



ast46

TUNISIAN REPUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and
Technology



Graduation Project
In order to obtain the
National Engineering Diploma
Specialty : Software Engineering

**Enforcing Best Practices through LLM Integrations in
Google's internal IDE**

Presented by
Arij KOUKI

Hosted by



INSAT Supervisor : Ms. YOUSSEF Rabaa
Company Supervisor : Ms. LOPEZ Irene

Presented on : --/--/2025

JURY

M. President FLEN (President)
Ms. Reviewer FLENA (Reviewer)

Academic Year : 2024/2025



TUNISIAN REPUBLIC
Ministry of Higher Education and Scientific Research
University of Carthage
National Institute of Applied Sciences and
Technology



Graduation Project
In order to obtain the
National Engineering Diploma
Specialty : **Software Engineering**

Enforcing Best Practices through LLM Integrations in Google's internal IDE

Presented by
Arij KOUKI
Hosted by



Company Supervisor	University Supervisor

Academic Year : 2024/2025

Acknowledgements

I wish to extend my deepest gratitude and appreciation to everyone who has contributed significantly to the successful completion of this project.

My sincere thanks go to Irene Lopez and Andrew Xue, my host and co-host at Google Zurich, for their invaluable guidance, support, and trust throughout my internship. Their mentorship and kindness made this experience not only enriching but truly transformative.

Thank you to the YouTube Developer Infrastructure team for their warm welcome, collaborative spirit, and continuous encouragement, and to my mentor Veronica Radu, whose insights and support greatly contributed to my personal and professional growth.

I also wish to thank Mrs. Rabaa Youssef, my academic supervisor, for accompanying me in this final academic milestone.

To the distinguished members of the jury, I am grateful for your time and consideration in reviewing my work. I hope this report lives up to the standards expected of a graduation project.

To the professors of the National Institute of Applied Sciences and Technology, thank you for playing a vital role in shaping my academic and professional foundation.

And to my family and friends, your unwavering belief in me has carried me through the most demanding moments. Thank you for your love, patience, and presence.

Finally, a quiet note of gratitude to myself for the perseverance and dedication that made this journey possible.

Table of Contents

List of Figures	iv
List of Tables	v
Résumé	vi
Abstract	vii
General Introduction	1
I Project Overview	3
1 Host Company: Google	4
1.1 Presentation	4
1.2 Products and services	4
1.3 Focus Area	5
1.3.1 YouTube	5
1.3.2 YouTube Developer Infrastructure Team	5
2 Project Overview	5
2.1 Project Context	5
2.2 Existing Solutions	6
2.3 Problem Statement	6
2.4 Proposed Solution	7
3 Work Methodology	7
3.1 Agile Development Approach	7
3.2 Kanban Workflow	7
3.3 Development Process	8
3.4 Project Timeline	10
II Business Understanding and Comparative Analysis	14
1 Business and Reasoning	15
1.1 Software Development Life Cycle (SDLC)	15
1.2 Artificial Intelligence in Software Development	20
2 State of the Art: Existing Solutions	26
3 Project Requirements	29

3.1	Functional Requirements	29
3.2	Non-Functional Requirements	30
III Implementation		32
1	Outils et langages utilisés	32
2	Présentation de l'application	32
2.1	Exemple de tableau	33
2.2	Exemple de Code	33
3	Remarques sur la bibliographie	34
Conclusion and Perspectives		36
References		37
Appendix : Miscellaneous remarks		38

List of Figures

I.1	Google Logo	4
I.2	Overview of some Google products	4
I.3	YouTube Logo	5
I.4	Kanban Workflow	8
I.5	Project Development Iteration Cycle	10
I.6	Project Timeline - Detailed Schedule	11
II.1	Software Development Life Cycle Overview	15
II.2	Waterfall SDLC Model	17
II.3	V-Model SDLC	17
II.4	Iterative and Incremental SDLC Model	18
II.5	Agile SDLC Model	19
II.6	DevOps Cycle Integration with SDLC	19
II.7	Hierarchy of AI Technologies	20
II.8	A chronological overview of large language models (LLMs) [2018-2024]	22
II.9	AI Agent Architecture and Orchestration Workflow	23
II.10	Current Developer Workflow Feedback Timeline	27

List of Tables

II.1	Comparison of available software quality support solutions	28
II.2	Summary of project requirements	30
III.1	Tableau comparatif	33

Résumé

Ce projet a été réalisé chez Google Zurich dans le cadre d'un Diplôme National d'Ingénieur en Génie Logiciel. Il explore l'intégration de l'intelligence artificielle générative dans le processus de développement logiciel en incorporant un agent basé sur un Large Language Model (LLM) au sein d'un Environnement de Développement Intégré (IDE) interne.

Cet agent effectue une analyse approfondie du code afin de détecter des violations complexes ou subjectives que les outils d'analyse statique traditionnels peuvent négliger. En générant des explications claires et compréhensibles ainsi que des suggestions concrètes, il aide les développeurs, notamment ceux contribuant à YouTube, à maintenir une haute qualité de code et à respecter les bonnes pratiques. Intégré de manière transparente au sein du flux de travail via une extension de l'IDE, l'agent améliore la productivité et contribue à la réduction de la dette technique sans perturber l'expérience de développement.

Ce travail met en évidence le potentiel des outils assistés par l'IA pour transformer l'expérience des développeurs et ouvre des perspectives pour l'avenir des environnements de développement intelligents.

Mots-clés : Intelligence Artificielle Générative, Génie Logiciel, Intégration IDE, Qualité du Code, Productivité des Développeurs

Abstract

This project was carried out at Google Zurich as part of a National Engineering Diploma in Software Engineering. It investigates the integration of generative artificial intelligence into the software development process by embedding a Large Language Model (LLM)-powered agent within an internal Integrated Development Environment (IDE).

The agent performs in-depth code analysis to detect complex or subjective violations that traditional static analysis tools may overlook. By generating clear, human-readable explanations and actionable suggestions, it supports developers, particularly those contributing to YouTube, in maintaining high code quality and adhering to best practices. Seamlessly integrated into the development workflow through an IDE extension, the agent enhances productivity and helps reduce technical debt without disrupting the coding experience.

This work demonstrates the potential of AI-assisted tooling to transform the developer experience and raises broader implications for the future of intelligent software engineering environments.

Keywords: Generative AI, Software Engineering, IDE Integration, Code Quality, Developer Productivity

General Introduction

The software engineering industry is experiencing a major shift driven by the rapid evolution of artificial intelligence and the growing demand for scalable, high-quality code development practices. As development teams grow and systems become more complex, ensuring consistent code quality and adherence to best practices presents an ongoing challenge, especially in large organizations managing massive codebases across distributed teams.

Traditional static analysis tools and linters, while helpful, often fall short when it comes to identifying nuanced or subjective coding issues that depend on context or internal guidelines. In fast-paced environments like those of YouTube engineering teams, developers need intelligent, responsive tools that go beyond rule-based checks to provide deeper insights and real-time guidance, without adding friction to their daily workflows.

This graduation project, conducted at Google Zurich, explores the integration of generative artificial intelligence into an internal Integrated Development Environment (IDE) to support software engineers in their day-to-day coding activities. It introduces a Large Language Model (LLM)-powered agent capable of performing in-depth code analysis, detecting complex violations, and offering clear, contextual suggestions for improvement. Embedded directly within the IDE through an extension, this intelligent assistant aims to enhance code quality, reduce technical debt, and support developer productivity at scale.

This report begins by presenting the context and challenges of applying AI in modern software development. It then outlines the design and integration of the AI agent, followed by an evaluation of its contribution to improving engineering workflows within a high-impact, real-world environment.

