Adapting Coding and QA Practices for Using AI-Assisted Coding Tools in the European Regulated Healthcare Environments

Blazej Goszczynski

Haaga-Helia University of Applied Sciences
Business Information Technology
Bachelor's Thesis
2026

Abstract

Author
Blazej Goszczynski

Degree
Bachelor of Business Administration

Report/Thesis Title
Adapting Coding and QA Practices for Using AI-Assisted Coding Tools in the European Regulated Healthcare Environments

Number of pages and appendix pages
42 + 4

The increasing use of AI-assisted coding tools is reshaping software development, including European regulated healthcare environments. These environments are shaped by regulations and quality requirements, and put great value on safety, reliability, transparency, and traceability of software systems. AI-assisted coding tools offer potential benefits in productivity or creativity; however, existing regulations and standards provide high-level requirements, however, they offer limited practical guidelines on how such tools should be used in daily coding and quality assurance practices. This can cause uncertainty for developers and organizations aiming to balance productivity and regulatory compliance.

The objective of this thesis was to examine how developers can adapt their coding and QA practices when using AI-assisted coding tools in European regulated healthcare environments. The study focused on identifying benefits and risks associated with AI-assisted coding, analyzing how European regulations influence modern healthcare software development, determining necessary adaptations to maintain safety and compliance, and comparing AI-assisted coding with conventional coding in terms of coequalities coverage, and development efficiency. The theoretical framework of this study was based on European healthcare regulations and standards, including MDR/IVDR, IEC 62304, ISO 13485, ISO 14971, GDPR, and the emerging EU AI Act.

The study was implemented using a mix-method approach. Empirical data was collected through six semi-structured interviews with professionals related to software development and quality assurance and through a comparative coding experiment including a prototype cardiovascular report generation system implemented both manually and with AI assistance. The findings indicate that AI-assisted coding improved development in terms of efficiency, especially for routine tasks, however, it introduces risks in reliability, explainability, data security, and increased QA workload. Test coverage remained similar between AI-assisted and manual implementation, with differences in defects, code duplication, and structural code quality.

Based on these findings, a practical developer-oriented framework was created supporting responsible use of AI-assisted coding tools in European regulated healthcare environments. The framework complements existing regulations and standards and provides practical guidelines in data governance, human oversight, review process, testing, traceability, and developer training.

Key words
AI-Assisted Coding, Software Quality Assurance, Healthcare Software Development, European Regulations

Table of Contents

1 Introduction

Healthcare software development is one of the most regulated industries, where its integration with AI-assisted coding tools presents both opportunities, risks, and challenges. These tools promise faster development and improved efficiency; however, they additionally raise concerns about AI-generated code outputs' reliability, safety, and transparency. Therefore, it is essential to ensure compliance with strict legal frameworks such as, such as the Meical Device Regulation (MDR) or General Data Protection Regulation (GDPR) (Regulation on medical devices (EU) 2017/745; Regulation on the protection of personal data 2016/679). The rising shortage of medical staff and professionals, and the rising number of patients, call for reliable and efficient healthcare software more than ever (European Parliament 2025). Therefore, the adoption of AI-assisted coding tools into healthcare software development environments is appealing to many. This creates a need for structured and systematic practices that developers and quality assurance specialists can use to remain in control over these adaptations.

This thesis aims to address these issues by exploring how coding and quality assurance practices can be adapted when AI-assisted coding tools are used in European healthcare environments.

1.1 Objectives, research problems, and demarcation

The primary objective of this thesis is to combine a literature review, semi-structured interviews, and a comparison prototype experiment to create a practical framework for the responsible use of AI-assisted coding tools in the European regulated healthcare environment. The framework consists of guidelines and QA recommendations. To ensure the framework is supported by different sources, this research applies triangulations through a mixed-method approach combining qualitative research (semi-structured interviews), quantitative research (comparative prototype experiment), and a literature review.

The main research problem of this thesis is: How can developers adapt their coding and quality assurance practices when using AI-assisted coding tools in European regulated healthcare environments.

Investigative questions are formed in attempt to solve this question. The first investigative question explores the benefits and risks behind AI-assisted coding to identify motives behind adapting such tools into the software development and identify what risks must be watched out for when adopting them. However, framing the discussion between only benefits and risks may oversimplify the outcomes, therefore this question requires a deeper understanding of real development environment functioning.

The second investigative question explores the impact of European regulations on software development in healthcare. This question aims to contribute to the main research questions by defining current conditions or giving examples to which software developers must adhere to and understand how software developer practices can be adapted without compromising existing regulations. However, this question impels that regulations alone determine developer practices, whereas these regulations often define only high-level obligations. The third investigative question investigates what adaptations are necessary to use AI-assisted coding and addresses this issue by indicating where developers can strengthen their coding and QA practices when using AI without compromising human responsibility. Additionally, this question is necessary to identify which main practices are affected by AI-assisted coding tools. Although identifying individual adaptations is beneficial, this question does not address how these adaptations can be put into use.

The fourth investigative question aims to solve this issue by transforming adaptations into a usable framework that developers can apply in their work. By synthesizing the adaptations into a framework, this question is necessary to ensure the research outcomes extend beyond isolated recommendations. However, even well-defined adaptations can be applied incorrectly, this critique motivates the need for a framework that can be translatable to many environments. Finally, the fifth investigative question shows differences between manual and AI-assisted coding, reinforcing the main research question by investigating the level of technical contrast between the two methods and providing support for empirical evidence. However, comparative results will reveal only performance differences, the fifth question is used to contribute to the first and third questions by providing technical insights.

This research is deliberately limited to meeting the timeframe and scope of a bachelor's thesis. This work will focus on the European healthcare market and EU/EEA laws such as Medical Device Regulations (MDR) and General Data Protection Regulations (GDPR), and harmonized standards such as ISO and EN IEC. The empirical part contains the comparison prototype experiment, which involves the development of a cardiovascular report generation system in two versions (Traditional and AI-assisted) and not a production-level healthcare software or comparison between AI-assistant tools; therefore, it will not make use of clinical trials or any access to patient data (the data used will be synthetic). Additionally, the empirical section will contain semi-structured interviews with only software professionals working in healthcare or other regulated environments. No outside EU/EEA market regulations will be included.

1.2 Key Concepts

- **AI-Assisted Coding Tools** – Computer programs using AI to help developers in any form.

- **Software Industry professionals** – People working is software-related positions with experience in software development, quality assurance or related technical roles.
- **Software Development Practices** – Common to software development practices (such as version control, testing, documentation or code reviews)
- **Software Quality Assurance (QA)** – System responsible for monitoring product standards
- **Healthcare IT Systems** – systems designed for healthcare environments
- **Regulated Environments** – Industries subjected to government-mandated rules

1.3 Sustainability Perspective

The objectives of this thesis reflect sustainability in multiple forms:

- **Economic sustainability** – By exploring how developers can increase their productivity by using AI-assisted coding tools, this thesis contributes to a more efficient and cost-effective software development, which companies can use to stay highly competitive and effective without sacrificing regulatory compliance.
- **Social sustainability** – Safe and reliable code is the foundation of trust between the developer and customer; AI-generated coding weakens trust. This thesis strengthens this bond by exploring how AI-coding techniques can stay transparent and compliant, ultimately, improving patient outcomes and reducing usage strain on healthcare IT systems.
- **Environmental sustainability** – By promoting responsible training for using AI-assisted coding tools, this thesis establishes awareness about sustainable and energy-cautious AI, leading to better resource management. (Strubell, Ganesh & McCallum 2020.)

1.4 Cover Matrix

Investigative Questions	Theoretical Framework	Results	Methods
1. What benefits and risks hide behind AI-assisted coding in European healthcare software development?	2.1, 2.4	4.1.1	Literature review, interviews
2. How do European regulations influence modern software development in healthcare?	2.2, 2.3	4.1.2	Literature review, interviews
3. What adaptations are necessary to use AI-assisted coding and maintain safety, transparency, reliability, and compliance?	2.2	4.1.3	Literature review, interviews
4. How can those adaptations be put together into a usable framework that developers can use?	2.1, 2.2	4.1.3, 4.2	Prototype experiment
5. How does AI assisted coding compare to conventional coding in terms of code quality, test coverage, and development efficiency?	2.1, 2,4	4.1.4	Integration of interview & prototype results

2 Theoretical Framework

Every day, we become more dependent on tools that make us more efficient; it is no different in software development. The advancements in artificial intelligence prompted the emergence of AI-assisted coding tools such as GitHub Copilot and ChatGPT, which have affected how software development functions. These tools, built upon Large Language Models (LLMs) and trained with vast amounts of programming data and code, can automate numerous tasks previously perceived as solely for programmers, and by utilizing Natural Language Processing (NLP), these tools have no challenge in understanding and generating human or programming language. Unlike traditional coding, they use pattern recognition and nonlinear thinking to generate contextual code, suggestions, and decisions, making them appeal to a range of developers. The adoption of AI-assisted coding tools changes how developers work, from sole programmers to putting them into a collaborative state with the AI tools. (Wong, Guo, Hang, Ho & Tan 2023.)

The right column of Figure 1 was originally used in the work of Saravanan & al (2025), to present an Overall System Architecture as part of their dual-module approach. To illustrate the differences between manual and AI-assisted coding workflows, this thesis adapted the workflow and added manual coding system architecture.

