Spectrum of Engineering Sciences
ISSN (e) 3007-3138 (p) 3007-312X Volume 4, Issue 6, 2026

A TAXONOMY AND RISK-AWARE CONCEPTUAL FRAMEWORK FOR AGENTIC AI-BASED AUTONOMOUS TASK SELECTION IN SOFTWARE ENGINEERING WORKFLOWS

Saria Irshad[*1], **Dr. Atif Hussain**[2]

[*1,2]*Department of Computer Science, University of Engineering and Technology, Lahore,*

[*1]saira.irshad@tech.uol.edu.pk, [2]aatif@uet.edu.pk

DOI: https://doi.org/10.5281/zenodo.20677525

Keywords
Agentic AI; software engineering workflows; autonomous task selection; LLM agents; human-in-the-loop AI; taxonomy; conceptual framework.

Article History
Received: 07 April 2026
Accepted: 19 May 2026
Published: 13 June 2026

Corresponding Author: *
Saria Irshad

Abstract

Agentic artificial intelligence is changing software engineering assistance by shifting from immediate response to code generation to systems that understand the context, use tools, observe feedback, and decide on follow-up actions. Most AI programming assistants and software agents today focus on tasks like code completion, debugging, testing, or issue handling at repository level, without considering task selection as a distinct, explainable, and measurable decision process layer. In this paper, we propose a taxonomy and risk-aware approach to the concept of agentic AI for autonomous task selection in software engineering processes. Our framework uses developer input and workflow signals for task classification, computes uncertainty and risk estimates, makes decisions about selecting the next suitable action, and refers uncertain or critical cases to human confirmation/clarification. We contribute to literature through a task taxonomy definition, comparison with prior research and gap analysis, design of next action selection architecture, decision policy proposal, and outline of experiment scenarios. The present study represents a survey/conceptual framework type of contribution and will be developed later in prototype-based evaluations.

1. INTRODUCTION:

Software engineering is a complicated activity that entails decision making on an ongoing basis during various phases of the software development life cycle. Software engineers have to comprehend requirements, generate code, conduct tests, debug issues, refactor code, document the software, and make sure that it is production-ready [1][4][5]. However, in real-world settings, such tasks are performed not independently of one another. The same task initiated by the software engineer or in the project might lead to a range of actions. Thus, there might be a need to clarify a requirement, test the code, debug the issue, or obtain approval from a human prior to performing certain tasks. Thus, choosing the next action becomes crucial.

New advancements in Large Language Models and AI-assisted coding tools have altered software engineering practices quite a bit. AI programming assistants currently available can help developers write, debug, document, refactor and test their code. The introduction of such programs increased efficiency in coding and cut down the amount of work necessary to complete basic coding assignments. [2][3] On the other hand, the vast majority of the existing AI systems remain reactive to a large extent. They rely on explicit instructions provided by developers to act in accordance with them. In some cases, it might prove useful. Nevertheless, it is far from being

enough to solve the issue related to determining the next step for the AI at the workflow level.

The concept of Agentic AI expands upon conventional AI support by making systems able to operate as agents that can pursue specific goals. These systems would have abilities to make sense of the situation, devise appropriate actions, employ tools, evaluate the results, and modify actions in response to changes. The significance of such functionality in software engineering is obvious considering the connected nature of various development processes [6][7][8]. Rather than being able to produce code upon request, the agent must assess whether there is enough clarity regarding the requirement, whether some tests need to be run, whether an error log needs debugging, whether some code modification entails refactoring, etc.

Even with the rising popularity of coding assistance tools that are either automated through AI or software engineers themselves, choosing what tasks to carry out is still one decision that needs to be addressed. Many of the current literature have found success in fields like automatic code completion, debugging code, generating tests, automated repairs of repositories, and multi-agent development processes [6][9][11]. The thing about these systems is that they do not address the issue of deciding on the proper actions that need to be carried out, since the task has been pre-selected or chosen even before these systems act on it. Choosing the wrong task could result in incomplete output, bad automation, making unnecessary changes to the code, improper sequence of activities, or not asking for clarification from the human engineer.

The present study attempts to fill this gap with the development of a taxonomy along with an associated risk-sensitive theoretical framework for Agentic AI-enabled autonomous action selection in software engineering processes [10][12]. The suggested theoretical framework considers the selection of the next action as an independent and measurable attribute. Specifically, the purpose of such a framework is to make decisions concerning whether to perform one out of several possible actions, which may include code creation, testing creation, debugging, code refactoring, writing documentation, code review, checking prerequisites for deployment, clarification from humans, or even approval from humans.

Although a comprehensive system for software engineering tasks has not been provided, the core contribution of the paper lies in a structured foundation on which future research can build a safe and understandable approach to software engineering tasks of the agentic nature [13][14][15]. Firstly, related software engineering tasks and current gaps within this context are studied. Secondly, the categories of such tasks are classified and organized into a taxonomy. Lastly, a decision-making model for action selection with consideration of safety risks is suggested. Since the paper itself serves as a basis for future empirical studies, no fictional experiments have been conducted. Instead, an application scenario has been proposed for validation purposes.

The rest of the paper is structured in the following manner. Section 2 addresses related work that has been done in AI-aided software engineering before. The research problem and motivation for the paper are addressed in Section 3. The significance and contributions of the paper are detailed in Section 4. The proposed classification scheme is provided in Section 5. Section 6 provides an introduction to the proposed conceptual model. Section 7 provides the methodology employed in this paper. Section 8 deals with the risk-oriented decision-making policy proposed here. Section 9 provides the gap analysis.

2. Related Tasks Already Studied:

The literature review indicates that there are already a number of relevant topics which have received attention under the umbrella of AI-assisted software engineering. Hence, the novelty of this paper does not lie in arguing that no work on Agentic AI is available. Rather, the novelty consists in defining and modeling next-action selection as an independent decision-making layer in software engineering [16] [17][18]. The previous research and technologies have succeeded in developing approaches related to

code generation, automated debugging, automatic testing, code summarization, code refactoring, documentation generation, issue resolution at the level of repositories, and multi-agent software engineering.