Part I

Project Foundation and Methodology

Chapter I

Project Overview

Summary

1	Host Company: Google	4
1.1	Presentation	4
1.2	Products and services	4
1.3	Focus Area	5
2	Project Overview	5
2.1	Project Context	5
2.2	Existing Solutions	6
2.3	Problem Statement	6
2.4	Proposed Solution	7
3	Work Methodology	7
3.1	Agile Development Approach	7
3.2	Kanban Workflow	7
3.3	Development Process	8
3.4	Project Timeline	10

Introduction

This opening chapter establishes the foundation of the graduation project by introducing the host company and defining the project's scope. We examine the organizational context, outline the main objectives and challenges, and present the methodological framework that guides the development process.

1 Host Company: Google

1.1 Presentation

Founded in 1998, Google LLC is a global leader in technology and innovation. As a subsidiary of Alphabet Inc., Google's mission is to organize the world's information and make it universally accessible and useful. Guided by values such as innovation, accessibility, sustainability, and user trust, Google has established itself as one of the most influential companies shaping the digital era. Its culture emphasizes collaboration, diversity, inclusion, and impact-driven engineering, enabling continuous leadership in research and product development.



Figure I.1 – Google Logo

1.2 Products and services

Google offers a broad ecosystem of products and services that touch nearly every aspect of digital life. Among its flagship consumer products are Google Search, Maps, Gmail, Chrome, and the Android operating system, serving billions of users daily.

Beyond consumer services, Google develops enterprise and cloud-based solutions such as Google Cloud Platform and Google Workspace, as well as advanced AI systems like Vertex AI. The company also invests in hardware, including Pixel devices, Nest smart home products, and ChromeOS.

These products reflect Google's commitment to connecting people, improving productivity, and driving digital transformation worldwide.



Figure I.2 – Overview of some Google products

1.3 Focus Area

1.3.1 YouTube

Acquired by Google in 2006, YouTube has become the world's leading video-sharing platform, serving more than two billion logged-in users monthly. It empowers individuals to create, share, and discover video content globally while sustaining a vibrant creator economy. From a technical perspective, YouTube integrates video processing, recommendation systems, live streaming, advertising, and trust and safety to deliver a seamless experience across devices.



Figure I.3 – YouTube Logo

1.3.2 YouTube Developer Infrastructure Team

Within YouTube's engineering organization, the Developer Infrastructure (Dev Infra) team supports thousands of engineers building the platform. The team develops tooling, automation, and guidelines that improve efficiency, reliability, and consistency in software development. By maintaining developer velocity and quality at scale, the Dev Infra team contributes directly to YouTube's ability to innovate and grow.

2 Project Overview

2.1 Project Context

This project was developed within the scope of my host team YouTube Dev Infra, which focuses on supporting developers by providing tools and extensions that improve their day-to-day workflows. As part of this mission, the team is exploring how artificial intelligence can be leveraged to further assist developers. One concrete idea is to enhance their existing extension by introducing a feature that helps enforce internal best practices. The goal is to explore how large language models (LLMs) can complement traditional approaches, offering developers more intelligent and context-aware guidance directly within the IDE.

In addition, this project is carried out as part of the National Institute of Applied Science and Technology's fifth-year mandatory final project, which is required to obtain the software

engineering degree.

2.2 Existing Solutions

To support best practices and maintain code quality, teams currently rely on three main approaches:

- **Code Reviews:** Engineers provide human feedback on design, readability, and best practices during review sessions. Recently, AI-assisted reviews have also been introduced, helping both reviewers and developers by suggesting improvements.
- **Presubmit Checks:** Automated checks executed before code is submitted. These ensure correctness, style consistency, and prevent simple errors.
- **Rule-Based Checks (Work in Progress):** Tools under development to enforce simple, objective rules such as naming conventions or syntax. These checks are intended for measurable guidelines and provide consistent, automated verification.

Together, these mechanisms form the current framework supporting developers in maintaining quality and consistency across the codebase.

2.3 Problem Statement

While these approaches provide valuable support, they also leave significant gaps in practice.

- **Delayed Feedback:** The most meaningful guidance on design and best practices typically arrives during code reviews, after development is complete. This often requires rework and slows iteration.
- **Limited Scope of Presubmit Checks:** Presubmits focus on correctness and safety rather than nuanced best practices, leaving developers without proactive guidance in those areas.
- **Surface-Level Coverage of Rule-Based Checks:** Rule-based tools can only enforce simple, objective rules. They are not able to reason about context-dependent or subjective best practices.

As a result, developers lack timely, intelligent support during the actual coding phase, where guidance would be most efficient and impactful.

2.4 Proposed Solution

This project introduces an **AI-assisted feedback system integrated directly into the coding workflow**.

- **Real-Time Guidance:** Provide developers with immediate, context-aware suggestions while they are writing code.
- **Framework-Specific Best Practice Enforcement:** Go beyond syntax and correctness by surfacing adherence to YouTube’s internal framework guidelines early in the development process.
- **Reduced Review Burden:** Shift part of the best practice enforcement from manual reviews to the authoring stage, making reviews faster and more focused on higher-level insights.

By bringing intelligent, framework-aware feedback closer to the point of coding, the solution aims to reduce back-and-forth during reviews, ensure internal consistency, and accelerate development velocity.

3 Work Methodology

3.1 Agile Development Approach

Agile practices were adopted to support iterative development and maintain flexibility in responding to evolving requirements. This approach allowed continuous integration of feedback from the host and co-host, ensuring that each increment of work aligned with both technical goals and the broader product vision. Testing, validation, and code reviews were incorporated throughout the process to maintain high quality, while frequent collaboration provided clarity and shared ownership of outcomes. Agile principles also complemented Google’s focus on engineering excellence, including rigorous design reviews, thorough testing, robust code reviews, and DevOps-enabled automation.

3.2 Kanban Workflow

The dynamic workload of the YouTube Developer Infrastructure (Dev Infra) team, including feature requests, bug fixes, and maintenance tasks, was managed using a Kanban workflow. By visualizing tasks and limiting work in progress, the team could prevent bottlenecks and quickly

shift focus to urgent issues when necessary. Work was structured into stages to maintain clear coordination while allowing the flexibility to adapt priorities as requirements evolved.

The Kanban workflow included the following stages:

- **Backlog:** Prioritized collection of feature requests, enhancements, and bug fixes.
- **Research & Design:** Assessment of technical feasibility and preparation of design specifications.
- **Development:** Implementation and integration of features into the system.
- **Review & Testing:** Code review, unit tests, and integration tests to ensure quality and correctness.
- **Deployment:** Release of validated features to developer environments.

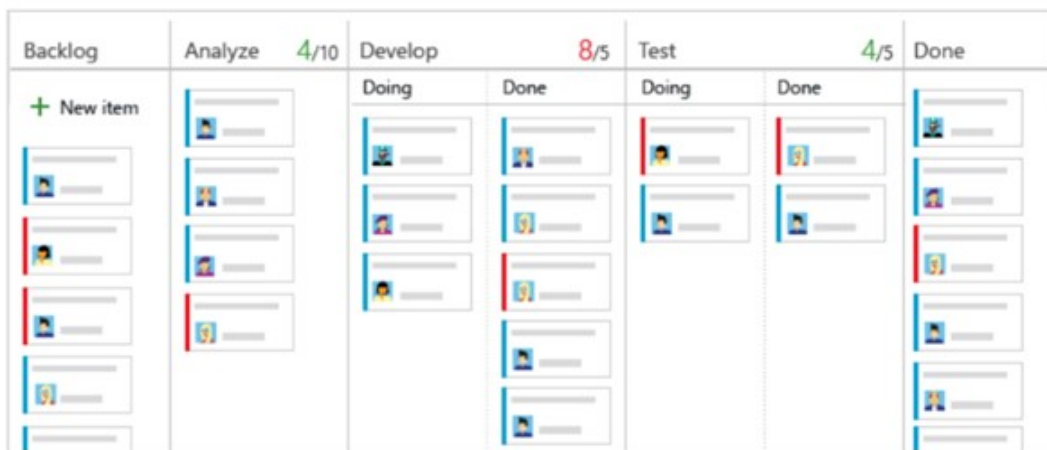


Figure I.4 – Kanban Workflow

3.3 Development Process

The project followed an iterative engineering cycle designed to balance thorough planning with incremental delivery. Each stage was supported by structured practices, dedicated tools, and regular collaboration:

- **Ideation and Research:** The project began with an exploration phase to clarify objectives, gather requirements, and investigate potential solution directions. This stage combined independent research with collaborative discussions to assess feasibility and align on priorities.