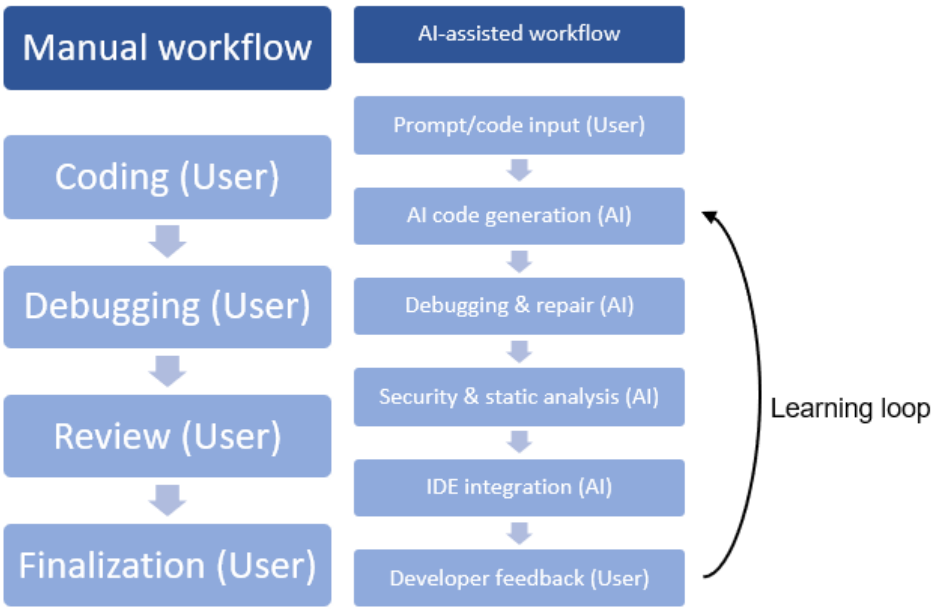

Figure 1. Comparison of manual and AI-assisted coding workflows (Right column (AI-assisted workflow) adapted from Saravanan et al. 2025, 2911)

The development of AI-assisted coding tools builds upon a long history of Programming assistants. Early tools, such as IntelliSense or static analyzers, offered autocomplete features and syntax checkers, however, were limited to the programming language or coding environment used. The release of ChatGPT Codex in 2021 and, soon after, GitHub Copilot marked a tipping point in software development, and the adoption of AI-assisted coding tools skyrocketed. As shown in Figure 2, 84% of developers use or plan to use AI tools for coding in year 2025. This rapid industry change calls for structured and systematic practices that developers and quality assurance specialists use to adapt AI-assisted coding tools into their work.

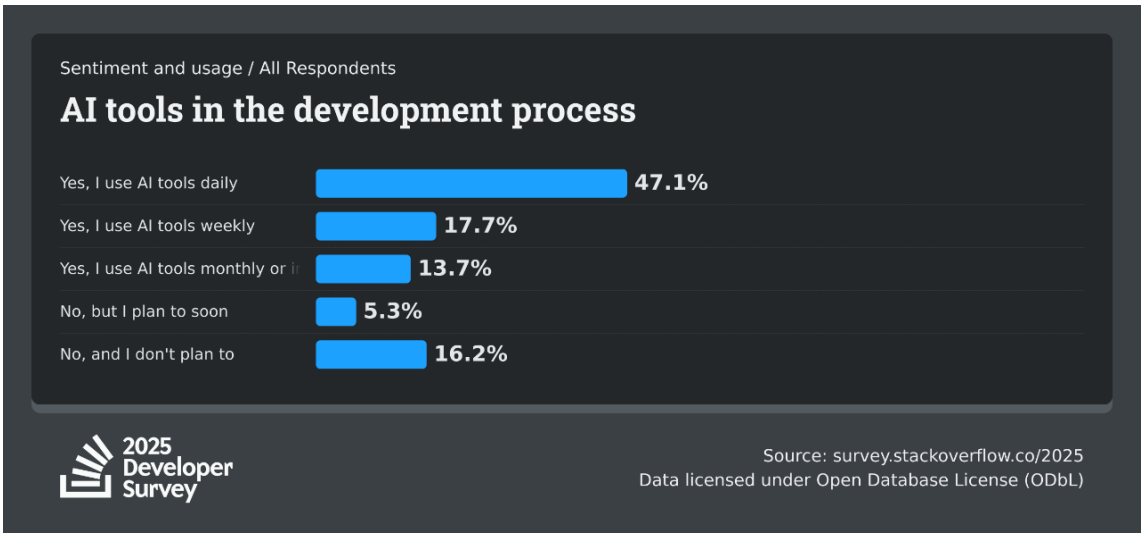

Figure 2. AI tools in the development process (Stackoverflow 2025)

Beyond its rapid adaptation speed, AI-assisted coding tools are additionally known for the numerous benefits they offer in software development in European healthcare. One of the benefits referred to is increased development efficiency, compared to traditional coding, where developers were responsible for manually implementing logic, writing unit tests, and debugging errors (Peng, Kalliamvakou, Cihon & Demirer 2023; Yuan & al. 2024). Developers using AI-assisted coding tools can widely automate these tasks, allowing them to prioritize system design and innovations. However, a study by Becker, Rush, Barnes and Rein (2025) found that experienced developers can be slowed down by these tools. Another benefit of AI-assisted coding tools is that they can serve as a learning aid, especially for junior and new developers. These tools can generate suggestions, feedback, and promote best practices, helping developers with less experience to boost their code and framework understanding. (Ferino, Hoda, Grundy & Treude 2025.) Additionally, these tools can function as a "programmer assistant", by generating suggestions that help developers brainstorm functionalities and debugging, or explain documentation, ultimately increasing developers' output (Chen & al. 2025). Taken together, these benefits show that AI-assisted coding tools not only boost developers' productivity, however, additionally change how they function daily.

Even though AI-assisted coding tools offer multiple benefits, they come with risks and challenges that need to be properly addressed before implementing these tools into software development in the European healthcare environment. The most popular vulnerability is the code quality produced by these tools, as AI-generated code can be over-engineered, incorrect, or disregard coding standards (Licorish, Bajpai, Arora, Wang & Tantithamthavorn 2025). Another major risk is maintaining safety and compliance, as developers who use AI-assisted coding tools can create significantly more security vulnerabilities compared to those who do not. What is worse is that these developers can be convinced that they wrote secure code. (Perry, Srivastava, Kumar & Boneh 2023.) Another risk and challenge are maintainability issues; a significant number of healthcare software systems cannot be left unattended after deployment and need to be accessed for potential updates or modifications. When it was the AI that created or coded the software, the risk of poorly documented and less readable code rises (Abbassi, Silva, Nikanjam & Khomh 2025). AI-assisted coding tools are effective in generating vast amounts of tests, which can often fool developers. These tests, however, have proven to lack behind in execution, finding edge cases, and in some cases, simply ineffective. (Huang & al. 2025.)

The combinations of benefits, risks, and challenges are proof that AI-assisted coding tools can be positive and negative for developers, which not only increase productivity, however, additionally change how developers operate entirely. These tools shift developers entirely from manual coding to prompt engineering, reviewing, and output selection, putting more weight on quality assurance systems, especially in regulated environments such as healthcare. It is essential that the wide applicability of AI-assisted coding tools does not cause overreliance and overconfidence in them and does not weaken the developer’s creativity and critical thinking, as those attributes are the foundational reasons why we can enjoy safe and reliable healthcare systems all over the world.

2.1 Software Development and Quality Assurance in Regulated Environments

Quality assurance (QA) plays a key role in European regulated environments; it is essential in industries such as healthcare, where badly organized systems or code defects can not only lead to financial losses or reputation damage, however, additionally endanger human life. In non-regulated environments, QA can be more flexible and focus its main resources on business objectives or performance, whereas in regulated industries, QA is responsible for risk management or ensuring compliance with strict legal frameworks such as Medical Device Regulations (MDR). It consists of multiple activities, however, the most common in regulated software development environments are test planning and design, test execution and bug fixing, and documentation and reporting. It serves not only as a test foundation, however, additionally as a wide system for accountability and defect management, ensuring that the development processes are safeguarded in terms of

compliance, efficiency, reliability, and safety, ultimately building trust for regulatory institutions, businesses, and customers. (Niţă, Stan & Ţîţu 2024.)

Quality assurance (QA) in a regulated software environment is built upon numerous industry standards and internal company processes. These processes include Verification & Validation (V&V) that check if the right software is being built and if it is done in the right way, and process audits where the development practices are validated for adherence to the industry standards. These processes are supported by standards such as EN ISO 13485 (Medical device QMS standard), EN ISO 14971 (Medical device risk management standard), EN IEC 62304 (Medical device software lifecycle standard), EN IEC 81001-5-1 (Health software cybersecurity lifecycle standard), and EN IEC 82304-1 (Health software product safety standard). (ISO 13485:2016; ISO 14971:2019; IEC 62304:2006/Amd 1:2015; IEC 81001-5-1:2021; IEC 82304-1:2016.)

Together, these standards form a framework that directly impacts how developers write comprehensive documentation, implement risk management, and ensure safety and full transparency, for example, by requiring that every code block needs to be properly documented, tested, and linked to the right requirement/ticket. By compliance with these standards, developers can contribute to safer software, better market access, and continuous improvement. It is essential to mention that these EN ISO and EN IEC standards are not laws themselves, rather they are part of the harmonized standards of MDR which is a European law, these standards are recommended to show compliance with legal regulations (European Union s.a.). Unlike nonregulated environments where quality assurance can be lightweight and inexpensive, in regulated environments they are often complex, formalized, and resource heavy.

Although Quality assurance and regulations can contribute to product safety and reliability, various of those practices can cause complications and a need for developers to adapt when using AI-assisted coding tools in regulated environments. A range of AI-generated codes can appear correct and well-structured while containing multiple flaws uncaught by a developer’s naked eye. A study by Pearce, Ahmad, Tan, Dolan-Gavitt and Karri (2025) in accessing AI-generated code found that out of 1689 generated programs, 40% were vulnerable; this leads to a risk of those vulnerabilities being pushed further in the development cycle or even to the production line. Developers to detect these flaws early and maintain reliability should adopt quality analysis of AI-generated code, static analyzers, and implement a mandatory human code review.

As AI-assisted coding tools evolve, maintaining transparency in complex environments can become more challenging, and calls for adaptations for developers to avoid conflicting with standards such as EN IEC 62304 (which requires documentation, planning, verification, and traceability). Watson and Van Itallie (2025) recommend organizations to a develop comprehensive GenAI

transparency strategy, including an immediate implementation of a transparency system to monitor AI-generated code throughout the whole life cycle, an investment in education and training to increase understanding of AI tools and transparency practices, additionally they recommend integrating government policies (quality standards, security requirements, intellectual property management procedures) into the software development lifecycle.