Nevertheless, most previous solutions aim at accomplishing a particular task, which is previously determined by the developer or assigned by some benchmarks. For instance, code generation is performed when the user requests code, debugging recommendations are proposed when there is an error that needs fixing, and test cases are generated if the target function has been determined [19][20]. In all these situations, the action taken by the system is driven by a command from the user and not by the determination that coding, testing, debugging, refactoring, documenting, reviewing, preparation for deployment, or clarification is necessary.

There exists a very significant gap within agentic software engineering as a result. The actual software engineering processes involved do not necessarily carry clear labels [14][16][18]. An engineer could supply an incomplete requirement, a failed test log, a generic bug report, an unfinished module, or even a risky deployment process. In order to determine what action to take in such cases, the agent needs to first understand the state of the workflow before choosing an appropriate action to take. A comparative overview of related software engineering tasks and the gaps left by existing work on AI-assisted and agentic software engineering is provided in Table 1 below.

Table 1. Summary of related software engineering tasks and remaining gaps.

Related task	What has been done	Gap for this paper
Code generation and completion [3][5][9]	LLMs and coding assistants generate functions, snippets, and explanations from natural-language prompts.	They usually respond to a requested task rather than deciding which task should happen next.
Repository-level issue resolution [11][14][15]	Benchmarks and agents attempt to solve GitHub issues by editing code, running tests, and submitting patches.	The emphasis is on solving a known issue, not general workflow classification across many task types.
Automated testing [1][6][7][8]	AI and search-based methods generate tests, improve coverage, and analyze failing outputs.	Testing is treated as a target activity, while the decision to test versus debug or clarify is less explicit.
Debugging and bug repair [14][16][17]	Agents localize faults, explain stack traces, and produce candidate fixes.	Debugging is commonly triggered by an observed failure rather than selected through a general task-selection policy.
Refactoring and maintainability [20][22][25]	Tools suggest cleaner structure, remove smells, or improve maintainability.	Refactoring is rarely balanced against risk, test readiness, documentation needs, or human approval.
Documentation generation [11][14][19][24]	LLMs summarize code, create comments, and produce usage documents.	Documentation is often requested directly rather than selected as a post-change workflow action.
Human-in-the-loop control [6][8][12][16]	Research highlights oversight, approval, and clarification for safer AI systems.	Clarification is often discussed as a safety concept but not encoded as a formal action class in task taxonomy.

As the table demonstrates, substantial progress has been achieved in the areas of code generation, testing, debugging, refactoring, documentation, and workflow automation among other specific tasks. However, most solutions presented in the literature do not provide autonomous software engineering action selection; instead, most of them rely on specific task execution and do not account for choosing the next appropriate action within a particular software development process. At the same time, most of the existing AI coding assistants are reactive because of the need to receive prompts from users and analyze the current state of workflow, potential risks, uncertainties, and requirements for human involvement in the decision-making process. Thus, the gap that has to be addressed is the lack of an actionable and explainable task selection mechanism that would enable the system to select a suitable software engineering task, such as code generation or debugging.

3. Research Problem and Motivation:

The central issue that needs to be tackled in this paper is the lack of an organized and understandable workflow process where an Agentic AI system can make decisions about what would be the appropriate next step in the software engineering process. AI-based software engineering assistants are able to perform many different functions such as generating code, explaining programming errors, creating tests, summarizing documentation, and debugging code [3][4][8]. Nevertheless, in these types of AI-based assistants, it could be necessary for the user to explicitly guide their work.

Unlike in ideal software engineering practices, it may not always be clear what to do next. A patch that is generated might have to be tested before it can be deemed reliable enough. An unsuccessful test might call for debugging before any new code is developed. An unclear requirement would need clarifying before coding begins. A refactoring request may have to be approved by humans because it could influence system behavior, maintenance, or compatibility with other modules. In a similar vein, a deployment-related request may have to undergo verification and even be rolled back instead of being acted on immediately [16][18]. The question here is not about AI's ability to carry out a certain task, but its ability to choose the appropriate one.

This makes the problem more critical considering that Agentic AI systems could be given access to development tools including IDEs, test-runners, static-analysis tools, package-managers, documentation generators, version-control systems, issue-trackers, CI/CD pipelines, and repositories APIs. Access to these tools will make these agentic systems even more useful since these AI agents will be able to work in the real development environment rather than responding only in natural language texts. Nevertheless, it will create more dangers associated with unsafe or improperly ordered activities of the AI agents. For instance, an agent that alters some source code without executing the tests on it can bring defects into the application.

This is why there is a requirement for a task selection system, which will include knowledge about the software development process, risks, and the involvement of a person. This system must analyze the workflow state, classify the task itself, estimate uncertainty, check if necessary software exists, and estimate risks related to the chosen action. It must make a decision to either execute the process, suggest an action, get further clarifications, or seek human confirmation. Human clarification and human confirmation become part of a valid workflow process rather than its exception or failure.

4. Novelty and Contributions:

Innovation offered by the paper includes creation of an awareness-driven task-selection layer for software engineering agents. Unlike typical intelligent systems, whose scope involves performance of specific activities like generating code, debugging, testing, and documenting etc., the scope of innovation presented in the paper is to define the way an agent can select amongst several options for actions within its workflow. What distinguishes this innovation from typical AI-based programming tools is that the core

objective here is more about selection rather than execution.

- A software engineering task taxonomy which maps developer inputs and evidence about project state to types of next actions.
- A risk-aware decision model where confidence, ambiguity, available tools, and impact level are assessed prior to taking action.
- An escalation approach with humans in the loop in which clarifications or approvals are considered outputs of the decision process.
- A procedure for the evaluation of prototypes in which labeled software engineering scenarios, confusion matrices, and action accuracy are used.
- A publication-ready theoretical framework that will subsequently serve as a basis for an empirical prototype paper.

The innovative aspect of the research can be considered as the identification of the problem of selecting the next action to perform as an autonomous decision problem in the context of software engineering tasks. In most existing software systems, it is assumed that the user gives the instructions to the software on how to behave – to write a code, to fix an error, and to generate tests. Nevertheless, in practice, there is no clear indication in the user's request or current project state on which particular action should be performed. The actions may include clarifying the request, generating the code, writing tests, debugging, etc. This research is aimed at solving this problem using.