- **Design and Planning:** A detailed design document was authored to present the technical choices, architectural considerations, and proposed workflow. This document was reviewed by engineers, iteratively refined, and approved. The final work plan was then transferred to the internal task management system, enabling structured tracking and prioritization.
- **Implementation:** Development was performed in small, reviewable increments using Google’s internal development environment within the company-wide repository. Each change was submitted accompanied by unit tests, and validated through both manual and AI-assisted code reviews. The implementation combined Python for backend logic, RPCs for inter-service communication, and TypeScript for the frontend.
- **Testing and Validation:** Functionality and reliability were verified continuously. Automated unit tests ensured correctness at the component level, while integration reviews validated the behavior within the larger system.
- **Maintenance and Optimization:** Refactoring, bug fixes, and updates were performed throughout development, particularly as some dependencies evolved or methods became deprecated. This ensured that the solution remained consistent, maintainable, and aligned with evolving standards.

Collaboration was supported through a structured communication rhythm, combining regular syncs with the host and co-host, weekly team meetings, and occasional cross-team discussions. This cadence provided timely feedback, clear guidance, and alignment on shared dependencies.

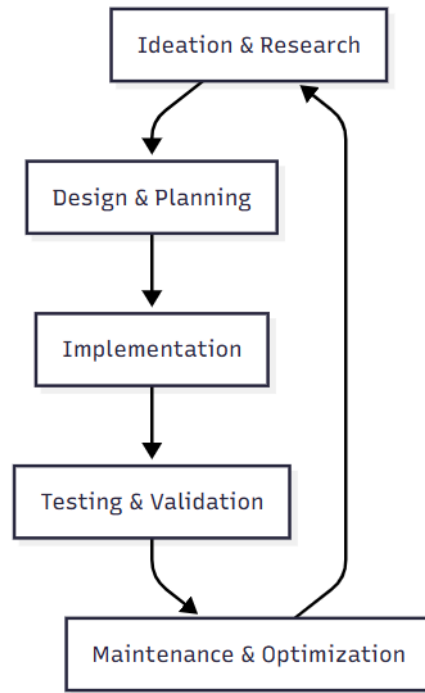


Figure I.5 – Project Development Iteration Cycle

3.4 Project Timeline

The project spanned four months (May 5 – September 5, 2025). Work was scheduled based on business priorities and technical dependencies. Early weeks focused on research and design, followed by implementation, testing, and iterative refinement.

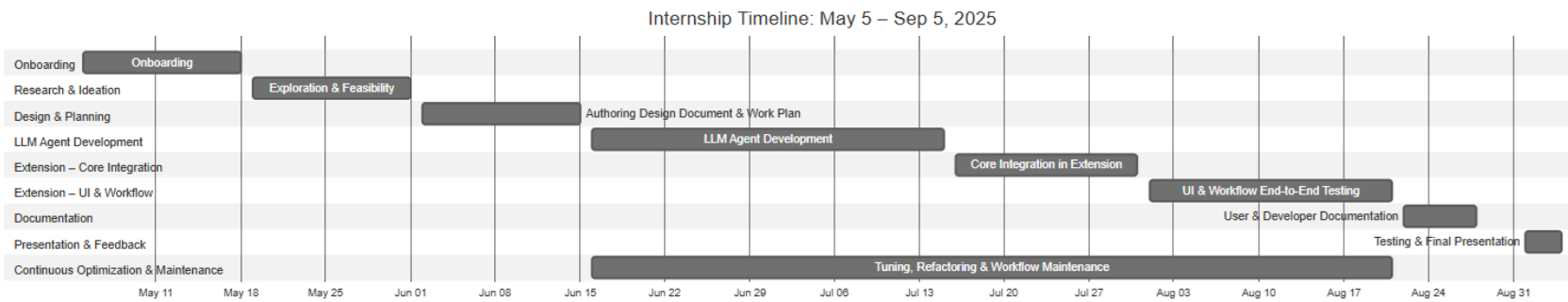


Figure I.6 – Project Timeline - Detailed Schedule

Conclusion

In summary, this chapter outlined the project's context by presenting the host company, defining the problem, and clarifying the main objectives and challenges. It also described the methodology chosen to guide the work, which provides the basis for the technical developments detailed in the following chapters.

Part II

Partie 2

Chapter II

Business Understanding and Comparative Analysis

Summary

1	Business and Reasoning	15
1.1	Software Development Life Cycle (SDLC)	15
1.2	Artificial Intelligence in Software Development	20
2	State of the Art: Existing Solutions	26
3	Project Requirements	29
3.1	Functional Requirements	29
3.2	Non-Functional Requirements	30

Introduction

This chapter establishes the theoretical and contextual foundation of the project. It begins with an overview of the software development lifecycle (SDLC) and the role of emerging AI technologies such as Large Language Models (LLMs), Generative AI, and AI agents in modern software engineering. It then presents the state of the art by reviewing existing solutions currently supporting code quality and best practices. Finally, it defines the project requirements, both functional and non-functional, showing how this work addresses existing gaps by integrating AI-driven assistance into a key phase of the SDLC.

1 Business and Reasoning

1.1 Software Development Life Cycle (SDLC)

The Software Development Life Cycle (SDLC) is a structured framework that defines the process of planning, creating, testing, deploying, and maintaining software systems. It ensures that software development follows a disciplined approach, promoting quality, efficiency, and maintainability while reducing risks and development costs. The SDLC helps teams manage complex projects by providing a roadmap from initial requirements to system retirement.

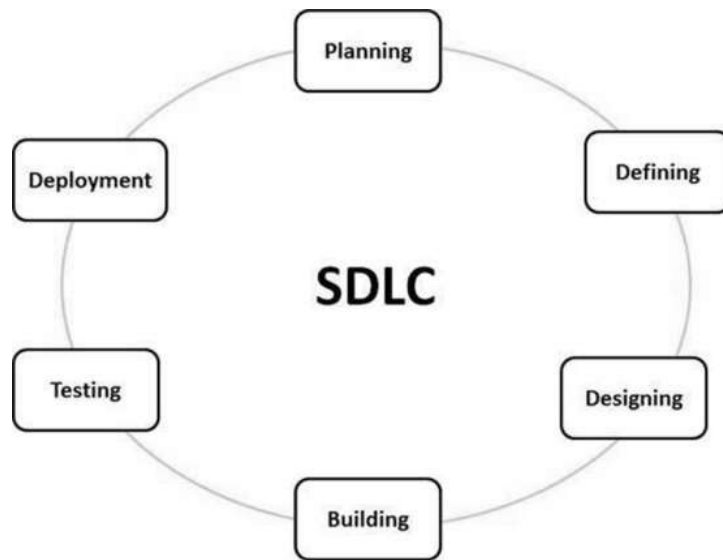


Figure II.1 – Software Development Life Cycle Overview

Key Steps in SDLC

A typical SDLC consists of seven core phases, each serving a distinct purpose in ensuring that software meets functional and quality requirements:

1. **Planning:** This initial phase defines the project scope, objectives, resources, and timeline. Risk assessment, feasibility studies, and cost-benefit analyses are performed to ensure the project is viable before investing substantial resources.
2. **Requirements Analysis:** Stakeholders, including end-users and business analysts, collaborate to specify functional and non-functional requirements. Clear, documented requirements reduce ambiguity and serve as a reference for design and implementation. Techniques such as use cases, user stories, and requirement specifications are commonly used.