Due to LLMs being used in AI-assisted coding tools are trained on vast amounts of publicly available data (for example, from GitHub) and data from prompts shared with the tools can be reused as training data, these tools can unintentionally generate license-protected code (King, Klyman, Capstick, Saade & Hsieh 2025; Xu, Gao, He & Zhou 2025). As it is difficult to spot license issues and security vulnerabilities, causing security and compliance risks, developers can adapt by using SAST/DAST (Static / Dynamic Application Security Testing) and Software Composition Analysis (SCA) tools. Additionally, a recent study in developer self-declaration of AI-generated code showed that almost 25% of responders claimed to never self-declare AI-generated code, this could lead to license issues, or difficulties in categorizing of AI-generated code. The study recommended practices such as writing comments indicating code origin, prompt used, and scope of AI-gen parts. (Kashif, Liang & Tahir 2025.)

While these complications can slow down software development in regulated environments, quality assurance practices and regulations can give valuable insight into safe and reliable software development across all industries. By combining these insights from recent research, industry regulations and standards, semi-structured interviews, and proposed implementations such as traceability structure, static analyzers, and variability and license scanners, a practical framework can be created to guide developers for safe use of AI-assisted coding tools in software development in European healthcare-regulated environments.

Multiple existing frameworks intend to mitigate with the challenges and risks introduced by AI-assisted coding tools. However, numerous of them focus on high-level structures or are insufficiently addressed to practices of the European and Healthcare software development domain. The EU Ethics guidelines for trustworthy AI are one of the most cited frameworks for responsible AI use. While it focusses on promoting and maintaining human agency, transparency, accountability, and technical robustness, relating closely to this works framework, it remains intentionally abstract, and does not provide concrete practices for developers on adapting AI-assisted coding tools.

Similarly, the EU AI Act introduces legally binding principles such as human oversight, risk management, and documentation for high-risk AI systems. Even though the act has high influence on European healthcare software development, it focuses mainly on regulating AI systems based on risk classification (Regulation (EU) 2024/1689). Other organizational frameworks such as the

ISO/IEC 42001 focus on managerial governance levels, and while essential for maintaining organizational readiness, they do not focus on developer level activities for adopting AI-assisted coding tools (ISO/IEC 42001:2023).

Finally, recent HULA (Human-in-the-Loop LLM-based Agents) framework introduces a structured human-AI collaboration process that enables AI planning and code generation while maintaining human control. Unlike high-level frameworks HULA focusses on enabling human oversight throughout reach development stage. However, it primarily focusses on interactions and control flow and does not cover topics such as data governance or QA integration. (Takerngsaksiri & al. 2025.)

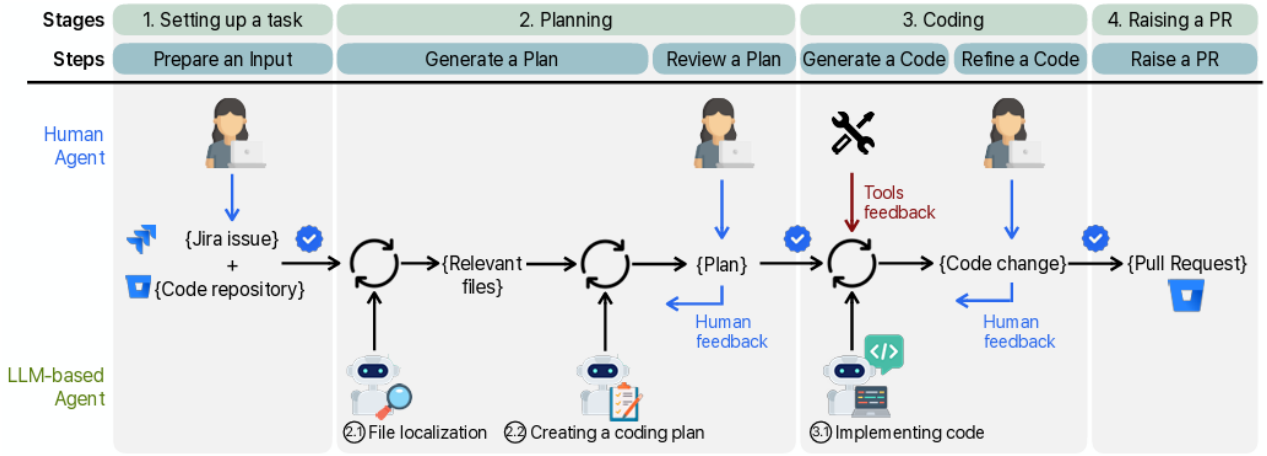

Fig. 1. An Overview of our Human-in-the-Loop LLM-based Software Development Agents Framework (HULA), consisting of three agents (i.e., AI Planner Agent, AI Coding Agent, and Human Agent) work cooperatively to achieve the common goal of resolving a JIRA issue.

Figure 3. Overview of the Human-in-the-Loop LLM-based software development agents' framework (HULA) for solving a Jira ticket (adopted from Takerngsaksiri et al., 2025)

Despite the availability of ethical, regulatory and organizational AI frameworks, a clear gap remains in developer-oriented adaptations for coding and quality assurance practices for AI-assisted software development in regulated healthcare environments. In contrast to these frameworks, the proposed framework in this thesis focusses on day-to-day developer level operations that should directly influence coding and QA practices when dealing with AI-assisted coding tools in European regulated healthcare environments. The proposed framework discusses regulatory and theoretical requirements and translates them based on empirical evidence from semi-structured interviews and a comparative coding experiment to developer-oriented adaptations.

2.2 Healthcare IT Regulations in Europe

Coverage reports were generated identically using command represented by figure 5. These laws have been established to protect patients by ensuring the safety, reliability, and accountability of medical software systems and their development processes which is especially important when AI-assisted code might look correct, however, is flawed. The General Data Protection Regulation (GDPR) is a fundamental law of the EU/EEA, whose primary purpose is to ensure that every individual has control over how their personal data is collected, processed, stored, and shared (Regulation on The Protection of Personal Data 2016/679). It was adopted in 2016 and made fully applicable in 2018. In terms of coding and QA practices, it mandates, for example, that developers adhere to privacy by design principles by default, data minimization, security through encryption and testing, user rights such as data access and deletion, and full documentation of data processing and handling.

Another important regulation for healthcare software development is the European Medical Device Regulation (MDR) (Regulation on Medical Devices (EU) 2017/745) which applies to devices that have direct contact with or influence on the patients, such as surgical instruments or implants. Through its adaptation in 2017 and making it fully applicable in 2021, the European Parliament and the European Council mandate developers to follow, software risk classifications, verification and validation processes, traceability, change and configuration management, and documentation.

Additionally, Vitro Diagnostic Regulation (IVDR) applies to software that analyzes in vitro (biological samples) It was adopted in 2017 and made fully applicable in 2022; it specifically refers to in vitro diagnostic medical devices used, for example, in lab analysis, genetic testing, or disease screening (Regulation on in Vitro Diagnostic Medical Devices (EU) 2017/746). Both MDR and IVDR are closely related and cover several similar areas, such as software lifecycle management, V&V, traceability, and documentation. However, MDR is intended for traditional medical devices, and IVDR was for in vitro diagnostic medical devices; these differences lead to variations in areas such as classification rules, risk assessment, and documentation depth.

To put legal regulations such as GDPR, MDR, and IVDR into practice, developers and quality assurance specialists rely on harmonized standards. While MDR and IVDR define legal obligations, they do not provide exact technical processes or practices required. This is where harmonized standards come into play; by giving guidance on quality management, risk control, and the medical device software lifecycle, they help developers achieve adherence to MDR or IVDR. Harmonized standards are not legally mandatory, and their use is voluntary; however, following them demonstrates compliance with legal frameworks, and organizations are still free to choose different technical solutions to demonstrate compliance.

Developers using AI-assisted coding in healthcare environments should follow standards such as EN ISO 13485:2016, which was introduced already in 1996, and with its most recent amendment A11 in 2021, was harmonized with its amendment under MDR and IVDR in 2022 (ISO 13485:2016; Commission Implementing Decision (EU) 2022/757). Though quality management system principles that are tailored for the safety and reliability requirements of medical devices, it shapes how developers do software design, implementation, validation, and documentation. EN ISO 14971:2019, built from earlier standards with its first consolidated version published in 2000, with its most recent amendment A11 in 2021, became harmonized under MDR and IVDR in 2022 (ISO 14971:2019; Commission Implementing Decision (EU) 2022/757). It governs the risk management of medical devices. For developers, this standard declares how software should be designed, implemented, and maintained with a focus on finding, evaluating, and controlling risks through software lifecycles.

EN IEC 62304:2006, being first published in 2006, with its first amendment (Amd 1) in 2015, was announced as a harmonized standard under MDR and IVDR in 2021 (IEC 62304:2006; Commission Implementing Decision (EU) C(2021) 2406 final). It provides information and guidelines for software lifecycle processes for software used in medical devices, covering planning, requirements, design, implementation, verification, validation, and maintenance. IEC 81001-5-1:2021 published in 2021 and announced harmonized under MDR and IVDR in 2024 (IEC 81001-5-1:2021; Commission Implementing Decision (EU) C(2021) 2406 final). It mandates that developers must implement various security activities though the lifecycle such as threat analysis, secure coding, and vulnerability handling. EN IEC 82304-1:2016 published in 2016 and harmonized under MDR and IVDR in 2024 (IEC 82304-1:2016; Commission Implementing Decision (EU) C(2021) 2406 final). It defines general security requirements thought the whole software lifecycle.

Requirements from MDR/IVDR, GDPR and harmonized standards such as EN ISO 13485, EN ISO 14971, and IEC 62304, can overlap with each other; to avoid repetition and formulize them into concrete examples of obligations from coding and QA practices in healthcare environments, they can be grouped into categories such as lifecycle processes, traceability, risk management, data protection, verification, and quality.

In lifecycle processes, developers are required under MDR/IVDR, EN IEC 62304, and EN ISO 13485 create a documentation system (e.g. confluence) that incorporates the whole software lifecycle process throughout all the requirements, design, implementation, verification, and maintenance stages, store all decisions such as inputs, outputs, reviews, and V&V, and implement version control (e.g. Git) and change control by tracking, versioning, risk assessing, and impact analysis of all updates made. Additionally, developers are obligated to integrate post-market monitoring

and updates as a normal part of the development cycle. For traceability, MDR/IVDR, EN IEC 62304, and ISO 13485, require that all code must be justified and linked to requirements, design, and verification in a requirements management and issue tracking system (e.g. JIRA or GitHub Issues), forming a thread-like structure that can follow the clinical requirement all the way back to the code snippet. In addition, these regulations and standards require risk traceability through documented threats, mitigation, and test evidence.