The second proposal is the introduction of the risk-based decision-making approach. This method will not enable the agent to perform the action simply based on the fact that the category of tasks has been determined. In addition to determining the task category, the decision-making process will consider such factors as confidence, ambiguity, tools, workflow, and level of impact. For instance, while the task of writing documentation can be performed by the agent as the risk involved is low, the task of database migration, authentication modification, or anything deployment-related should involve human confirmation despite task categorization. Further, the paper adds an approach to human-in-the-loop escalation, where clarification and approval are defined as outputs of an action. In other words, asking the user to provide missing information or to give their approval for changes involving high risk is no longer seen as a failure of automation but rather as part of the decision process used by the agent. This is significant because there are many cases in software engineering where there is a certain level of uncertainty, domain knowledge, security implications, or business considerations.

Moreover, this work will provide a methodological way of assessing future prototypes through labeled software engineering scenarios. These scenarios could include development queries, snippets of code, error logs, failed test cases, documentation, refactoring instances, and unclear requirements. Future evaluation can consider factors like classification performance, correct action selection, accuracy of human escalation, confusion matrices, explanations offered, and expert review. Through such an approach, the current conceptual paper can be developed into a future empirical study.

The overall nature of the contribution made can thus be defined as being both theoretical and taxonomic, while having a definite validation strategy that could be employed during future experiments with the concept. The paper provides a framework for an article suitable for publication at the initial stages of research, avoiding any unfounded assertions that prior to this time no one had ever researched anything like this topic before. It does not deny the fact that existing literature has examined individual software engineering activities.

5. Proposed Taxonomy:

The suggested taxonomy categorizes observable software engineering scenarios into task selection classes that have certain inputs, suggested course of actions to follow, and safety issues to consider. This taxonomy is aimed at helping the Agentic AI system determine the current workflow context before it takes any actions [21] [22]. Unlike treating all developer's inputs as mere commands, the taxonomy distinguishes between several

software engineering tasks like requirement clarification, code writing, test generation, debugging, refactoring, documentation, review, pre-deployment, and human confirmation.

The objective behind developing this taxonomy was to establish a systematic way in which evidence available during a software engineering process could map to the appropriate action to be taken subsequently. Thus, for instance, evidence of a clear requirement would mean the choice of code generation, while a failed test output would translate into a debugging activity [23][24]. In addition, a unit-test need not being satisfied would suggest the necessity of test generation, while an ambiguous requirement would necessitate human intervention. This also applies to requirements pertaining to authentication, payments, database migration, and deployment configuration.

Every one of the classes of tasks selection has three main components. First, there is the input signal, that is the data available to the agent based on which he makes a decision; it could be a developer's request, some piece of source code, error, testing output, requirement for documentations, etc. Second, there is the proposed next action; it describes the action to perform and could be writing code, performing tests, debugging errors, refactoring, generating documentation, asking questions, or requesting an approval [25][26][27]. Third, there is the safety condition; it defines whether the next action could be safely recommended, needs developer's approval, or has to be rejected.

The taxonomy is useful for explainability in the sense that it is possible to justify any chosen action through the signals that have triggered the action. For example, if debugging is chosen by the agent, it can explain that the reason was because the input included an exception trace, failing test outputs, or errors in the runtime. On the other hand, if the clarification action is chosen by the agent, it means that the requirement had issues with completeness or ambiguity.

Table 2. Proposed taxonomy of autonomous task-selection categories.

ID	Task class	Observable signals	Recommended next action	Risk level
T1	Requirement clarification	Ambiguous request, missing constraints, conflicting business rules	Ask targeted questions; request examples or acceptance criteria	Low to high depending on ambiguity
T2	Code generation	Clear feature request, API description, expected behavior	Generate code or implementation plan	Medium; may require tests
T3	Test creation/execution	New feature, changed logic, missing coverage, regression risk	Create tests or run relevant tests	Medium
T4	Debugging	Error logs, stack trace, failing tests, runtime exception	Localize cause and propose fix	Medium to high
T5	Refactoring	Code smell, duplication, complexity, maintainability issue	Suggest refactor and preserve behavior	Medium to high

T6	Documentation	Completed feature, public API, complex logic, onboarding need	Write docs, comments, or README updates	Low to medium
T7	Code review	Patch available, pull request context, quality concern	Review design, correctness, security, and style	Medium
T8	Deployment pre-check	Release request, dependency update, configuration change	Run checks; verify build, tests, environment, rollback plan	High
T9	Human approval	High-impact edit, security-sensitive change, low confidence	Escalate to human decision before execution	High

6. Proposed Conceptual Framework:

Inputs to the framework can be derived from software engineering inputs and workflow-state data. Some of these inputs include developer requests, requirements, code snippets, error messages, failed test results, documentation requirements, refactoring requirements, change in repositories, or deployment actions. The input layer is tasked with gathering enough inputs to help the agent understand the current software engineering environment before making an action selection decision.

Contextual information will be extracted by the system using the received input. The signals may be related to intent, code-related signals, errors, testing, ambiguity, dependencies, requirements for tools, and potential risks. For instance, an exception trace may represent a debugging signal, while a new requirement would trigger code generation. Similarly, a missing test case might trigger test generation, while an ambiguous requirement would represent a need for clarification. Understanding the context at this stage is important since the user's request can yield different outcomes based on the state of the workflow.

The extracted signals are then mapped to the task taxonomy that is introduced. This task taxonomy serves to categorize the current context into one particular task selection category, including such categories as code generation, testing, debugging, refactoring, documentation, code review, deployment pre-check, human clarification, and human validation. Yet the framework does not perceive the classification result as the final choice, because the right classification alone might not necessarily make the execution safe.

Due to this, the decision-making process utilizes a risk and uncertainty gate prior to making the final decision. This gate considers the confidence level, the degree of ambiguity, impact magnitude, tool preparation status, and whether human intervention is required. Decisions with low risks can be either recommendations or decisions made through control, but those with high risks are submitted to human intervention. Where the input data is inadequate or ambiguous, human clarification is chosen over an autonomous choice.

Following the gate of risk and uncertainty, the decision policy picks the most suitable action among others. These actions could include the creation of code, testing, debugging of an error, refactoring of code, documentation, revision, preparation of deployment tests, questioning, or human approval. Following the decision, the explanation module gives an explanation behind the selection so that the developer understands why the software agent recommended this particular course of action. Lastly, the feedback

module captures information about the result like whether a test was successful, errors were corrected by the user, or the action was not executed successfully.