3. **System Design:** Architects and developers define the software architecture, data models, interface designs, and technology stack. This phase produces detailed design documents and prototypes that guide the implementation phase.
4. **Implementation (Coding):** Developers translate design specifications into source code using appropriate programming languages, frameworks, and tools. Best practices such as modularization, code reviews, and version control are critical in this phase.
5. **Testing:** The system undergoes rigorous verification to detect and correct defects. Testing can include unit testing, integration testing, system testing, performance testing, and user acceptance testing (UAT). Automated testing frameworks are increasingly used to ensure efficiency and repeatability.
6. **Deployment:** Once verified, the software is deployed to production environments. Deployment strategies may include phased rollouts, blue-green deployments, or canary releases to minimize risk and disruption.
7. **Maintenance:** After deployment, the software requires ongoing updates, bug fixes, performance improvements, and adaptations to changing requirements or environments. Maintenance ensures long-term system reliability and user satisfaction.

Common SDLC Models

Various SDLC models provide different approaches to organizing these steps. Each model offers specific advantages and trade-offs, depending on project complexity, team structure, and business goals.

Waterfall Model The Waterfall model is a linear and sequential approach where each phase must be completed before the next begins. Its structure ensures strict documentation and clear progress milestones.

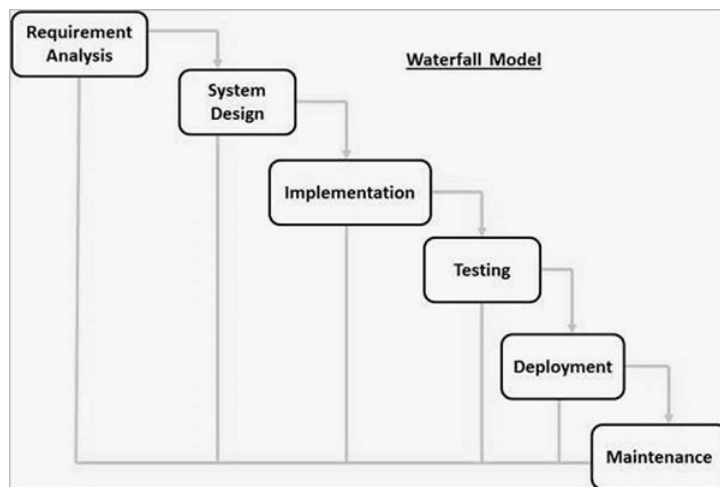


Figure II.2 – Waterfall SDLC Model

Advantages: Simple and easy to manage, clear documentation, predictable timelines.

Limitations: Inflexible to changing requirements, late discovery of defects, high risk in complex projects.

V-Model (Verification and Validation) The V-Model extends Waterfall by emphasizing verification and validation. Each development phase has a corresponding testing phase to ensure quality at every stage.

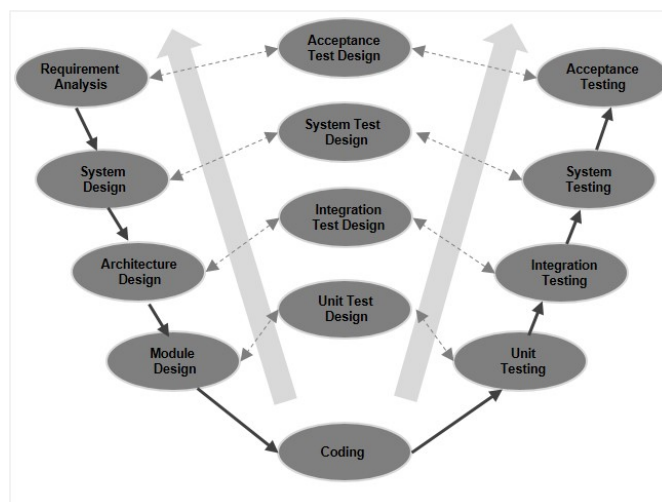


Figure II.3 – V-Model SDLC

Advantages: Strong focus on testing, early defect detection.

Limitations: Like Waterfall, less flexible with changing requirements.

Iterative and Incremental Model This approach develops software in small iterations, delivering functional increments at each cycle. Feedback from early releases informs subsequent iterations.

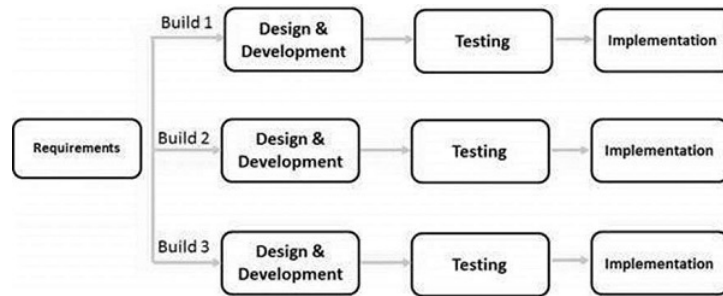


Figure II.4 – Iterative and Incremental SDLC Model

Advantages: Flexibility to change, early delivery of usable software, reduced risk.

Limitations: Requires careful planning, can increase complexity if iterations are poorly managed.

Agile Model Agile emphasizes collaboration, adaptability, and rapid delivery. Work is organized into sprints, typically 2-4 weeks, producing a potentially shippable product increment. Agile can be implemented through frameworks such as:

- **Scrum:** Defines roles (Product Owner, Scrum Master, Development Team) and ceremonies (Daily Stand-ups, Sprint Planning, Retrospectives) to organize work in sprints.
- **Kanban:** Visualizes workflow and limits work in progress, emphasizing continuous delivery and efficiency rather than fixed-length iterations.

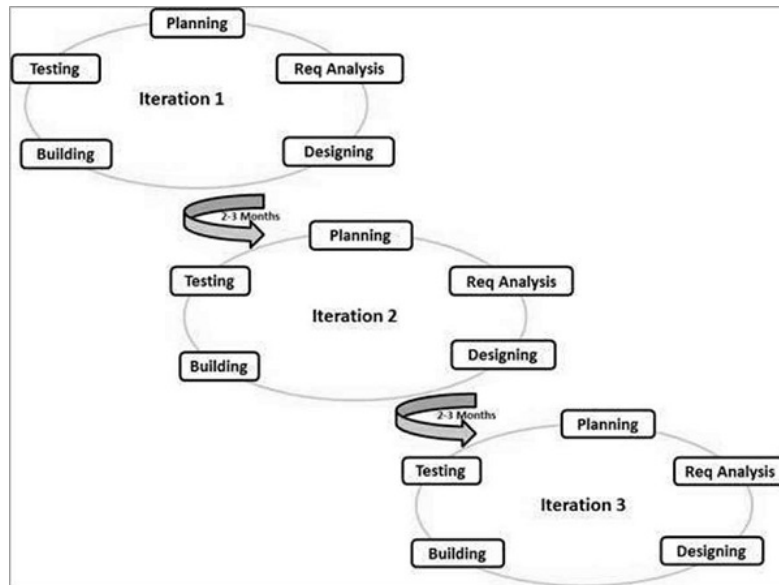


Figure II.5 – Agile SDLC Model

Advantages: Highly flexible, promotes continuous feedback, encourages customer involvement.

Limitations: Less predictability in timelines, requires disciplined team practices.

DevOps DevOps is a cultural and technical approach that integrates development and operations teams, promoting continuous integration, continuous delivery (CI/CD), automated testing, and real-time monitoring. By combining SDLC phases with DevOps practices, organizations achieve faster delivery, higher reliability, and improved collaboration between teams.