In risk management, MDR/IVDR, EN IEC 81001-5-1, and EN ISO 14971 require a systematic risk identification protocol that identifies possible software hazards (e.g. misdiagnosis, wrong alerts), with risk classification, using modules identified as high risk through stricter V&V and documentation, and implementing a risk control system that proceeds to mitigation procedures and alerts in case of any failures, if any risks remain after mitigation attempts, they should be documented and justified clearly. For data protection, GDPR and EN IEC 81001-5-1 requires developers to use data encryption and pseudonymization methods and collect/use only minimal and necessary data relevant to their projects. Users should be presented with consent forms allowing them to agree to use their data with the option of withdrawing; additionally, they should have the right to access, edit, delete, and export their data. Systems created by the developers are required to have a logging system for all activities and an incident reporting system.

In verification, MDR/IVDR and standards EN IEC 62304, EN ISO 13485, EN ISO 14971, and EN IEC 81001-5-1 dictate that developers must perform systematic code reviews, static analysis, create, execute, and document all unit and integration tests for software requirements, depending on the risk classification and system tests for products intended use covering performance and functional requirements. Additionally, developers must ensure through tests that each risk control executes properly and simulates clinical real-world performance. Developers can prove adherence with quality obligations through meeting requirements from the previous sections in lifecycle processes, traceability, risk management, data protection, and verification. Additionally, developers are required to be qualified for medical software development and follow a structured development system with adherence to coding conventions, peer reviewing, and confirming the technical suitability of third-party libraries, APIs, and AI models. (Regulation on the protection of personal data 2016/679; Regulation on medical devices (EU) 2017/745; Regulation on in vitro diagnostic medical devices (EU) 2017/746; ISO 13485:2016; ISO 14971:2019; IEC 62304:2006.)

When developers use AI-assisted coding tools, they fall under the EU AI Act expectations for transparency, human oversight, and risk management, as a result the outcome is a high-risk medical software. It requires, among others, that developers have full control and can intervene, override, and stop AI-generated decisions. (European Commission 2025.)

2.3 Explainability, Transparency and Trust in AI

Using AI-assisted coding tools in healthcare can cause major doubt in code quality and product reliability. This is due to numerous AI models work as “black boxes” with billions of parameters, internal reasoning, and operations that are hard to justify at first sight, causing difficulties for developers and clinicians. This problem closely relates to the interpretability of AI models, which is explained as the level of AI models’ internal workings that can be understood. It determines how much developers and clinicians can understand why a model behaves in a specific way. (Garouani, Mothe, Barhrhouj & Aligon 2024.)

Interpretability is crucial for debugging, validation, and assessment of AI-assisted coding tools. However, developers cannot simply look at model weights and justify the reasoning behind the AI-generated code. To increase interpretability, one can apply explainability methods. Explainability refers to the set methodologies used to increase interpretability of an AI model, and involves documentation and techniques such as confidence scoring, assigning a confidence level to each AI generated output, prompt-to-code attribution, where each code generates is linked to the used prompt, and semantic and structural analysis, such tools help explain and understand the AI generated code,, however, these techniques are experimental and not widely adopted yet. In regulated environments, explainability and interpretability can support transparency and verification.

Transparency and trust in medical software are cornerstones of efficient collaboration between developers and clinicians. Studies on explainable AI in healthcare show that clinicians are more likely to trust and accept systems where they understand their inner workings (Markus, Kors & Rijnbeek 2021; Dave, Naik, Singhal & Patel 2020). Especially as transparency is expected from MDR/IVDR and EN ISO 14971. Developers ensure transparency through the Software Development Life Cycle (SDLC) and use it in software validation, documentation, and maintainability. Clinicians, on the other hand, use transparency, for example, to understand, evaluate, and trust the software’s functionalities, make informed decisions, and ensure accurate data processing (U.S. Food and Drug Administration 2024).

However, ethical questions arise when developers use AI-assisted coding tools. Alongside reliability concerns, safety risks, and intellectual property concerns on AI-assisted coding tools, to what extent are developers ethically responsible for AI-caused problems/bugs? When coding or developer decisions are influenced by AI-assisted coding tools, where does the use of these tools officially start and where does it end? And at what point should it be disclosed? How ethically responsible should developers be for maintaining their coding competence, judgment, and preventing over-reliance on AI-assisted coding tools? Additionally, when clinicians decide to incorporate systems that were developed with the use of AI-assisted coding tools, they face several ethical

questions. Should clinicians trust systems developed with the use of AI-assisted coding tools? Especially if they do not know the extent of AI involvement? To what extent should clinicians trust the medical output of an AI-built system, given the Hippocratic Oath? How ethically responsible should the clinician be if an AI-built system mispresents patient results? Should clinicians disclose that their system was built with AI-assisted coding tools? And will disclosure cause concerns and weaken patients' trust? These questions are not only theoretical; however, they additionally directly relate to patient safety, professional integrity, and public trust in the application of AI-assisted coding tools in healthcare.

2.4 Summary of Theoretical Frameworks

AI-assisted coding tools in Software Development change the role of developers in software projects. Developers by entering a collaborative state with these tools, expect opportunities such as increased efficiency and faster development speeds, leading to these tools becoming rapidly more common, however, they additionally carry risks and challenges such as concerns about the quality, safety, and maintainability of AI-generated code, that need to be carefully evaluated, especially when ap-plying these tools in regulated healthcare environments.

To ensure oversight and structurization of healthcare software development, the EU/EEA has adopted several regulations, such as MDR, IVDR, GDPR, and regulatory standards such as EN IEC 62304, EN ISO 13485, and EN ISO 14971. These regulations and standards collectively influence coding and QA practices by directing how healthcare software must be developed, validated, documented, and maintained, and defining requirements for software lifecycle processes, traceability, risk management, data protection, and quality assurance.

For developers, quality assurance not only serves as a tool to achieve compliance with legal frameworks, however, additionally as a set of internal company processes that enhances the control and understanding of the development processes. Developers to achieve compliance with regulatory frameworks and address the risks of using AI-assisted coding tools in healthcare software environments must follow rigorous quality assurance practices, including static analyzers, quality analysis of AI-generated code, and enhanced validation and documentation. As a result of using these tools can introduce uncaught flaws that are invisible to developers.

Concurrently, transparency and trust remain crucial for software systems developed with AI-assisted coding tools, as healthcare professionals and regulators require not only safe and reliable systems, however, additionally transparent, easy-to-understand, and justifiable systems, as they may have an impact on human health. Together, these perspectives reveal a need for a structured and systematic framework that can guide developers and quality assurance specialists to remain in

control when using AI-assisted coding tools in European healthcare environments. The next chapter is an empirical investigation, exploring though semi-structured interviews and a comparison prototype experiment, the importance of QA practices, the impact of AI-assisted coding tools, adaptations needed, and comparison between manual and AI-assisted coding.

3 Empirical Part

The empirical part of this thesis focuses on investigating how AI-assisted coding tools influence coding and QA practices in European healthcare software development. The research target of this section is to generate empirical insights that contribute to the overall aim of this thesis: creating a practical framework that developers can use to responsibly use AI-assisted coding tools in the European regulated healthcare environment.

This was done through the two primary objectives of this section:

- Conduct Semi-structured interviews with professionals working in software development to collect practical viewpoints on the benefits and risks associated with AI-assisted coding tools, the influence of European regulations on coding and QA practices, and adaptations viewed as necessary to use AI-assisted coding tools.
- Conduct a prototype comparison experiment by developing cardiovascular report generation software in two versions, one with the use of traditional coding practices and one with the use of AI-assisted coding tools, to test the practical implementation of those tools and evaluate the differences in code quality, test coverage, and development efficiency.

Together, the results from these objectives will be combined with theoretical frameworks to create conclusions and recommendations.

3.1 Research Methods and Justifications

A mixed-method research approach was chosen for this thesis, combining qualitative and quantitative research methods. This design allows for exploring both professionals' viewpoints and a comparative experiment of how AI-assisted coding tools affect coding and QA practices in European healthcare software development. According to Ahmed, Pereira and Jane (2024) mixed-method research approach of qualitative and quantitative methods can be used to deepen the understanding of complex phenomena and offers an enhanced research process. By the combination of these methods, researchers can cover areas that each method alone misses, and therefore, it is suitable for research questions that require insights from different perspectives. Alternative approaches were considered, such as surveys, however, were not selected as they provide less contextual data compared to interviews, needed for the thesis objective.

Semi-structured interviews were conducted as a qualitative method, because they offer a flexible and concurrently structured approach that allows researchers to gain in-depth information and understanding from interviewees staying on track with the research topic, with the possibility of asking spontaneous follow-up questions (Mashuri, Sarib, Alhabsyi, Syam & Ruslin 2022). This approach is

well suited for complex and context dependent topics such as regulatory compliance challenges, different perspectives, and professional practices in healthcare software development. Compared to an unstructured approach with no predefined questions.

To complement the qualitative method, a quantitative method of a prototype comparison experiment has been conducted. Experimental approaches allow for careful examination under controlled conditions and allow for a comparative evaluation of different approaches (Juristo & Moreno 2013; (Sjoeberg & al. 2005). Making it an ideal approach to evaluate and compare the influence of AI-assisted coding tools on code quality, test coverage, and development efficiency between the two software prototypes. This method was selected over observational study and case study due to their lack of controlled conditions needed for the comparison between AI-assisted and traditional development code outcomes.

3.2 Implementation of Semi-Structured Interviews

The implementation of semi-structured interviews in this research consisted of planning and validation of the interview guide, recruiting and selecting interview participants, conducting and recording of the interviews, and analyzing the interview results.

3.2.1 Progression of the Semi-Structured Interview Study

The planning of the interviews started with consultation and creation of interview questions based on the thesis research questions and theoretical frameworks relevant to Coding and QA practices, European healthcare, AI-assisted coding, and potential adaptations. This resulted in an interview guide document that was validated by academic experienced professionals and can be found in Appendix 1.