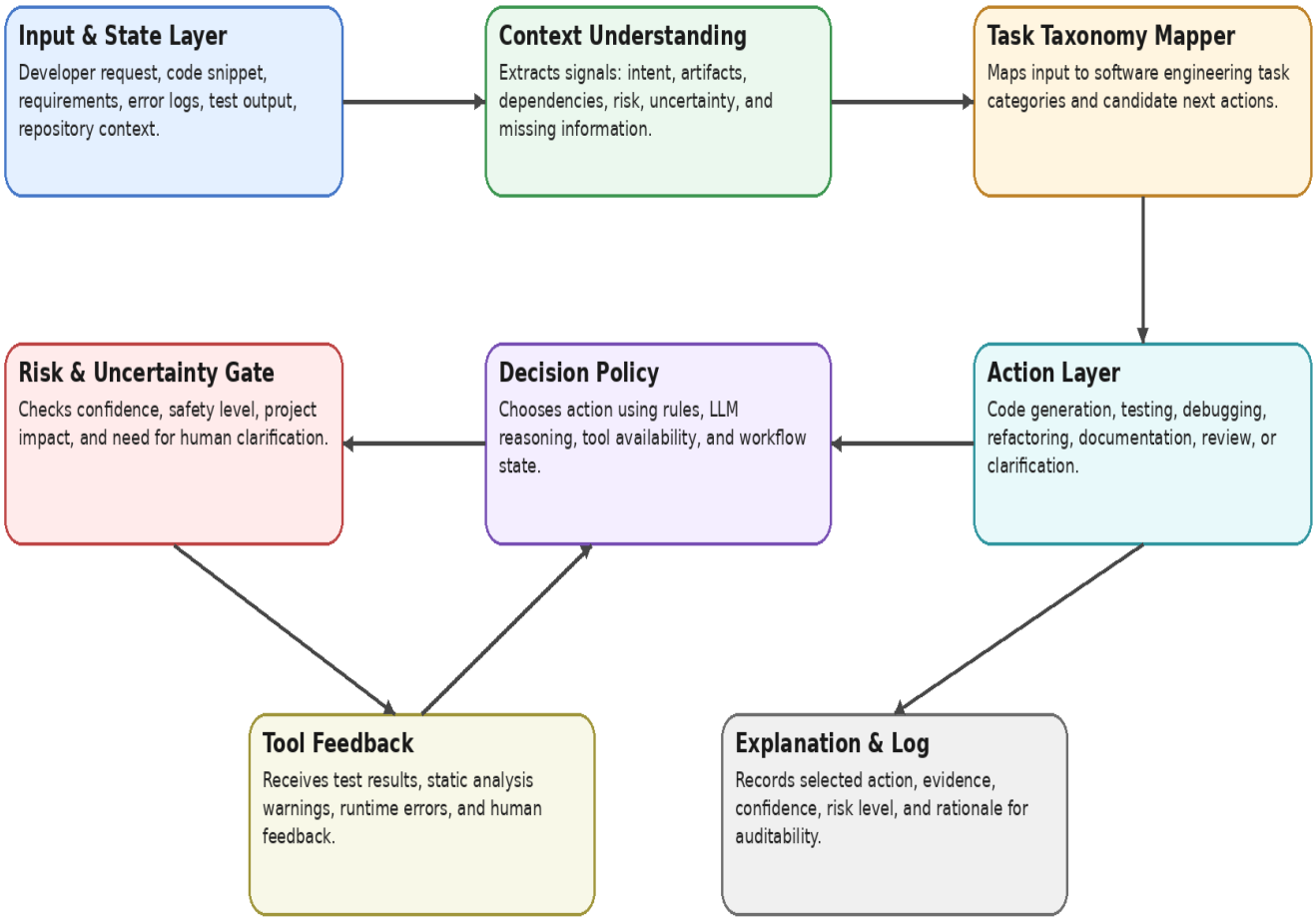

Figure 1. Proposed risk-aware agentic task-selection architecture.

The suggested architecture is outlined in Figure 1 below. It involves the initial step involving the software engineering process and its workflow state, the extraction of contextual cues, mapping to a task hierarchy, the application of risk and uncertainty gate, followed by the decision policy selection and the recommended or performed action.

7. Methodology:

The research methodology adopted in this case is that of survey research and concept formation. The study uses the method since the aim of the paper is not to propose an industrial application of the proposed framework but rather conduct research to find out the emerging trends, the requirements of task selection, create taxonomy, propose a risk-based decision framework and suggest a validation strategy. It fits perfectly to the aim of the thesis proposal which revolves around the areas of taxonomy creation, framework formation, prototype construction, and evaluation.

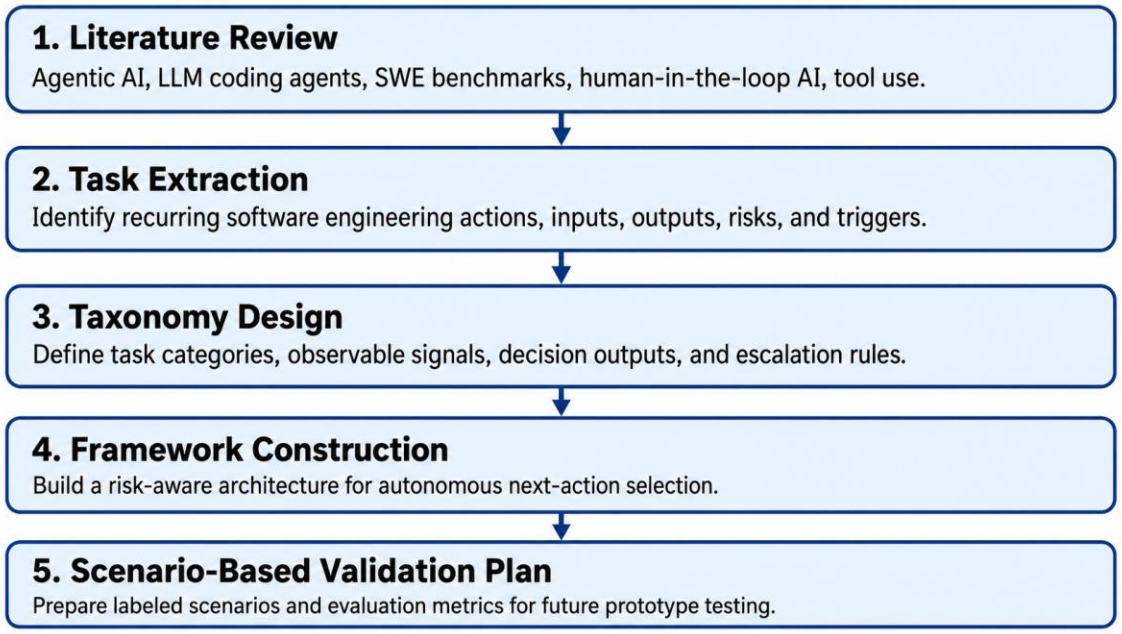

Figure 2. Methodological flow used to develop the taxonomy and conceptual framework.