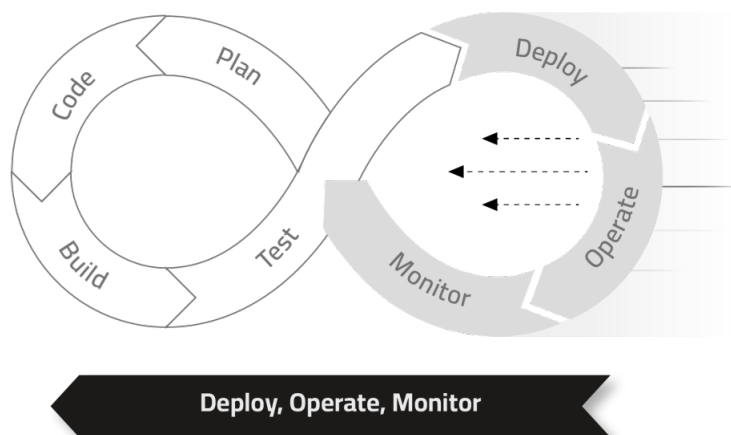


Figure II.6 – DevOps Cycle Integration with SDLC

Advantages: Accelerates delivery, improves collaboration, enables automated testing and

monitoring, enhances reliability.

Limitations: Requires organizational culture change, relies on mature tooling, can be complex to implement in large teams.

Emerging Enhancements Modern software development also increasingly relies on AI-powered tools that provide predictive analytics, code suggestions, and real-time quality checks. These enhancements further increase SDLC efficiency and reliability, setting the stage for intelligent, AI-driven development assistance.

1.2 Artificial Intelligence in Software Development

Foundational AI Concepts

Artificial Intelligence (AI) refers to computational systems capable of performing tasks that typically require human intelligence, such as reasoning, learning, problem-solving, and decision-making. AI encompasses a broad spectrum of techniques and paradigms, each with distinct capabilities and applications.

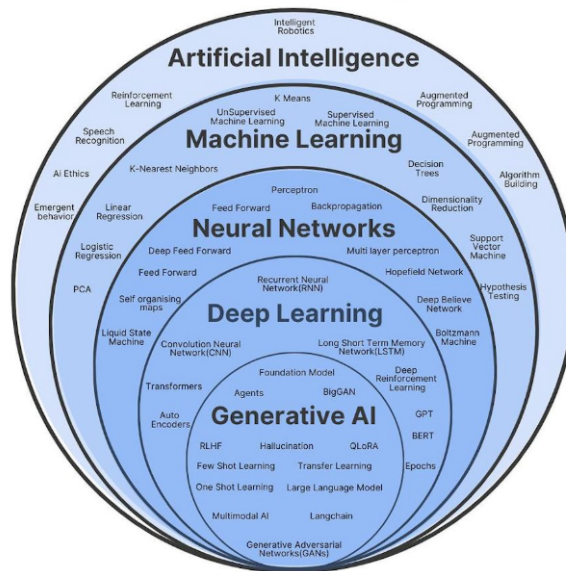


Figure II.7 – Hierarchy of AI Technologies

The AI landscape includes several key categories:

- **Symbolic AI:** Systems based on rules and logic to represent knowledge and perform reasoning. These systems excel at tasks with well-defined rules but struggle with ambiguity and complex pattern recognition.

- **Machine Learning (ML):** Algorithms that improve performance on tasks by learning from data rather than being explicitly programmed. ML systems can identify patterns and make predictions based on historical data.
- **Deep Learning (DL):** A subset of ML using artificial neural networks with multiple layers to model complex patterns in data. Deep learning has revolutionized fields like computer vision and natural language processing.
- **Generative AI:** AI systems capable of producing new content, such as text, code, or images, by learning patterns from existing datasets. Unlike traditional AI that classifies or predicts, generative AI creates novel outputs.

The AI Revolution in Software Engineering

As software development becomes increasingly complex and teams face mounting pressure to deliver high-quality code faster, traditional development approaches are reaching their limits. The emergence of Artificial Intelligence technologies presents a transformative opportunity to address these challenges by bringing intelligent assistance directly into the development workflow.

The integration of AI into software development represents more than just technological advancement—it addresses fundamental business challenges that organizations face in maintaining code quality, reducing technical debt, and scaling development practices across growing teams. This represents a paradigm shift from traditional SDLC approaches to what we can call the AI-Enhanced SDLC—a development lifecycle where intelligent assistance is embedded throughout every phase, creating a more efficient, proactive, and intelligent development process.

This transformation is primarily driven by two key AI technologies: Large Language Models (LLMs) and AI agents, which together enable the intelligent development workflows that characterize the modern AI-enhanced SDLC.

Large Language Models: The Foundation of Intelligent Development

Large Language Models (LLMs) represent a breakthrough in generative AI technology, trained on massive corpora of natural language and source code. Unlike traditional programming tools that rely on rigid rules and patterns, LLMs understand context, semantics, and intent, making them uniquely suited for complex reasoning tasks across various domains.

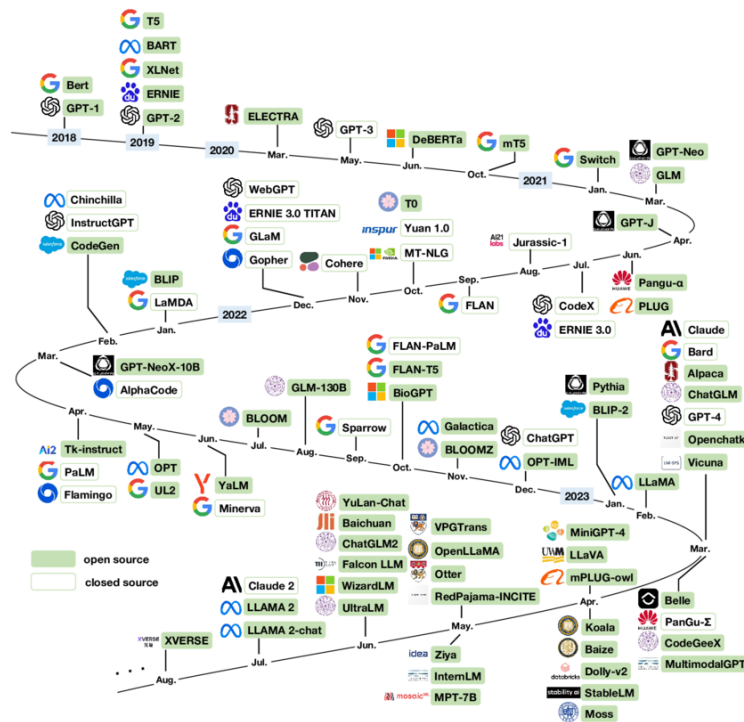


Figure II.8 – A chronological overview of large language models (LLMs) [2018-2024]

The fundamental capabilities of LLMs include:

- **Natural Language Understanding:** Ability to comprehend and interpret human language with nuanced understanding of context, intent, and meaning.
- **Pattern Recognition:** Capacity to identify complex patterns in data, including code structures, design patterns, and domain-specific conventions.
- **Content Generation:** Capability to produce coherent, contextually appropriate text, code, and other content based on learned patterns.
- **Reasoning and Inference:** Ability to perform logical reasoning, make inferences, and solve complex problems through step-by-step analysis.

In the context of software development, these capabilities translate into powerful applications:

- **Contextual Code Analysis:** Understanding code within its broader project context, including relationships between components, dependencies, and business logic.