The interview participants were selected from researcher's network using purposive sampling. The aim of this selection was to find industry professionals with backgrounds in software development, European healthcare or medical devices, and those that can comment on QA and regulatory compliance. This group included professionals with experience using AI-assisted coding tools. The selection resulted in 6 participants. The target profiles included:

- Roles relevant to regulatory and QA compliance (CTO, QA manager, product owner)
- Software Developers working in healthcare or medical device projects

Selection criteria included:

- At least one year of experience in software development in a relevant field
- Past or present Involvement in projects that required European healthcare regulation

The interviews were anonymous and conducted online (Microsoft Teams and Google Meet) to ensure time and location flexibility for interviewees. Each interview lasted from 30 minutes to an hour depending on the participant's availability and depth of conversation. During the interview participant approved notes were taken, with follow-up questions to elaborate and deepen participant answers.

3.2.2 Interview Guide

The interview guide was constructed to be semi-structured and lead the interview though its main themes additionally, allowing the interviewer to adjust the order depending on the conversation and ask elaboration and follow-up questions. The Interview Guide main themes include:

1. Background
    a. Role and experience in software development
    b. Project examples relevant to regulatory compliance and healthcare
2. Coding and QA Practices
    a. Typical coding workflow
    b. Most important quality assurance practices
    c. Big challenges in ensuring compliance with healthcare regulations
3. AI-Assisted Coding
    a. AI-assisted coding tools usage extent
    b. Potential benefits behind using AI-assisted coding tools
    c. Potential risks behind using AI-assisted coding tools
4. Adaptations and Future Outlook
    a. Adaptations needed in coding and QA practices for wider adaptation of AI-assisted coding tools at work
    b. Guidelines, checklists, and regulations recommendation
5. Closing
    a. Additional comments and experience about coding, QA, or AI-assisted coding tools

3.2.3 Ethical Considerations

To ensure ethicality of the interview process several measures have been applied:

The participation is voluntary, and the participants have been informed in advance about the purpose of the research, how their data will be used, their right to withdraw, and their anonymity. Participation consent forms Appendix 2. and participant information sheets Appendix 3. have been collected according to Haaga-Helia guidelines and kept in a secure location. In case of withdrawal from the participation, all data will be removed if still technically feasible.

Interviews avoided collecting unnecessary data and any data collected was transcribed and stored in a password-protected location in researcher's personal computer and backed up in cloud storage provided by Haaga-Helia. All participant specific information was aggregated or paraphrased when transcribed to ensure anonymity of the individuals.

No sensitive information other than professional background and role was intentionally collected. The study follows Haaga-Helia ethical guidelines.

All interview data such as notes and transcripts, and consent forms will be permanently deleted 3 months after the thesis has been accepted and graded. Only anonymized and rephrased fragments included in the thesis will remail as part of permanent research record

3.2.4 Analysis

After the interview phase, findings have been manually checked for correctness, transcribed, and imported to a text editor tool. Data from the six interviews was subjected to a thematic analysis flowing the principles of (King & Brooks 2017). Because the interview guide was structured around five predefined themes, such as: Background, Coding and QA Practices, AI-assisted Coding, Adaptations and Future Outlook, and Additional Observations, the transcripts were organized deductively. This approach allowed consistency in statement comparison, pattern identification, and flexibility to incorporate predefined themes derived from Theoretical Frameworks.

Within each interview statement, its main ideas were highlighted and assigned an interpretive code that was later grouped up in a matrix. With rows representing the main themes and columns representing the anonymous participants, the researcher was able to compare all statements within x and y axis as well as individual boxes. The similarities and recurring ideas were subsequently grouped into sub-themes that described shared perspectives. The defined sub-themes were confirmed with raw statements to ensure their accuracy and minor adjustment were made to manage overlapping ideas. The analysis was finalized by writing down findings for each theme including main pattern of the theme, key ideas (both shared and individual), and a summary of the theme.

3.3 Implementation of the Prototype Experiment

The implementation of the Prototype Experiment in this research was conducted by developing a cardiovascular report generation system in two versions. The development of the two versions included panning mutual requirements and setting up the environment, two separate development phases, one conducted manually and one conducted with use of AI-assisted coding tools, by the researcher. A comparative analysis between the two versions focusing on code quality, test coverage, and development efficiency.

3.3.1 Progression of the Prototype Experiment

The prototype experiment started with planning and setting the base requirements, resulting in the decision to develop a cardiovascular report generation system due to its lightweight, low risk, medical aspects, the base requirements were set to meet with the research objectives and theoretical frameworks defined earlier in this thesis.

The requirements were:

- The system generates its own patient mock data in a JSON format with vitals such as CCA (psv_cm_s, edv_cm_s, imt_mm), ICA (psv_cm_s, edv_cm_s, imt_mm), and ECA (psv_cm_s, edv_cm_s).
- The system inputs the JSON data, analyses the data, and generates a HTML/PDF document containing patient data and diagnosis.
- The system contains unit tests for all report generation modules

The development was conducted twice using identical requirements. The Manual coding implementation was carried out first by the researcher, consequently its development was not affected by the AI-Assisted coding implementation logic. The AI-Assisted coding implementation was carried out with use of ChatGPT and Copilot, prompts were used to generate modules, structure, documentation, and tests. Adjustments were made only where necessary to ensure functional and design correctness, the initial AI prompt can be found under Appendix 4. As result, two functionally comparable, however, structurally different codebases were produced, allowing a fair comparison of Reliability, Security, Maintainability, test coverage, and development efficiency.

Both versions were created with VS Code 1.105.1, Python 3.12.7, pytest 7.4.4, weasyprint 66.0, Jinja2 3.1.4

Both implementation versions can be found in researchers GitHub repository:
https://github.com/Blazej3/cardio_report_experiment

The data collection proceeded with the developer (researcher) measuring time of completion for each implementation and using a static analysis tool SonarQube. This tool was chosen because it is research validated platform for detecting maintainability issues, code smells, security hotspots, and test coverage metrics in software engineering projects. Prior research indicates that numerous companies and developers constantly rely on static analysis tools such as SonarQube to detect defects and enforce coding standards. Additionally, Guaman, Quezada-Sarmiento, Barba-Guamán, Cabrera and Enciso (2017) demonstrate that SonarQube is one of the most used tools in industry and academia for evaluating code quality due to its high-quality models and well-defined metrics, that link its evaluation criteria to software quality frameworks such as ISO-9126 and

SQALE. On top of metrics collected during the implementation phase, the researcher collected personal observations seen as noteworthy.

3.3.2 Ethical Considerations

To ensure ethicality of the experiment process several measures have been applied. The experiment did not use any sort of patient data and only random synthetic mock data was used. The software was explicitly defined as non-medical to prevent misinterpretation for medical diagnosis/advice software. The AI-assisted coding was transparently documented, and all AI generated code was manually verified to ensure human oversight in line with the EU AI Act. The analysis phase for both implementations was conducted identically to ensure fair and controlled comparison to avoid bias. Additionally, all methodological choices were documented to ensure reproducibility, transparency, and academic integrity.

3.3.3 Analysis

The individual implementation analysis was conducted by first setting up a local SonarQube server deployed using Docker (sonarqube:lts) and accessed via http://localhost:9000, with each implementation containing a separate sonar-project.properties file, defining a unique sonar.projectKey, project name, source paths, and coverage report paths, this ensured SonarQube treated the implementations as two separate projects. For each implementation the scanner was executed from project root directory using the official SonarScanner CLI container, figure 4. Coverage reports were generated identically using command represented by figure 5

```
C:[redacted]> docker run --rm \
  -e SONAR_HOST_URL="http://host.docker.internal:9000" \
  -e SONAR_TOKEN="TOKEN_HERE" \
  -v "${PWD}:/usr/src" \
  sonarsource/sonar-scanner-cli
```

Figure 4. Docker command used to run the SonarScanner CLI container

```
pytest --cov=. --cov-branch --cov-report=xml --cov-report=term
```

Figure 5. Pytest command used to run tests with branch coverage

The resulting coverage.xml file for each project were automatically imported by SonarQube. Successful ingestion was confirmed via SonarQube dashboard, where code quality, and test coverage metrics were displayed for each project independently. SonarQube analysis for Manual implementation is represented by figure 6. and analysis for AI-Assisted implementation is represented by figure 7.

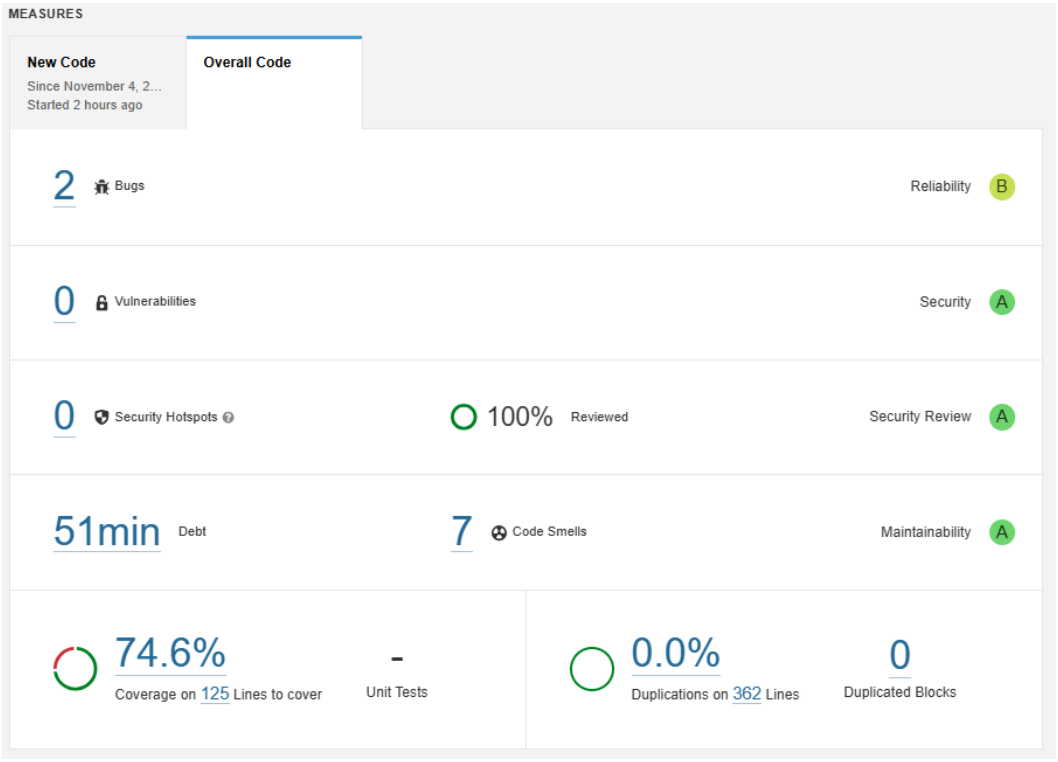

Figure 6. SonarQube overall code quality dashboard summarizing metrics from the manual implementation.