Figure 2 shows the methodology that would be adopted for the development of the Agentic AI-based autonomous next action selection framework. The first step in the method is conducting literature review, where previous research on Agentic AI, LLM-based code generation agents, software engineering benchmarks, human in the loop techniques, and tooling augmented techniques will be analyzed. Once a thorough literature review is carried out, recurring software engineering tasks, inputs, outputs, risks, and triggers for decision making are identified. Identified items are then categorized and organized into a taxonomy, which categorizes the tasks, the observable signals, the output decisions made based on these observable signals, and the triggering criteria to elevate the decision-making from the agent to humans. Using the created taxonomy, a risk-aware framework will be developed to perform the autonomous next action selection.

Table 3. Methodological stages of the study.

Stage	Process	Output
Literature review	Search agentic AI, LLM coding agents, SWE benchmarks, tool use, human-in-the-loop systems.	Set of relevant studies and extracted themes.
Thematic coding	Extract software engineering inputs, actions, outcomes, and risks from the literature.	Initial list of task categories and decision factors.
Taxonomy design	Group recurring tasks into action classes with observable signals and risk levels.	Task-selection taxonomy.
Framework design	Define modules for input, context understanding, classification, risk gate, action selection, feedback, and explanation.	Conceptual architecture.

Validation plan	Define scenario set, labels, metrics, and expert review process for later prototype testing.	Future empirical evaluation protocol.

Table 3 shows the methodology steps taken in this study to develop the proposed Agentic AI-based autonomous task-selection framework. First, a literature survey will be conducted to gain insight into existing research related to Agentic AI, language model-based software engineering agents, task planning, tool usage, and human-in-the-loop systems. In the following step, recurring software engineering tasks and decision signals will be gathered based on the information found in the literature and the proposal scope. Next, the collected information will be arranged in a taxonomy that describes task categories, input signals, decision outputs, risk levels, and conditions under which escalation to humans should occur. After completing the creation of the taxonomy, the framework can be developed for task selection with consideration of risks involved. At last, a scenario-based validation process will be planned to evaluate the framework in future work.

8. Risk-Aware Decision Policy:

The decision policy distinguishes between task categorization and task authorization. The task could be categorized appropriately, but the action could still require authorization due to the risks, incompleteness, or impact on system availability. For instance, a database migration operation could be classified as a code generation task or one requiring deployment preparation, but the agent must not execute it unless confirmed and authorized by a human being. It is imperative to make the distinction between appropriate task identification and the safety of performing the task autonomously.

The agent begins by categorizing the task, for example, code generation, debugging, testing, refactoring, documentation, or getting ready for deployment. After categorization is done, the risk level is assessed to determine whether the action suggested will have a low, medium, or high risk level. For tasks with a low risk level, like writing documentation, explaining the code, and suggesting the creation of unit tests, they may be suggested outright [24][25][26]. Actions that carry a moderate level of risk include refactoring functions and changing code that does not have any vital purpose; this may be done optionally along with an explanation. However, for a high risk task, human intervention is needed.