- **Intelligent Code Generation:** Producing code suggestions, refactoring recommendations, and automated fixes that align with project-specific patterns and requirements.
- **Explanatory Documentation:** Providing clear, human-readable explanations of code behavior, potential issues, and suggested improvements.
- **Semantic Standards Enforcement:** Detecting violations of coding standards and best practices that go beyond simple syntax checking to include design patterns, architectural principles, and domain-specific guidelines.

AI Agents: Orchestrating Intelligent Development Workflows

While LLMs provide the foundational intelligence for understanding and generating code, AI agents represent the next evolution—autonomous systems that can reason, plan, and execute complex development tasks. AI agents serve as intelligent orchestrators that bridge the gap between human developers and AI capabilities, providing seamless integration of AI assistance into the development workflow.

Agent Architecture and Workflow The architecture of AI agents follows a structured workflow that enables reliable and scalable operation. As illustrated in Figure II.9, the agent orchestration process begins with input processing, where user requests are captured and pre-processed. The core reasoning component then analyzes the input using LLM capabilities to determine the appropriate course of action. This is followed by tool integration, where the agent interfaces with external systems to execute planned actions. Throughout this process, the memory layer maintains context and stores relevant information for future reference.

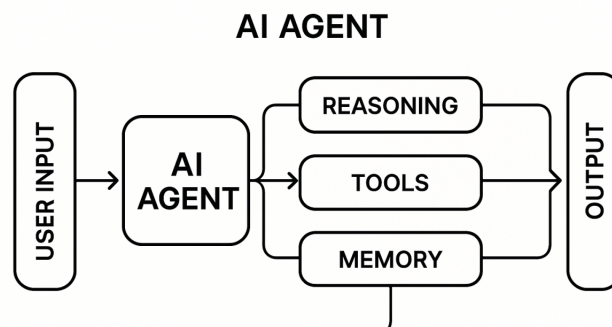


Figure II.9 – AI Agent Architecture and Orchestration Workflow

Core Components of Development-Focused AI Agents AI agents designed for software development incorporate specialized components tailored to the development workflow:

- **Code Analysis Engine:** Processes source code, understands project context, and identifies violations of coding standards, design patterns, and best practices.
- **Contextual Reasoning:** Utilizes LLMs to understand the broader project context, including dependencies, business logic, and architectural constraints when making recommendations.
- **Development Tool Integration:** Interfaces with IDEs, version control systems, testing frameworks, and other development tools to provide seamless assistance within the existing workflow.
- **Learning and Adaptation:** Continuously learns from codebase patterns, team feedback, and project outcomes to provide increasingly relevant and accurate assistance.
- **Feedback and Explanation System:** Provides clear, actionable explanations for identified issues and suggested improvements, helping developers understand not just what to change, but why.

Balancing AI Capabilities with Practical Considerations While AI agents offer transformative potential for software development, their deployment requires careful consideration of both capabilities and limitations:

Key Capabilities:

- **Contextual Understanding:** Ability to analyze code within its broader project context, understanding relationships and dependencies
- **Automated Analysis:** Consistent, comprehensive code analysis that scales across large codebases and development teams
- **Intelligent Recommendations:** Context-aware suggestions that go beyond simple rule-based checks to include design patterns and architectural considerations
- **Continuous Learning:** Adaptation to project-specific patterns and team preferences over time

Practical Limitations:

- **Computational Cost:** LLM-based analysis requires significant computational resources, impacting response times and operational costs
- **Context Constraints:** Limited ability to process extremely large files or maintain context across very long code sequences
- **Accuracy Considerations:** Potential for misinterpretation or hallucination, requiring human oversight for critical decisions
- **Integration Complexity:** Requires careful integration with existing development tools and workflows

These considerations inform the design of practical AI-assisted development systems that maximize benefits while managing limitations effectively.

AI-Enhanced SDLC: The Complete Integration

The combination of LLMs and AI agents enables the complete integration of intelligent assistance throughout the software development lifecycle. This AI-enhanced SDLC represents a fundamental transformation from traditional reactive approaches to proactive, intelligent development assistance that addresses the core business challenges identified earlier.

The Transformative Impact of AI Integration The integration of AI into the SDLC addresses key limitations of conventional development practices:

- **From Reactive to Proactive:** Traditional approaches identify issues after they occur, often during code review or testing phases. AI-enhanced development provides real-time feedback during coding, preventing issues before they become embedded in the codebase.
- **From Inconsistent to Scalable:** Human-based quality assurance varies in consistency and availability. AI provides uniform, expert-level analysis that scales across teams and projects without degradation.
- **From Static to Adaptive:** Traditional tools rely on fixed rules and patterns. AI systems learn and adapt to project-specific patterns, team preferences, and evolving best practices.
- **From Isolated to Integrated:** Rather than treating quality assurance as a separate phase, AI integrates intelligent assistance throughout the entire development workflow.

AI Applications Across the Development Lifecycle AI technologies provide targeted assistance at each phase of the SDLC, addressing specific challenges and opportunities:

- **Requirements and Planning:** AI analyzes historical project data to predict timelines, identify requirement ambiguities, and suggest optimal resource allocation. Natural language processing helps translate business requirements into technical specifications.
- **Design and Architecture:** AI assists architects by analyzing existing codebases, suggesting architectural patterns, detecting design anti-patterns, and ensuring consistency with organizational standards.
- **Implementation:** This is where AI agents provide the most immediate value, offering real-time code completion, best practice enforcement, bug detection, and refactoring suggestions directly within the development environment.
- **Testing and Quality Assurance:** AI generates comprehensive test cases, prioritizes test execution based on risk analysis, and identifies edge cases that human testers might miss.
- **Deployment and Maintenance:** AI monitors deployment health, predicts potential issues, and automatically suggests optimizations based on usage patterns and performance metrics.

This comprehensive understanding of AI's potential in software development provides the foundation for evaluating current solutions and identifying opportunities for improvement. The next section examines existing approaches to code quality and best practices enforcement, highlighting both their contributions and limitations in addressing the business challenges outlined above.

2 State of the Art: Existing Solutions

While the AI-enhanced SDLC presents a vision of intelligent, proactive development assistance, the current reality in most development environments, including our workspace, relies on more traditional approaches to code quality and best practices enforcement. Software engineers currently depend on a combination of human review, automated scripts, and rule-based systems to maintain code quality. These mechanisms vary in scope, effectiveness, and feedback timing, but as our analysis will show, they leave significant gaps in providing the real-time, intelligent assistance that characterizes the AI-enhanced development paradigm.

II.2 State of the Art: Existing Solutions

Current Feedback Mechanisms The existing approaches to code quality and best practices enforcement can be categorized into several key mechanisms:

- **Code Reviews:** Human reviewers examine code for design quality, readability, maintainability, and adherence to standards. While this approach provides high-level, context-aware feedback, it often introduces delays and requires significant effort. Recently, AI-assisted review tools have been developed to accelerate this process by providing initial suggestions or flagging common issues.
- **Presubmit Checks:** Automated scripts that validate code before submission, enforcing style guides, ensuring compilation, and verifying simple correctness and safety constraints. Although fast and reliable, they primarily focus on surface-level checks and do not consider design or contextual issues.
- **Rule-Based Checks:** Systems that enforce coding conventions, naming schemes, and formatting standards. These tools provide consistency and objectivity but cannot reason about complex or context-dependent best practices.