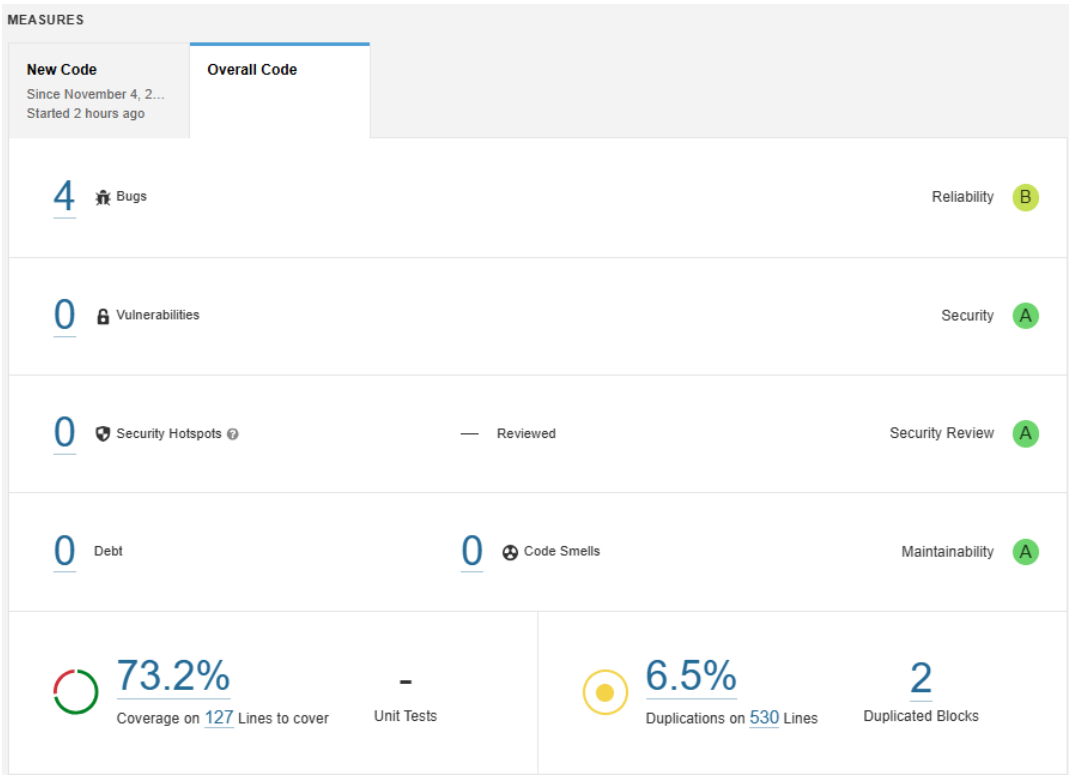

Figure 7. SonarQube overall code quality dashboard summarizing metrics from the ai-assisted implementation.

It is essential to mention that in the manual implementation, SonarQube reported two security hotspots related to the use of Python's random module for generating synthetic patient attributes (age_years and sex). These were automatically flagged under rule python: S2245, which warns against pseudorandom number generators in security-critical contexts. In this implementation, the random values were sole used for mock patient data generation in a controlled non-production environment. This required a human review, which classified both risks as safe. Additionally, the data generation module was not subjected to any tests because it does not contribute to the core behavior of the report generation system. However, it was still subjected to static analysis and test coverage reporting, because SonarQube evaluates structural quality of the entire codebase, and not functional modules, which makes it appropriate for collecting maintainability and security data.

3.4 Empirical Results

This section presents empirical results from semi-structured interviews and the comparison prototype experiment conducted in this thesis. The purpose of this section is to present results without interpretation. These results are later discussed and connected to theoretical frameworks in Chapter 4.

Table 1. Findings from Semi-Structured Interviews

Theme 1: Professional Background and Experience
Pattern: All participants have over one year experience in various fields such as: research-based ML engineering (P1), long-term health-tech product development (P2), backend development (P3), feature planning and implementation (P4), full-stack and product management (P5), and test management in QA (P6). **Key Insights:** Regulatory experience increases with seniority; juniors rely more on leadership for compliance. **Summary:** Participants come from various backgrounds, years of experience, career stages, and positions.
Theme 2: Coding and QA Practices
Pattern: All participants follow some form of SDLC model adaptation to their projects' needs (often requirement analysis, design, implementation, and testing), with tests and reviews being central to QA practices. **Key Insights:** Testing (e.g. manual, unit, robustness, functionality) and reviews/pull request (e.g. peer review) are seen as essential in quality assurance. Documentation is crucial in regulated environments (P4, P5). Challenges include regulatory communication (P1), ambiguous QMS requirements (P2), and multi-layered documentation and requirements mapping (P1, P2, P5).

Summary: QA precatives can vary depending on the project; participants expressed importance of testing and documentation; however, regulation requirements can cause extra complexity and increased workload.
Theme 3: AI-Assisted Coding and Its Impact
Pattern: Five out of six participants have used AI-assisted coding tools for coding; however, the level of use varied from light assistance (P3) to integrated daily use (P1, P4, P5, P6). **Key Insights:** The perceived benefits included speed and efficiency in routine tasks (P1, P4, P5, P6), quality improvements for simple tasks (P2, P6), knowledge support (P3), and compliance potential if tools could “understand the QMS” (P1, P3, P5). The perceived risks included code security and data leakage during prompting (P3, P4, P6), overreliance and quality loss (P2, P5), lack of explainability (P3, P6), and limited compliance model awareness (P1, P6). **Summary:** Ai-assessed coding tools that can understand QMS can be a huge benefit. Ai-assessed coding tools are better for simple tasks than complex ones. Multiple developers are cautious about overreliance in AI. Several participants agree that AI should assist and not replace human rezoning. Use of AI-assisted coding tools can speed up coding, however, can additionally put more pressure on QA, leading to no change in productivity.
Theme 4: Adaptations needed for QA and Coding Practices
Pattern: Participants expressed, adopted QA system, developer training, and organizational control requirements for wider AI adaptations. **Key Insights:** Increased reviews and tests for AI-generated code and logic (P1, P3, P4) Smarter and broader QA coverage using AI itself (P2) Implement data governance policies (what data can be shared with AI) (P3, P5) Developer awareness of responsible use of AI-assisted coding tools (P1, P3, P4, P5, P6) **Summary:** Multiple participants emphasized the importance of understanding proper use of AI-assisted coding tools. QA should remain a stable part of the AI workflow. Implementation of internal data hierarchy (which files can be used with AI). Reviewers must use a structured review system and frequently review logic and code created with AI. Human oversight is necessary (Human-in-the-loop) in the whole SDLC, especially in the review stage.

A comparative prototype experiment was conducted to evaluate the differences in code quality, test coverage, and development efficiency between manual coding and AI-assisted implementation of a “Cardiovascular Report Generation System”. The key results are represented by table 2.

Table 2. Findings from Prototype Experiment

Code Quality	Manual Implementation	AI-Assisted Implementation
Bugs	2	4
Vulnerabilities	0	0
Security Hotspots	0 (two false positives)	0
Code Smells	7 (~51 min to fix)	0
Duplications on Lines	0.0% on 362 lines	6.5% on 530 lines
Test Coverage[1]	74.6%	73.2%
Development Efficiency	19 hours	9 hours
Additional observations		The developer experienced higher sense of trust in the tool and possibility to clarify/advice on ideas.

[1]Test coverage excludes the data generation module in both cases

3.5 Summary of Empirical Insights

The empirical findings of semi-structured interviews and the comparative prototype experiment reveal several key findings. The interviews showed that all participants regardless of background already follow structured coding practices, with testing, documentation, and reviews being central QA activities. Participants viewed AI-assisted coding tools as beneficial for routine tasks and early/small implementations, with speed and efficiency gains, however, highlighted potential risks with data security, explainability, and potential overreliance. Additionally, they emphasized the need for increased reviews, stronger QA processes, and clear organizational guidelines (particularly with data governance), when integrating AI-assisted coding tools into regulated environments.

The comparative prototype experiment supported these observations. AI-assisted development reduced implementation speed by more than 50%, however, produced more bugs and a higher level of code duplication. In contrast, the manual implementation produced a cleaner structure, and SonarQube reported two security hotspots which were identified as false positives in an additional manual review. Both implementations had similar test coverage and reported no vulnerabilities, with the manual implementation having more code smells with an estimate of 51 min to fix, additionally during the development of the manual implementation developers experience a higher sense of trust in the AI-assisted coding tool. These results suggest that AI-assisted coding tools accelerate small scale development (Potentially forming advantages for incremental development strategies.) and shift the workload to review, verification, and refactorization.

Together these empirical insights present both potential advantages and disadvantages in using AI-assisted coding tools and combined with theoretical frameworks from Chapter 2, form a foundation for the proposed framework presented in Chapter 4, where a deeper discussion and interpretation of them is provided.

4 Discussion and Conclusion

This chapter interprets empirical findings collected in Chapter 3 and explains their relationship with theoretical frameworks and regulatory requirements presented in Chapter 2. By the combination of interview findings, prototype experiment findings, and theoretical frameworks this chapter identifies key adaptations when using AI-assisted coding tools in European Healthcare environments.