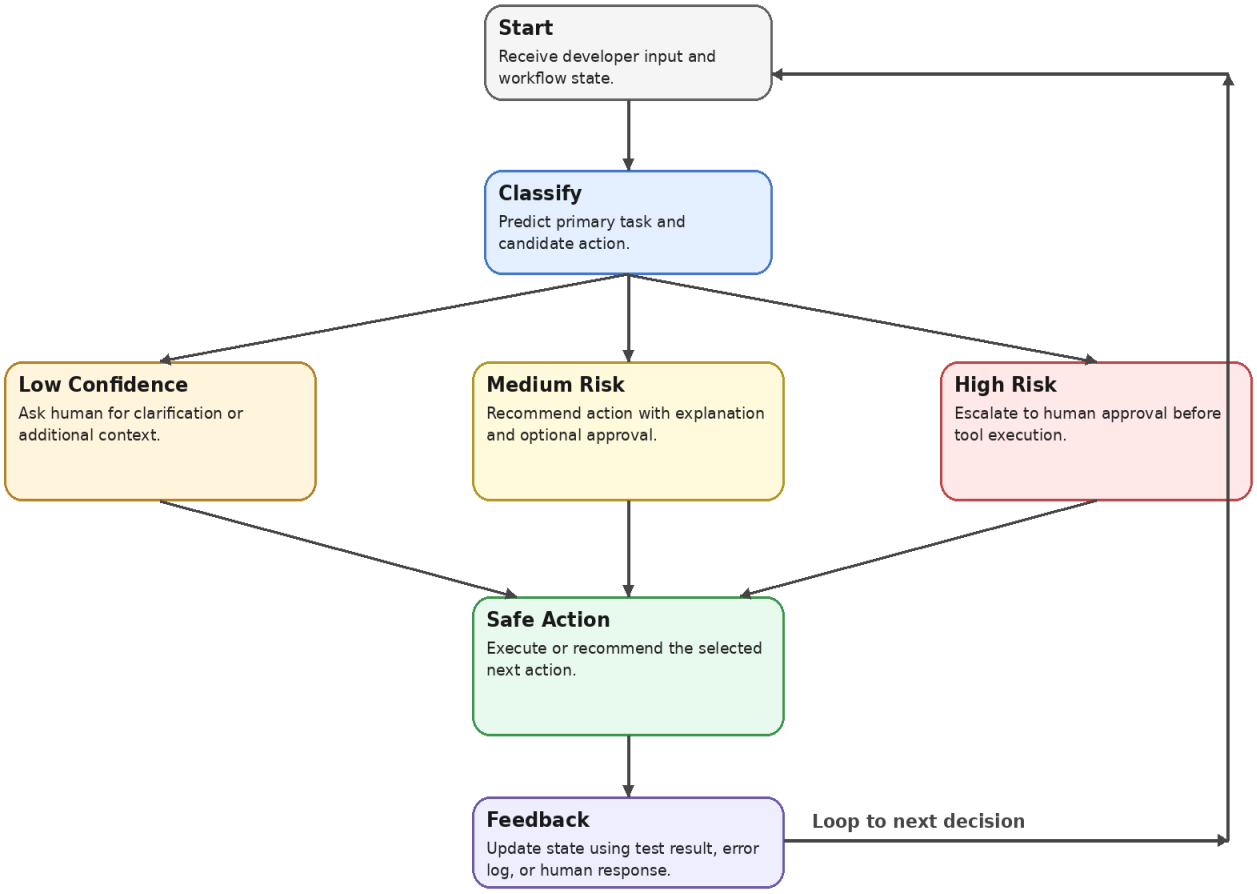

Figure 3. Decision logic for confidence, risk, human clarification, and action selection.

The decision-making logic of the proposed framework in choosing between confidence, risk, human clarification, and action execution is demonstrated by Figure 3 below. First, the framework makes use of the information provided in the form of a software engineering input, which may come in different forms including but not limited to developer requests, pieces of code, logs, errors, testing failures, and documentation needs [27][28]. In the first step, the framework determines the type of the action that has been requested by the user and computes the confidence level regarding the chosen type. If the confidence level is low, no action will be performed; instead, human clarification will be chosen as the most appropriate course of action. However, in case the confidence level is acceptable, the risk level of the action is calculated and determined. Depending on the risk level of the action (low or moderate), human clarification will take place and developer confirmation will be needed before proceeding further.

Table 4. Decision factors used by the proposed task-selection policy.

Factor	Description	Influence on decision
Confidence score	Probability or agreement level for the predicted task class.	Low confidence triggers clarification.
Risk level	Estimated impact of the recommended action on correctness, security, data, or deployment.	High risk triggers human approval.
Tool readiness	Availability of required tools such as test runner, linter, build tool, or documentation generator.	Missing tool leads to recommendation rather than execution.
Workflow state	Previous action, current artifact, error feedback, and test status.	Prevents poor sequencing such as documenting before code stabilizes.
Explanation quality	Whether the selected action can be justified with observable evidence.	Weak explanation lowers trust and may require review.

The following Table 4 identifies the principal decision criteria adopted by the suggested task selection policy to determine the most appropriate course of action in the software engineering process. The criteria ensure that the agent's decision process does not solely rely on the classification of tasks but also takes into account their context completeness, confidence, severity, tool availability, and human intervention necessity. For instance, the task might be identified as the debugging task; however, when the problem involves an impact on authentication, transactions, database operations, or deployment configurations, the decision must take into account higher risk levels associated with the course of action and trigger the human validation step before proceeding with the action itself. Similarly, incomplete developer requests must be followed by the clarification step rather than automatic generation and modification.

9. Comparative Gap Analysis:

The above comparison clearly illustrates the necessity for a framework to select tasks. Although existing frameworks have provided useful pieces of the puzzle, the current work has attempted to integrate all of the taxonomy, workflow state, risk gate, reasoning, and human input into one model for selecting next actions [29][30]. Prior solutions mostly concentrate on doing a certain task once the task has been identified by the user such as code generation, bug fixing, test generation, or documentation summary. However, in practical software engineering scenarios, the most challenging part may actually be determining which task to do next.

This becomes an issue due to the fact that software development is a multi-step process, requiring different decisions to be taken along the way; for instance, a developer request may need some clarification, generated code needs to be tested before being accepted, code testing failure might need debugging, or a potentially dangerous action needs to be approved by a person first. Even though there are existing software systems assisting humans in coding, these AI applications may lack the layer of decision-making necessary for choosing which step should be taken next.

Table 5. Comparative gap analysis of related work and the proposed paper.

Work direction	Main focus	Remaining limitation / difference
SWE-bench style evaluation [7][13][17]	Repository-level issue solving with real software tasks.	Focuses on solving known issues, not a general next-action taxonomy.
SWE-agent style systems [3][4][9][22]	Use tools to inspect repositories, edit files, and run tests.	Strong execution pipeline, but task selection is tied to issue-solving workflows.
AgentCoder and multi-agent coding [6][13][16][19]	Specialized agents for coding, testing, and verification.	Agent roles are predefined; dynamic task selection remains limited.
Tool-use benchmarks [4][7][8]	Measure API/tool calling ability and error handling.	They evaluate tool usage, not software workflow action choice.
Human-in-the-loop AI [13][15][26]	Improves safety, oversight, and trust.	Often separate from task taxonomy; clarification is not always modeled as an output class.
Risk-aware conceptual framework [14][17][22]	Taxonomy plus risk-aware conceptual framework for autonomous software engineering task selection.	Targets the decision layer before execution and provides a structured validation plan.

This is depicted in Table 5 below. From the table above, it can be seen that previous studies have made substantial contributions to the field of AI-assisted software engineering by means of code generation, debugging, testing, repository level repairing, benchmarking, and agent based automation of software engineering processes. Most of these studies either concentrate on a single task in software engineering or assess their agents once a task assignment has been done. In contrast, the proposed paper concentrates on the early decision making layer.

10. Validation Plan for Future Prototype:

The future prototype could employ a scenario dataset that includes developer requests, code snippets, error logs, test results, and documentation requirements. The dataset could be labeled manually based on the intended task category, appropriate actions, risk levels, confidence requirements, and whether human intervention is required. Validation of the agent's action could then involve checking the correctness of the selected action against predefined labels for the given scenarios. If the failure to pass a unit test is considered a debugging scenario, the action will need to be compared with ground-truth labels indicating the correct action to take. The same will apply in cases where the scenario involves feature requests, high-risk requests, authentication, payment processing, database migrations, and deployment configurations.

The future assessment process can include qualitative and quantitative assessments. The quantitative assessment could be carried out through the measurement of classification accuracy, action selection correctness, precision, recall, F1 score, confusion matrix values, and human escalation accuracy. The qualitative assessment can be done by checking whether the reasoning provided by the agent is comprehensible, whether the risky tasks are performed properly, and whether the next action selected is helpful for the software engineer developer.

