irrelevant to other technologies and teams.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Workflow. The workflow described in Fig. 2: input and requirement analysis, technical planning, task decomposition, scheduling, and plan generation, and then validation and human approvals. If the initial plan was incomplete or impossible, feedback loops the plan back for correction.

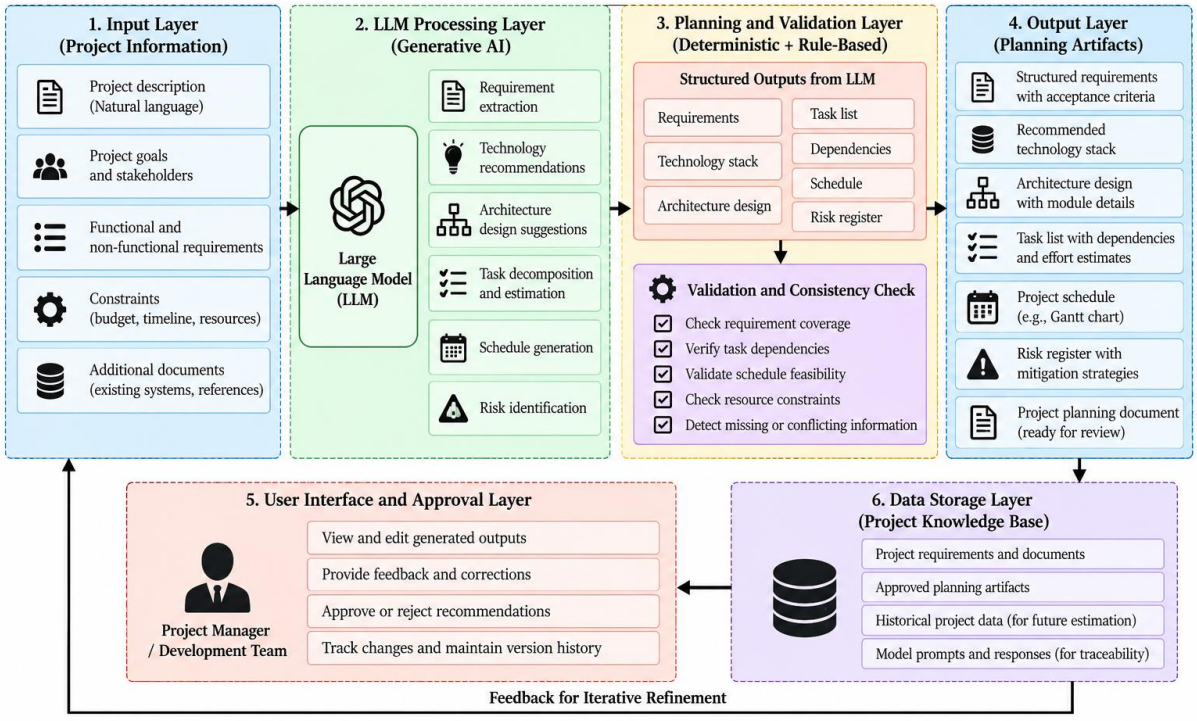


Fig. 2. Proposed AI-assisted software project planning framework.

VII. RELEVANCE TO CURRENT TRENDS

AI-assisted software engineering. AI assistance has moved beyond programming into requirements, task structure, scheduling and documentation; the merit of the latter is determined by quality of artifacts and effort required for their verification, not quantity produced.

Multi-agent planning. Specialized agents (requirement, architecture, dependency, risk) would facilitate a modular approach to planning, however would introduce coordination issues, e.g., conflicts, double-bookings and contradictory assumptions, thus a communicated representation and centralized validation become important.

Human-AI collaboration. The system produces a candidate plan, while the manager brings domain knowledge, performs trade-offs and makes decisions, maintaining professional accountability intact.

Traceable structured output. Stable identifiers connecting requirements, tasks, dependencies and risks make planning easy to verify, tool-based and subject to impact analysis.

VIII. PROPOSED EVALUATION FRAMEWORK

The framework requires empirical evaluation, and the following criteria are offered for this purpose in the future.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

Requirement coverage is the fraction of requirements within the scope that have at least one valid corresponding task:

$$CR = N_{covered} / N_{total} \times 100 \quad (3)$$

However, requirement coverage is not sufficient, because the corresponding task may not implement the requirement, and so an expert evaluation of the mapping is needed.

Task completeness is the ratio of valid fields (description, acceptance criteria, estimation, and dependencies) from the checklist, prepared prior to the experiment.

Estimation error is the difference between predicted and observed efforts for completed tasks; let there be n tasks in consistent units, such as person-hours:

$$MAE = (1/n) \Sigma_{i=1}^{n} |A_i - P_i| \quad (4)$$

Dependency consistency is the proportion of dependencies adhering to structural rules (valid references, no self-references, no cycles, precedence adhered to) and is reported separately from expert assessment of their appropriateness.