4.1 Discussion of Findings

This discussion is structured around research questions from Section 1.5 focusing on addressing benefits and risks of AI-assisted coding, influence of European healthcare regulations, adaptations needed to ensure responsible AI use, and how does AI-assisted coding compare to conventional development.

4.1.1 Benefits and Risks of AI-Assisted Coding

Interview participants and coding experiment results highlighted several benefits and disadvantages consistent with literature on AI in software development. Majority of interview participants expressed benefit in using AI-assisted coding tools for speed and efficiency in routine tasks, such as debugging and documentation generation aligned with resent study on AI impact on developer productivity (Peng et al. 2023; Chen & al. 2025). Additionally, this was confirmed by the coding experiment where AI-assisted development reduced implementation by more than 50% compared to the manual implementation speed. The interview results additionally showed that these tools can work as learning support for junior developers as shown by participant interest and use of AI-assisted coding tools only for those purposes. This stands with the additional observations generated by the coding experiment where the developer conducting the experiment observed the opportunity of asking the tool for additional clarity or advice on coding. This finding is confirmed by the statements made in the study by Ferino et al. (2025).

Despite these benefits empirical data revealed multiple substantial concerns that align with theory associated with AI-assisted coding tools. Interview participants highlighted concerns about data leakage, security, and uncertainty about what information can be used in the prompts. These issues are directly linked to GDPR requirements and highlighted by Licorish et al. (2025) that emphasized the possibility of over-engineered, incorrect, or disregard coding standards code. Additionally, participants reported worries about hallucinations, limited explainability, unpredictable logic, and overreliance supported by research on vulnerabilities and maintainability of AI-generated code (Perry e al. 2023; Abbassi et al. 2025). The coding experiment reinforced these issues, although AI-assisted implementation was nearly twice as fast, it produced more bugs and had a

higher level of code duplication, indicating that the price of initial coding speed impact the speed of review and correction.

Another recurring theme during interviews was overreliance on AI tools that can impact developers' ability to reason and decrease competency. This was observed in the coding experiment, even though it produced more faults the developer reported higher sense of trust in the tool which could result in overreliance. This concern was noticed by Perry e al. (2023) where developers created security vulnerabilities using AI-assisted coding tools convinced that they wrote secure code.

Collectively, these findings highlight the two-sided aspect of using AI-assisted coding tools in European regulated healthcare environment. While they increase developer productivity, speed, and support learning, they additionally introduce reliability, security, oversight, and overreliance risk challenges.

4.1.2 Influence of European Healthcare Regulations

Empirical findings show a significant relationship between European regulations and software development, especially in healthcare environments. Interview participants working in Health-IT emphasized the role of documentation, testing, code reviews, step-by-step workflows, and traceability in shaping daily development work. These practices are closely related to regulations and standards outlined in Chapter 2, including MDR/IVDR, IEC 62304 and ISO 13485. The most brought up topic in terms of importance were verification practices, particularly performing tests, and reviews. Interview participants often mentioned the importance of testing and review activities in meeting regulatory expectations. Functional, automated, robustness, and edge case testing, with pull request reviews, and peer review practices were often performed by the participants and described as "most important". These findings reflect the verification and validation principles required under IEC 62304 standard for software lifecycle processes for software used in medical devices.

Additionally, an often repeated view among the participants was the intensity of documentation and traceability requirements. Participants described detailed documentation as being an important and challenging part of their daily work and ensuring compliance. This aligns with the IEC 62304 standard which requires complete documentation of design, implementation, verification, and change history throughout the software lifecycle. Some participants noted that documentation becomes much heavier in regulated environments such as healthcare, especially as every feature and requirement must be mapped. Furthermore, participants described their coding workflow to be clear and systematic, often including requirement analysis, design, implementation, and testing, which flow the lifecycle requirements in standards such as IEC 62304 and ISO 13485.

A further observation form interview results suggest some participants experience practical challenges of navigating regulatory requirements. One participant described difficulties for companies to communicate with regulation institutions, noting that responses can be slow or inconsistent. Another participant stated having experience numerous “gray spots” in regulations as numerous are not definitely defined, complicating interpretation and implementation. A Third participant highlighted that when multiple regulations apply to the same product forming a “regulation matrix” navigation between them becomes increasingly complicated. These findings suggest that some developers do not potentially struggle with regulations themselves, however, rather with their interpretation and receiving clear guidelines.

Interview participants did not particularly mention related to risk management or data governance; this does not imply that participants didn’t see any value in those topics, however, rather that they preferred to focus on more visible aspects such as documentation, resting, or review. It may additionally reflect that often risk management or data governance practices are carried out by dedicated regulatory or QA professionals. Additionally, interview results showed that regulatory experience increases with seniority and juniors rely more on leadership for compliance.

Taken together, these findings reflect that European regulations substantially influence coding and QA practices by requiring deep documentation, structured workflows, systematic testing, and traceability. Concurrently, developers experience uncertainties caused by regulative ambiguity and dependance on senior team members.

4.1.3 Adaptations Needed to Ensure Responsible AI Use

Empirical findings collected in this research indicate that the integration of AI-assisted coding tool into healthcare software development would greatly benefit from special adaptations to coding and quality assurance practices. One of the most referenced adaptations by interview participants was the need for extended and more righteous review processes. Participants repeatedly highlighted that work created with AI-assisted coding tools “needs more reviews”, “puts pressure on reviews”, and may “behave in unexpected ways without human oversight”, consequently suggesting that some AI-generated work should be reviews differently than manual work. Results from the coding experiment confirm these statements as AI-assisted implementation produced more bugs and duplications that were not noticed by the developer before the review process. Standards such as the IEC 62304 and EU AI Act indirectly support these statements by requiring risk-based verification and adequate human review for any software that introduces potential risks and High-Risk AI systems.

Moreover, to ensure reviewability of all decisions and code created with use of AI-assisted doing tools, would require additional transparency measures specifically directed for AI. According to recent literature, immediate implementation of advanced transparency practices would allow for adherence to internal and external requirements (Watson and Van Italie (2025). Additionally, interview results revealed that one participant recommended making QA as a stable part of any AI tool interaction. Although they did use the term transparency explicitly, this recommendation inherently implies it, as maintaining transparency is a stable part of QA practices. This suggests that transparency measures for code and work created with use of AI-assisted coding tools would benefit from labeling which parts were and to what extent were influenced by these tools which are in accordance with Kashif et al. (2025).

Another adaptation identified by participants concerns is data governance and secure prompting. Most participants expressed concerns about “security”, “data sharing when prompting”, “sensitive data leakage”, these issues would potentially be caused by accidental leakage of personal or safety-critical information through AI prompts, and some participants suggested implementing “internal hierarchy of files that can be shared”. Although not found in the coding experiment, these suggestions align with integration of government policies and intellectual property management procedures recommend for organizations by Watson and Van Italie (2025). Additionally, Xu et al. (2025) and King et al. (2025) support this claim by highlighting that by using correct prompts can cause the AI-assisted coding tools can generate license-protected code and potential data leakage though prompts. This issue potentially violates the GDPR’s requirements for data minimization and secure processing.

A further observation confirmed the importance of human oversight and human-in-the-loop strategies throughout the while SDLC. According to participants AI-assisted coding tools in their work environment “should assist, not replace” developers and developers should always fully understand the product of their work. Which was experienced in the coding experiment which required occasional adjustments to the generated code during development and a human review. These observations directly align with the EU AI Act’s requirements for meaningful human oversight. Consequently, deducing that all AI decisions affect functionality, safety, logic, data handling, or architecture must be thoroughly reviewed and fully understood by a human developer, who remains responsible for the outcome. (European Commission 2025; Takerngsaksiri & al. 2025.)

Finally, Empirical findings revealed the importance of developer training and organizational guidance. Participants repeatedly highlighted the need for “learning proper use of AI”, “developer awareness”, and “company internal guidelines for using AI”. An investment in education and training (for staff and organization) to increase understanding of AI tools and transparency practices is

recommended by Watson and Van Italie (2025). These observations match the 13485's competence requirements and highlight that safe use of AI integration depends not only on tools, however, additionally on organizational maturity and staff capabilities. Ultimately suggesting allowing only developers that undergo internal/external training to use these tools would be beneficial and contribute to responsible and aware use of AI-assisted coding tools.

In summary, the empirical findings show that for safe adoption of AI-assisted coding tools into European healthcare development requires adaptations in review processes, testing practices, data governance, human oversight, and developer training. These findings are supported by interview participant responses, the coding experiment outcomes, regulatory and theoretical requirements.

4.1.4 How Does AI-Assisted Coding Compare to Conventional Development

The prototype experiment demonstrated clear differences between conventional (manual) coding and AI assisted coding in terms of coding quality and development efficiency, however, test coverages were highly similar across both cases. The AI-assisted implementation was completed in 9 hours compared to the manual implementation that took 19 hours, supporting interview results that claimed AI-assisted coding tools can significantly accelerate and enhance development processes and quality improvements for simple tasks. This is in accordance with research on the impact of AI on developer productivity that reveal that 55.8% developers with access to the AI pair programmer completed the test in the study faster than others (Peng et al. 2023). This finding confirms the clear benefit of using AI-assisted coding tools, however, additionally rases the question about its downsides, the coding experiment along with interview results introduces a concern, code quality.

The coding experiment resulted in SonarQube finding 2, 7 code smells that estimated would take 51 minutes, 2 false positive security hotspots and 0% code duplication in the manual implementation compared to the AI-Assisted Implementation which had 4 bugs, 0 code smells, 0 security hotspots, and 6.5% of line duplications. This underlines the possibility of AI-assisted code could introduce more vulnerabilities and faults showcased by Licorish et al. (2025) and Perry e al. (2023). Interview participants shared a similar view claiming that unsupervised or unreviewed code generated with these tools can cause additional issues. This as a result could cause more pressure on the QA processes as one participant in the interviews mentioned that using these tools "becomes a zero-sum game, due to the time you save on development you have to compensate in the QA", as the work shifts it calls for an immediate investment in AI oriented QA practices.