Table 6. Proposed validation plan for future empirical work.

Component	Description	Purpose
Scenario set	50-150 representative software engineering scenarios.	Each scenario labeled with expected task class and action.
Baselines	Rule-based classifier and prompt-based LLM classifier.	Compare simple deterministic logic with LLM-assisted reasoning.
Metrics	Accuracy, macro F1, confusion matrix, action-selection correctness, clarification precision.	Measure both classification and practical action quality.
Expert review	Supervisor or software engineering practitioners review disputed labels.	Improve validity of expected actions.
Error analysis	Analyze false positives, false negatives, unsafe actions, and unnecessary clarifications.	Identify weaknesses and refine taxonomy.

A proposed validation plan for empirical testing of the task selection framework is shown in Table 6 below. Because this paper is written as a taxonomy and conceptual framework, the validation plan outlines how the proposed model may be empirically validated in future studies using actual experiments, but not by fabricating data. The validation plan involves creating a dataset that consists of various scenarios from software engineering practice, such as developer queries, code samples, error messages, failed tests output, refactoring needs, documentation, and ambiguous requirement cases. These scenarios will have an annotation with the expected task type, action to take, risk level, confidence threshold, and human consultation needed.

11. Expected Outcomes:

Outcome of This Research Paper

This research paper will deliver the expected outcome in form of a foundation which will be useful for developing software engineering agents. This outcome will be publishable and does not require immediate experiments since they can be conducted later using this concept.

- Explicit taxonomies of task types and decision outputs in agentic software engineering processes.
- A risk-sensitive conceptual architecture of next action selection.
- A decision policy that considers confidence, risk, tool-readiness, process status, and explanation quality.
- Scenario-driven testability strategy that allows the conceptual paper to be followed up empirically.
- More conservative treatment of novelty that does not rely on unsupportable statements of uniqueness.

Another anticipated result would be the design of a scenario-based validation scheme that would enable the transformation of the existing theoretical paper into an empirical follow-up investigation. The future investigators could generate scenarios of software engineering cases that will include developers' inquiries, code samples, error messages, test results, documentation requirements, and vague requirements. Such scenarios could then serve as the basis for evaluating the accuracy of task classification, action selection, human escalation, and explanation generation.

Last but not least, the paper offers a safer perspective on novelty by not making unjustified assertions such as none existing before. Rather than asserting that Agentic AI has not been researched in software engineering, the paper grounds its novelty on the particular layer of decision-making where autonomous action selection is considered. This is a stronger

perspective from an academic viewpoint, as it considers previous research in AI programming assistants, software engineering agents, and automatic software development without missing out on defining a gap.

12. Limitations:

This research article addresses only the issues related to taxonomies and the conceptual framework. No prototype implementations, experimental findings, or industry deployment data are presented. Hence, the assertions in this article are purely theoretical and analytical. This framework helps us understand how the Agentic AI could perform task classification, risk evaluation, selection of actions, and human interaction in such systems. However, its effectiveness has yet to be verified through implementation and experimentation.

Another limitation of this approach is its reliance on labeling and scenario quality. Future testing data sets containing poorly defined, biased, incomplete, or unrealistic scenarios will provide inaccurate evaluations of task selection quality. Furthermore, expert-generated ground truths used for testing might also be inconsistent among developers who could make different decisions based on the specifics of their projects, coding standards, risk preferences, and experience in specific fields. In other words, a correct solution to the problem of task selection may differ among developers.

It is also constrained in terms of the range of the selected tasks categories. For instance, this work covers common actions that may be performed while doing software engineering like generating code, testing, debugging, refactoring, documenting, reviewing, preparing for release, and clarifying to a person. Yet, there might be other activities in actual software development like negotiating the requirements, designing an architecture, auditing for security issues, optimizing for performance, managing dependencies, DevOps monitoring, or responding to incidents.

13. Conclusion:

The paper presented a classification as well as a risk-sensitive framework for Task Selection for Agentic AI-driven Autonomy in software engineering processes. The main novelty of the paper consists of the decision layer that decides what the next appropriate action could be - code generation, testing, debugging, refactoring, documentation, code review, deployment pre-check, clarification from humans, or human approval. Unlike regular AI code assistants which operate primarily on requests from users, the framework in question aims to determine the appropriate workflow state and action.

The paper emphasizes the importance of the issue of automatic task choice as a significant yet understudied challenge in the context of agency-oriented software development. While most existing systems have exhibited advanced levels of competence in generating code, performing tests and debugging, automating version control and other related actions, they still lack the capability to make a decision about whether to perform certain actions at the right time, whether to postpone, ask questions, etc. In order to solve the problem, the authors suggest integrating several aspects within one framework.

The presented taxonomy systematizes typical software engineering tasks based on their structure, input triggers, output options, and risk. That allows the agent to identify whether the provided input refers to code development, testing failure, diagnostic problem, refactoring possibility, documentation issue, or ambiguous requirement. The risk-sensitive decision-making policy also guarantees that proper task recognition will not guarantee autonomous action. Such high-risk tasks as changes related to authentication procedures, migration of databases, payment logic issues, deployment preparation, and destructive tasks must receive human permission for execution.

On the whole, the structure provides better safety, explainability, and robustness to agentic software engineering processes. It can be considered as a conceptual publication at an early stage since it does not have any fake experimentation data. Rather, the framework lays

down an organized basis for future prototype testing. In the future, researchers can construct the architectural model, create a labeled scenario database, test the rule-based and LLM-based decision-making techniques, and assess the framework by means of metrics like task-classification accuracy, action selection efficiency, human escalation precision, explanations, and developer usefulness.

REFERENCES

[1] Yousif, I. (2025). AI-Driven Autonomous Manufacturing: A Novel Taxonomy, Vision-Guided Planning, and Diversity-Aware Active Learning Frameworks.

[2] Banerjie, S., Zhu, Y., Freeman, I., Machado, J. V., Ahmed, A., Sarker, A., & Al-Garadi, M. (2025). Agentic AI in Healthcare: A Comprehensive Survey of Foundations, Taxonomy, and Applications. *Authorea Preprints*.

[3] Kiasari, M., & Aly, H. (2026). Agentic Artificial Intelligence for Smart Grids: A Comprehensive Review of Autonomous, Safe, and Explainable Control Frameworks. *Energies*, *19*(3), 617.