Limitations While valuable, these approaches leave significant gaps, particularly in providing timely and intelligent feedback during the active coding phase. This timeline illustrates where current feedback mechanisms fit into the developer workflow:

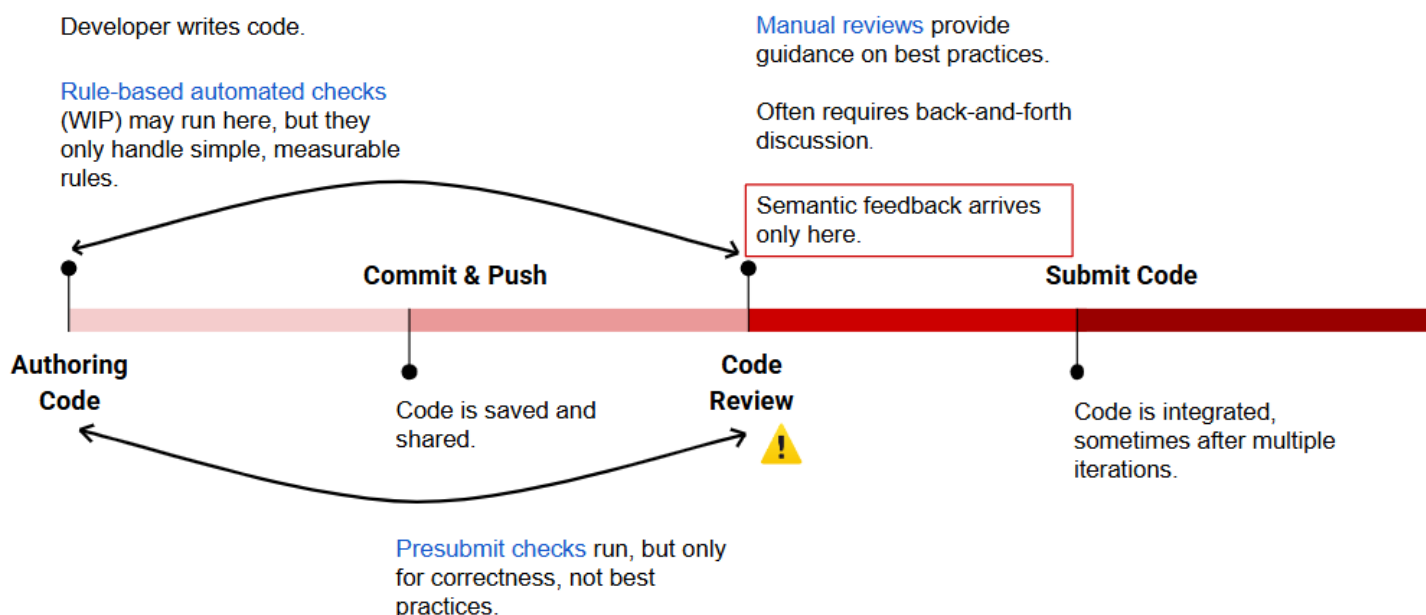


Figure II.10 – Current Developer Workflow Feedback Timeline

II.2 State of the Art: Existing Solutions

During the **Coding Phase**, developers get some support from Rule-Based Checks, but these are limited to simple, objective rules and often miss the nuances of complex best practices.

The most insightful feedback on design and best practices typically comes during **Code Review**, but this happens after the initial development, making changes more disruptive.

Finally, **Presubmit Checks** before submission focus on correctness and safety, not on proactive best practice guidance.

This leaves a significant gap: there's no real-time, intelligent support for adhering to complex best practices while the developer is actively coding, which is the gap our project aims to fill.

Approach	Strengths	Limitations
Code Reviews	Context-aware, high-level insights	Feedback delayed, time-consuming
Presubmit Checks	Fast, automated validation	Limited scope, mostly syntax and safety
Rule-Based Checks	Consistency, objectivity	Cannot handle complex or contextual practices

Tableau II.1 – Comparison of available software quality support solutions

Impact of Delayed Feedback Following from the previous analysis, these workflow issues create concrete impacts on development:

- **Technical Debt Accumulation:** Delayed feedback and limited enforcement of best practices lead to accumulating technical debt and inconsistencies in code quality over time.
- **Prolonged Review Cycles:** Review cycles take longer because developers must iterate multiple times to address issues discovered late in the process.
- **Developer Frustration:** The repetitive cycle of late-stage corrections contributes to developer frustration and reduced productivity.
- **Inconsistent Quality Standards:** Without real-time guidance, adherence to best practices varies significantly across team members and projects.
- **Increased Development Costs:** The cost of fixing issues increases exponentially the later they are discovered in the development process.

This analysis clearly demonstrates why we need a solution that brings guidance earlier, directly into the authoring process, addressing the fundamental timing and intelligence gaps in current approaches.

Trends and Emerging Practices Modern development increasingly integrates AI-driven assistance, predictive code analysis, automated documentation, and intelligent testing recommendations. These trends aim to provide proactive, context-aware guidance within the IDE, reducing errors and improving developer efficiency. However, as our analysis shows, there remains a critical gap in providing real-time, intelligent feedback during the active coding phase.

The limitations identified in current approaches—delayed feedback, limited scope of automated checks, and inability to handle complex best practices—create a compelling case for integrating LLM-powered assistance directly into the development workflow. This represents the next evolution in development tooling, moving from reactive quality assurance to proactive, intelligent guidance that addresses the fundamental gaps in current solutions.

3 Project Requirements

To tackle the issue of delayed feedback and limited intelligent assistance, our solution integrates the power of large language models directly into the developer’s workflow in the IDE.

The proposed system builds on gaps identified in existing solutions. It aims to provide real-time, AI-driven feedback directly in the developer workflow, while maintaining performance, scalability, and usability.

3.1 Functional Requirements

The system must:

- **Detect Framework Violations:** Identify violations of internal YouTube framework best practices and coding standards in real-time during development.
- **Provide Contextual Explanations:** Explain violations in clear, developer-friendly language with context-specific rationale that helps developers understand why certain patterns are problematic.
- **Generate Actionable Fixes:** Suggest specific, actionable fixes leveraging AI-generated solutions tailored to YouTube framework patterns and conventions.
- **Enable Developer Interaction:** Allow developers to accept, reject, or modify AI suggestions, providing full control over the implementation of recommended changes.
- **Maintain Contextual Relevance:** Ensure feedback appears in the appropriate location within the code and remains relevant and accurate as the developer continues coding.

- **Seamless IDE Integration:** Integrate seamlessly with the existing internal IDE, extension, and backend workflow without disrupting the developer’s current development process.

3.2 Non-Functional Requirements

The system should also meet broader quality criteria:

- **Performance:** Provide fast, near real-time responses to avoid interrupting developer workflow.
- **Scalability:** Efficiently handle large codebases and multiple simultaneous users.
- **Maintainability:** Enable modular updates, addition of new AI models, or coding rules.
- **Reliability:** Ensure robustness in production environments with minimal downtime.
- **Security and Privacy:** Comply with organizational policies, ensuring safe handling of code and data.
- **Usability:** Deliver concise, context-aware, and minimally intrusive feedback.
- **Extensibility:** Easily add new rules, models, or integrations.

Requirement Type	Description
Functional	Framework violation detection, contextual explanations, actionable fixes, developer interaction, contextual relevance, seamless IDE integration
Non-Functional	Performance, scalability, maintainability, reliability, security, usability, extensibility

Tableau II.2 – Summary of project requirements

Rationale These requirements address limitations identified in existing solutions by embedding proactive, context-aware feedback directly in the coding workflow, specifically targeting YouTube framework best practices. This approach enhances developer productivity, reduces framework-specific errors, and supports consistent adherence to internal YouTube development standards.

Conclusion

This chapter established the business and theoretical foundation for integrating AI into software development workflows. Beginning with an analysis of the Software Development Life Cycle, it identified key challenges in maintaining code quality and consistency across growing development teams. The exploration of AI technologies—particularly Large Language Models and AI agents—revealed their potential for addressing these challenges through proactive, context-aware assistance.

The examination of existing solutions highlighted significant limitations: traditional tools provide limited analysis, and human review offers inconsistent feedback. There remains a critical gap in delivering real-time, intelligent assistance that scales with team growth. The AI-enhanced SDLC paradigm addresses these limitations by embedding intelligent assistance throughout the development process.