Schedule feasibility is the question of whether the schedule adheres to dependencies, deadlines, calendars and resource constraints, with all unsolved assumptions stated. The planning time is the interval from the start of planning to its completion with reviewers' approval, including time spent on review and revisions; fast generation is not efficiency if it needs many corrections afterwards.

Experimental design. Future research could compare (1) manual planning, (2) planning with LLM assistance without deterministic validation, and (3) planning with validation and approval by humans, using similar projects and criteria, similar participants and condition order. Independent reviewers would evaluate the coverage of tasks, quality of tasks, dependencies, risk relevancy and schedule feasibility. In addition, this experiment must include information on version of model, prompts, rules and procedures used. This paper does not claim any improvements in performance, as it is yet to be tested.

IX. RESEARCH GAPS

- Lack of standardized benchmarks for assessing generated project plans.
- Estimation unreliability: LLMs can generate estimates with no data on the capability or past performance of the team; systems should reflect uncertainty and test against actual data.
- Consistency across artifacts: separately generated requirements, tasks, schedules, and risks might conflict with each other.
- End-to-end evidence: success in estimating one task doesn't mean that it will be successful overall in all the software estimation process.
- Oversight and explainability: managers need to know the reasons behind the task, estimate, and risk generation.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

- Adaptation: practices vary between different areas and various techniques used, so historical comparisons might not apply.

X. LIMITATIONS AND FUTURE WORK

Limitations. Output quality depends on input quality, since missing requirements or vague constraints give incomplete plans. LLMs can make mistakes or unsupported assumptions that validation cannot fully catch, and estimates and schedules suffer without historical, resource or dependency data, so a generated timeline is not a commitment. The literature covers software engineering broadly rather than planning directly, and the framework itself is untested, so no gains in productivity, accuracy or cost are established.

Future work. The next step is a prototype combining an LLM with a requirement schema, task representation, dependency graph, validation engine and review interface. Further directions include evidence-based estimation from protected historical records, multi-agent planning with a central validator, and evaluation on real or carefully built projects with expert review, reproducible prompts and honest reporting of failures and correction time.

XI. CONCLUSION

Generative AI can be used to prepare for software projects by generating requirements, technical recommendations, task breakdowns, estimations, schedule and risk register but a well-written plan does not necessarily mean that it is complete and feasible. In this paper, a framework is suggested that uses LLM-based generation along with validation based on rules and human approval. Also, ten publications have been reviewed, research directions have been compared and evaluation approach is suggested. Main idea of this approach is to combine the ability of generation with constraints, traceability and professional validation. Practical usefulness of the approach should be proved empirically.

REFERENCES

[1] Y. Zhang et al., “A survey on large language models for software engineering,” Science China Information Sciences, vol. 69, Art. no. 141102, 2026, doi: 10.1007/s11432-025-4670-0.

[2] P. He, C. Treude, and D. Lo, “LLM-based multi-agent systems for software engineering: Literature review, vision and the road ahead,” ACM Transactions on Software Engineering and Methodology, vol. 34, no. 5, Art. no. 124, 2025, doi: 10.1145/3712003.

[3] A. Kharrufa et al., “LLMs integration in software engineering team projects: Roles, impact, and a pedagogical design space for AI tools in computing education,” arXiv preprint arXiv:2410.23069, 2024.

[4] F. Song, A. Agarwal, and W. Wen, “The impact of generative AI on collaborative open-source software development: Evidence from GitHub Copilot,” arXiv preprint arXiv:2410.02091, 2024.

[5] A. Pandey et al., “Transforming software development: Evaluating the efficiency and challenges of GitHub Copilot in real-world projects,” arXiv preprint arXiv:2406.17910, 2024.

[6] X. Hou et al., “Large language models for software engineering: A systematic literature review,” ACM Transactions on Software Engineering and Methodology, vol. 33, no. 8, Art. no. 220, 2024, doi: 10.1145/3695988.

BODHIVRUKSHA JOURNAL OF DIVERSE DISCIPLINE
E-ISSN : 3139-1486
An Open Access, Peer-Reviewed, Multidisciplinary Scholarly Journal
Bi-monthly | Multilingual (English, Hindi, Marathi) | Academic Research
@ Volume 2, Issue 5 (September-October 2026) (Special Issue 1) ISSN: 3139-1486 (Online)

[7] T. Sauvola et al., "Future of software development with generative AI," Automated Software Engineering, vol. 31, Art. no. 26, 2024, doi: 10.1007/s10515-024-00426-z.

[8] R. Bairi et al., "CodePlan: Repository-level coding using LLMs and planning," Proceedings of the ACM on Software Engineering, vol. 1, no. FSE, pp. 675–698, 2024, doi: 10.1145/3643757.

[9] A. Nguyen-Duc et al., "Generative artificial intelligence for software engineering—a research agenda," arXiv preprint arXiv:2310.18648, 2023.

[10] A. Fan et al., "Large language models for software engineering: Survey and open problems," in Proc. IEEE/ACM Int. Conf. Software Engineering: Future of Software Engineering (ICSE-FoSE), 2023, pp. 31–53, doi: 10.1109/ICSE-FoSE59343.2023.00008.