[4] Adabara, I., Sadiq, B. O., Shuaibu, A. N., Danjuma, Y. I., & Maninti, V. (2025). Trustworthy agentic AI systems: a cross-layer review of architectures, threat models, and governance strategies for real-world deployment. *F1000Research*, *14*(905), 905.

[5] Navneet, S. K., & Chandra, J. (2025). Rethinking autonomy: Preventing failures in AI-driven software engineering. *arXiv preprint arXiv:2508.11824*.

[6] Gadekallu, T. R., Zhao, Y., Wen, Z., Bhattacharya, P., Xia, Y., Cai, J., ... & Feng, H. (2026). Artificial Intelligence of Things as a Foundation for Agentic AI Systems: Architectures, Applications, and Challenges.

[7] Boddapati, K. B., Madduri, P., & Turaga, K. (2026, March). Intelligent Risk-Aware Release Governance Using Agentic AI in Cloud-Native P&C Insurance Ecosystems. In *2026 IEEE International Conference on AI Engineering and Innovations (AIEI)* (pp. 1-5). IEEE.

[8] Nibir, T. (2026). *Agentic vs Rule-Based Process Automation-A Comparative Study* (Doctoral dissertation, University of Jyväskylä).

[9] Acharya, D. B., Kuppan, K., & Divya, B. (2025). Agentic AI: Autonomous intelligence for complex goals—A comprehensive survey. *IEEe Access*, *13*, 18912-18936.

[10] Wang, H., Gong, J., Zhang, H., Xu, J., & Wang, Z. (2025). Ai agentic programming: A survey of techniques, challenges, and opportunities. *arXiv preprint arXiv:2508.11126*.

[11] Ruotsalainen, E. (2025). Systematic Literature Review of Agentic AI and AIOps Across Software Lifecycle.

[12] Akshathala, S., Adnan, B., Ramesh, M., Vaidhyanathan, K., Muhammed, B., & Parthasarathy, K. (2025). Beyond Task Completion: An Assessment Framework for Evaluating Agentic AI Systems. *arXiv preprint arXiv:2512.12791*.

[13] Reda, E., Yasser, M., & El-Metwally, S. (2026). Autonoma: A Hierarchical Multi-Agent Framework for End-to-End Workflow Automation. *arXiv preprint arXiv:2603.19270*.

[14] Horne, D. (2025, April). The agentic ai mindset–a practitioner's guide to architectures, patterns, and future directions for autonomy and automation. In *International Conference on the AI Revolution* (pp. 434-455). Cham: Springer Nature Switzerland.

[15] Biswas, A., & Talukdar, W. (2025). *Building Agentic AI Systems: Create intelligent, autonomous AI agents that can reason, plan, and adapt*. Packt Publishing Ltd.

[16] Hughes, L., Dwivedi, Y. K., Malik, T., Shawosh, M., Albashrawi, M. A., Jeon, I., ... & Walton, P. (2025). AI agents and agentic systems: A multi-expert analysis. *Journal of Computer Information Systems*, *65*(4), 489-517.

[17] Nagvekar, R. (2025, December). Agentic AI-Driven CI/CD Pipelines for Autonomous Software Delivery. In *2025 IEEE 5th International Conference on ICT in Business Industry & Government (ICTBIG)* (pp. 1-4). IEEE.

[18] Banerjie, S., Zhu, Y., Freeman, I., Machado, J. V., Ahmed, A., Sarker, A., & Al-Garadi, M. (2025). Agentic AI in Healthcare: A Comprehensive Survey of Foundations, Taxonomy, and Applications. *Authorea Preprints*.

[19] Abou Ali, M., Dornaika, F., & Charafeddine, J. (2025). Agentic AI: a comprehensive survey of architectures, applications, and future directions. *Artificial Intelligence Review*, *59*(1), 11.

[20] Giovanni Lucchiari Hartz, E. (2025). Development of an AI-driven DevOps Engineer: Automating Workflows with an LLM based Multi-Agent System.

[21] Kamalov, F., Calonge, D. S., Smail, L., Azizov, D., Thadani, D. R., Kwong, T., & Atif, A. (2025). Evolution of ai in education: Agentic workflows. *arXiv preprint arXiv:2504.20082*.

[22] Sarferaz, S. (2025). Implementing Agentic AI Into ERP Software. *IEEE Access*.

[23] Becker, E. C. (2021). Real-Time AI-Driven Software Development: Hybrid Fuzzy WPM and TOPSIS Integration with Deep Learning and Particle Swarm Optimization in Agentic Negotiation Frameworks. *International Journal of Computer Technology and Electronics Communication*, *4*(5), 4018-4023.

[24] Agarwal, S., He, H., & Vasilescu, B. (2026). AI IDEs or Autonomous Agents? Measuring the Impact of Coding Agents on Software Development. *arXiv preprint arXiv:2601.13597*.

[25] Ali, Z., & Grudén, K. (2025). Intelligent multi-agent framework for automated data analytics and machine learning tasks (AI agent).

[26] Mishra, G. (2024). Autonomous AI Agents for Enterprise Workflow Orchestration in HR Platforms. *International Journal of Research and Applied Innovations*, *7*(6), 11873-11887.

[27] Hassan, A. E., Oliva, G. A., Lin, D., Chen, B., & Jiang, Z. M. (2024). Towards AI-native software engineering (SE 3.0): A vision and a challenge roadmap. *ACM Transactions on Software Engineering and Methodology*.

[28] Kong, X., Xing, Y., Tsourdos, A., Wang, Z., Guo, W., Perrusquia, A., & Wikander, A. (2024). Explainable interface for human-autonomy teaming: A survey. *arXiv preprint arXiv:2405.02583*.

[29] Garousi, V., Joy, N., Jafarov, Z., Keleş, A. B., Değirmenci, S., Özdemir, E., & Zarringhalami, R. (2024). AI-powered software testing tools: A systematic review and empirical assessment of their features and limitations. *arXiv preprint arXiv:2409.00411*.

[30] Kakarlapudi, R. V., & Yaramchitti, J. K. (2024). Agentic AI for Autonomous Telecom Network Management. *International Journal of Emerging Research in Engineering and Technology*, *5*(3), 129-135.