The project requirements defined in this chapter focus on real-time, AI-driven feedback for YouTube framework best practices that integrates seamlessly into the development workflow. The proposed solution bridges the gap between AI potential and practical development needs, setting the stage for the detailed system design presented in the following chapter.

Chapter III

Implementation

Summary

1	Outils et langages utilisés	32
2	Présentation de l'application	32
2.1	Exemple de tableau	33
2.2	Exemple de Code	33
3	Remarques sur la bibliographie	34

Introduction

Ce chapitre porte sur la partie pratique ainsi que la bibliographie.

1 Outils et langages utilisés

L'étude technique peut se trouver dans cette partie, comme elle peut être faite en parallèle avec l'étude théorique (comme le suggère le modèle 2TUP). Dans cette partie, il faut essayer de convaincre le lecteur de vos choix en termes de technologie. Un état de l'art est souhaité ici, avec un comparatif, une synthèse et un choix d'outils, même très brefs.

2 Présentation de l'application

Il est tout à fait normal que tout le monde attende cette partie pour coller à souhait toutes les images correspondant aux interfaces diverses de l'application si chère à votre coeur, mais abstenez vous! Il FAUT mettre des imprime écrans, mais bien choisis, et surtout, il faut les scénariser : Choisissez un scénario d'exécution, par exemple la création d'un nouveau client,

et montrer les différentes interfaces nécessaires pour le faire, en expliquant brièvement le comportement de l'application. Pas trop d'images, ni trop de commentaires : concis, encore et toujours.

Évitez ici de coller du code : personne n'a envie de voir le contenu de vos classes. Mais vous pouvez insérer des snippets (bouts de code) pour montrer certaines fonctionnalités [1][2], si vous en avez vraiment besoin. Si vous voulez montrer une partie de votre code, les étapes d'installation ou de configuration, vous pourrez les mettre dans l'annexe.

2.1 Exemple de tableau

Vous pouvez utiliser une description tabulaire d'une éventuelle comparaison entre les travaux existants. Ceci est un exemple de tableau: Tab [III.1](#).

Tableau III.1 – Tableau comparatif

	Col1	Col2	Col3	Col4
Row1		X		
Row2	X			
Row3	X	X	X	X
Row4	X		X	X
Row5	X		X	X
Row6	X		X	X
Row7	X		X	
Row8	X	X	X	

2.2 Exemple de Code

Voici un exemple de code Java, avec coloration syntaxique [III.1](#).

Listing III.1 – Helloworld Java

```
public class HelloWorld {  
    //la m thode main  
    public static void main(String[] args) {  
        System.out.println("Hello, World");  
    }  
}
```

}

3 Remarques sur la bibliographie

Votre bibliographie doit répondre à certains critères, sinon, on vous fera encore et toujours la remarque dessus (et parfois, même si vous pensez avoir tout fait comme il faut, on peut vous faire la remarque quand même : chacun a une conception très personnelle de comment une bibliographie devrait être).

- Une bibliographie dans un bon rapport doit contenir plus de livres et d'articles que de sites web : après tout c'est une biblio. Privilégiez donc les ouvrages reconnus et publiés pour vos définitions, au lieu de sauter directement sur le premier article wikipedia;
- Les éléments d'une bibliographie sont de préférence classés par ordre alphabétique, ou par thèmes (et ordre alphabétique pour chaque thème);
- Une entrée bibliographique doit être sous la forme suivante :
 - Elle doit contenir un identifiant unique: représenté soit par un numéro [1] ou par le nom du premier auteur, suivi de l'année d'édition [Kuntz, 1987];
 - Si c'est un livre : Les noms des auteurs, suivi du titre du livre, de l'éditeur, ISBN/ISSN, et la date d'édition;
 - Si c'est un article : Les noms des auteurs, le titre , le journal ou la conférence, et la date de publication;
 - Si c'est un site web ou un document électronique : Le titre, le lien et la date de consultation;
 - Si c'est une thèse : nom et prénom, titre de la thèse, université de soutenance, année de soutenance, nombre de pages;
 - Exemples :

[Bazin, 1992] BAZIN R., REGNIER B. Les traitements antiviraux et leurs essais thérapeutiques. Rev. Prat., 1992, 42, 2, p.148-153.

III.3 Remarques sur la bibliographie

[*Anderson, 1998*] ANDERSON P.JF. Checklist of criteria used for evaluation of metasites. [en ligne]. Université du Michigan, Etats Unis. Site disponible sur : <http://www.lib.umich.edu/megasite/critlist.html>. (Page consultée le 11/09/1998).

- Dans le texte du rapport, on doit obligatoirement citer la référence en faisant appel à son identifiant, juste après avoir utilisé la citation. Si ceci n'est pas fait dans les règles, on peut être accusé de plagiat.

Conclusion

Voilà.

Conclusion and Perspectives

C'est l'une des parties les plus importantes et pourtant les plus négligées du rapport. Ce qu'on ne veut pas voir ici, c'est combien ce stage vous a été bénéfique, comment il vous a appris à vous intégrer, à connaître le monde du travail, etc.

Franchement, personne n'en a rien à faire, du moins dans cette partie. Pour cela, vous avez les remerciements et les dédicaces, vous pourrez vous y exprimer à souhait.

La conclusion, c'est très simple : c'est d'abord le résumé de ce que vous avez raconté dans le rapport : vous reprenez votre contribution, en y ajoutant ici les outils que vous avez utilisé, votre manière de procéder. Vous pouvez même mettre les difficultés rencontrées. En deuxième lieu, on y met les perspectives du travail : ce qu'on pourrait ajouter à votre application, comment on pourrait l'améliorer.

References

- [1] SOUHEIB YOUSFI AND LILIA SFAXI. *Rapport Latex*. Département Math-info (2015).
- [2] MR. LATEX. Débuter avec Latex. www.latex.com, (2008). [En ligne; consulté le 19-Juillet-2008].

Appendix : Miscellaneous remarks

- Un rapport doit toujours être bien numéroté;
- De préférence, ne pas utiliser plus que deux couleurs, ni un caractère fantaisiste;
- Essayer de toujours garder votre rapport sobre et professionnel;
- Ne jamais utiliser de je ni de on, mais toujours le nous (même si tu as tout fait tout seul);
- Si on n'a pas de paragraphe 1.2, ne pas mettre de 1.1;
- TOUJOURS, TOUJOURS faire relire votre rapport à quelqu'un d'autre (de préférence qui n'est pas du domaine) pour vous corriger les fautes d'orthographe et de français;
- Toujours valoriser votre travail : votre contribution doit être bien claire et mise en évidence;
- Dans chaque chapitre, on doit trouver une introduction et une conclusion;
- Ayez toujours un fil conducteur dans votre rapport. Il faut que le lecteur suive un raisonnement bien clair, et trouve la relation entre les différentes parties;
- Il faut toujours que les abréviations soient définies au moins la première fois où elles sont utilisées. Si vous en avez beaucoup, utilisez un glossaire.
- Vous avez tendance, en décrivant l'environnement matériel, à parler de votre ordinateur, sur lequel vous avez développé : ceci est inutile. Dans cette partie, on ne cite que le matériel qui a une influence sur votre application. Que vous l'ayez développé sur Windows Vista ou sur Ubuntu n'a aucune importance;
- Ne jamais mettre de titres en fin de page;
- Essayer toujours d'utiliser des termes français, et éviter l'anglicisme. Si certains termes sont plus connus en anglais, donner leur équivalent en français la première fois que vous les utilisez, puis utilisez le mot anglais, mais en italique;
- Éviter les phrases trop longues : clair et concis, c'est la règle générale !

Rappelez vous que votre rapport est le visage de votre travail : un mauvais rapport peut éclipser de l'excellent travail. Alors prêtez-y l'attention nécessaire.