Test coverage was similar in both implementations (74.6% manual and 73.2% AI-assisted), both AI-assisted and manual created tests considered all vital functionalities. This finding is accurate to one interview participant, mentioning that one benefit of AI-assisted coding tools is test generation,

and potentially suggests that using AI-assisted coding tools has no effect on test coverage. These findings are supported by a study evaluating ChatGPT for unit test generation (in terms of test coverage) however, the research suggests AI-assisted coding tools may lack in execution, finding edge cases, and in some cases, simply ineffective (Yuan & al. 2024; Huang & al. 2025).

4.1.5 Discussion of Findings Conclusion

Overall, this comparison illustrates that the use of AI-assisted coding tools can accelerate development, however, puts more pressure on the review, test, and verification processes. Manual coding can be slower; however, it is more stable and easier to interpret. Code experiment results are supported by interview results that AI-assisted coding tools “should not replace human reasoning, and their effectiveness depends on developers using it and QA practices implemented”. These conclusions create a foundation for the framework proposed in chapter 4.2.

4.2 Proposed Framework

Based on empirical findings, theoretical and regulatory frameworks discussed in earlier chapters, this section proposes a framework to guide in safe, compliant, and efficient use of AI-assisted coding tools in European healthcare environments. This framework is designed to address key challenges identified through this study, including data governance, human oversight, review process, testing, traceability, and developer training. Its aim is to provide practical guidelines that complement existing practices and standards such as MDR/IVDR, IEC 62304, ISO 13485, ISO 14971, GDPR, and the EU AI Act. The framework is intended for developers and quality assurance specialists and is meant to support and not replace existing quality assurance processes.

1. **AI Usage and Data Governance Controls**

   Organizations should establish rules for which data can be shared with AI tools, including which files can be submitted and use of safe and secure prompting practices. The structure should include security designation of data and files and prohibition of sharing personal and patient information to AI tools.

2. **Human-in-the-loop review**

   All work created with use of AI-assisted coding tools must be verified in a human overseeing process before integration and must be linked with a person responsible for their review or implementation. Human reviews should verify logic, consistency with codebase, and adherence to the requirements.

3. **AI-assisted code review checklists**

   All code created with use of AI assisted doing tools should be reviewed with AI-specialized checklist that includes logic validation, dependency correctness, style consistency, hallucination mitigation, and regulatory adherence. The checklist must

integrated mandatory requirements from IEC 62304 (verification and validation) and ISO 14971 (risk controls).

4. **Enhanced testing for AI-generated code**

   Personnel responsible for tests must treat code created with AI-assisted coding tools with consideration that AI tools have impacted this code and generate tests according to the possible vulnerabilities of AI-generated codes. This includes explicit verification of AI-generated logic and bugs. AI tools may be used to generate tests given that they have been verified for correctness by a human.

5. **Documentation and transparency**

   Code created with a heavy dependence on AI-assisted coding tools must be labeled as such. All prompts and AI interactions should be recorded where feasible.

6. **Developer training and organizational guidelines**

   Developers, to utilize AI-assisted coding tools in their work must be required to have underwent training for responsible use of AI, regulatory risks, and secure prompting, with optional training for maximizing developer efficiency with AI tools.

   Organization should implement a basic set of internal guidelines outlining roles and responsibilities in terms of AI use.

4.3 Framework Justification

The proposed framework is justified based on semi-structured interviews and coding experiment conducted in this study, and the theoretical frameworks discussed in chapter 2. Each section of the framework is justified by challenges and requirements found throughout this study and aligned with existing requirements for safety, quality, and accountability in European healthcare software development.

AI Usage and Data Governance Controls are included in the framework in section 1, due to participants continuous expression of major concerns about data leakage and inappropriate information sharing during AI prompting. Additionally, Xu et al. (2025) and King et al. (2025) underline the risk of data leakage though prompts. The aim of this section is to address these issues by mandating adaptation of explicit data management rules for AI use, ensuring compliance with GDPR requirements.

Human-in-the-loop review section 2 is justified by both interviews, coding experiments and theoretical findings. Interview participants highlighted the need for human overuse and that AI tools should only complement developers. The coding experiment underlined the need for this section by making the developer feel a higher level of trust in the AI tools that could cause potential issues with

overreliance. This section is motivated by Takerngsaksiri & al. (2025) and the EU AI Act's to mitigate possible hidden errors, hallucinations, or unnecessary complexities.

AI-assisted code review checklists are included in section 3 because interview participants emphasized the need for structured and frequent validation of AI-generated logic. This section aims to decrease the complexity when multiple regulations influence the same development process and help developers reduce the challenges caused by AI-assisted coding tools. The implementation of a checklist would support consistent verification practices required by IEC 62304 and ISO 14971.

Enhanced testing for AI-generated code section 4 is justified by the code experiment that showed increased defects for AI-generated code and code duplications in the AI-assisted implementation. Additionally confirmed by Perry e al. (2023) and Licorish et al. (2025) this section aims to reduce the number of defects generated by AI-assisted coding tools, by treating all AI outputs correspondently.

Documentation and traceability section 5 is included because interview participants highlighted the regulatory importance of documentation in healthcare development, additionally helping other developers or reviewers understand the code more efficiently. In addition, this section is motivated by Abbassi et al. (2025) which highlighted the risk of poorly documented code by AI-assisted coding tools. This section aims to address these risks and support sections 1-4, for easier detecting AI-influenced work.

Section 6 Developer Training and organizational Guidelines is included because interview participants highlighted the need for properly understanding AI usage practices and the risks of overreliance, particularly among less experienced developers, and the need for organization rules for responsibilities. This section aligned with studies suggestions about developer training from Watson and Van Italie (2025). Additionally structured training supports competence requirements under ISO 13485 and promotes responsible AI usage.

4.4 Limitations

This study has introduced several limitations that should be considered when interpreting the results and applying the proposed framework. Firstly, the semi-structured interviews were conducted with a limited number of participants (n=6). Although the interviews focus on quality over quantity, they may not capture the full range of perspectives across the European healthcare software industry.

Secondly, the coding experiment was conducted by the same developer in a short time span in between in implementation. The Manual implementation was performed first which could have

obscured the AI-assisted implementation which was second with information the developer learned from the first implementation.

Thirdly, the cardiovascular report generation system used in the experiment was a non-medical prototype developed for research purposes only. The system was not intended for clinical decision making or diagnostic functions, therefore the findings cannot be compared to high-risk medical device software without further validation. Fourth, the study focused on commonly available AI-assisted coding tools at the time of writing. Rapid advancements of these tools and regulations may affect specific findings over time. Finally, the framework which resulted from this study was not validated in an industrial setting.

4.5 Conclusion

This thesis examined how developers can adapt their coding and quality assurance practices when using AI-assisted coding tools in European regulated software development environments. Through a mixed-methods approach, this thesis combined semi-structured interviews and a comparative coding experiment. The study explored the benefits and risks behind AI-assisted coding, influence of European regulations on modern software development in healthcare, adaptations necessary to use AI-assisted coding and maintain safety, transparency, reliability, and compliance, how can adaptations be put together into a usable framework that developers can use, and comparison of AI assisted coding and conventional coding in terms of code quality, test coverage, and development efficiency.

The findings show that AI-assisted coding offers clear benefits in European healthcare software development, most popular benefits are associated with increased development speed, support for routine tasks, and knowledge assistance, while additionally introducing risks related to reliability, explainability, data security, and overreliance. Additionally, the findings revealed that European regulatory frameworks such as MDR/IVDR, IEC 62304, ISO 13485, ISO 14971, GDPR, and the EU AI Act strongly influence healthcare software development processes. By enforcing structured lifecycles, extensive documentation, traceability, risk management, and data governance, they shape coding and QA practices.

The study determined that safe, transparent, reliable and compliant AI-assisted coding requires adaptations including strengthened human-in-the-loop review processes, AI aimed testing, secure data governance, clearer documentation practices, and developer training. These adaptations were put together into a framework in section 4.2 that utilizes regulatory and empirical requirements for practical adaptations developers can integrate into existing quality management systems. The comparative coding experiment demonstrated that AI-assisted coding tools increase

development efficiency, however, risk producing more defects and structural issues than conventional coding, while test coverage remained comparable, this highlighted the need for increased review and quality assurance efforts when AI-assisted coding tools are used.

Overall, this thesis contributes an empirically grounded and regulation-aware perspective on using AI-assisted voting tools in European healthcare software development environment. By combining theoretical frameworks with practical insights into a framework, this study supports developers and organizations in adopting AI-assisted coding tools responsibly without compromising safety or compliance. While AI-assisted coding offers significant increases in development speeds, this work demonstrates that these advantages must be balanced out by straightening quality assurance practices. The proposed framework forms a ground for future research as AI tools and regulations continue to evolve.

4.6 AI Usage Disclaimer

The thesis has used chatgpt.com, keenious.com, and mybib.com to support Information and document search and analysis, planning of the thesis structure, development of interview questions, and selected parts of the AI-assisted coding experiment. In addition, Microsoft Word’s built in AI accessibility features were used to generate alternative text for tables and figures. AI tools were further used for reference data retrieval and assisted with dictionary use and grammar correction in the English Language. All reference information was verified by the author against the original source.

For example, the questions "How can the research objective of the thesis xxx be further delimited and broken down into sub-objectives?" or “Provide a list of sources alongside with links to the original documents related to QA and the use of AI-assisted coding in healthcare in European environments” have been used as prompt. All AI generated content has been further refined to make it error free, relevant, clear and understandable. The AI applications have been used responsibly, considering data protection and copyright. All sources cited in the report have been used correctly and are not AI generated.

4.7 Recommendations for Future Work

Future research should evaluate the effectiveness of the framework in real-world healthcare software development projects. Additional studies could involve a higher number of participants and a real-world medical coding experiment, to improve generalizability of the European healthcare software development market. Further work is additionally recommended beyond Europe, diving into other markets such as USA and the FDA regulation.

Sources

Abbassi, A.A., Silva, L.D., Nikanjam, A. & Khomh, F. 2025. A Taxonomy of Inefficiencies in LLM-Generated Python Code. 2025 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 393–404. URL: https://doi.org/10.1109/ICSME64153.2025.00043.

Ahmed, A., Pereira, L. & Jane, K. 2024. Mixed Methods Research: Combining Both Qualitative and Quantitative Approaches. ResearchGate. URL: https://www.researchgate.net/publication/384402328_Mixed_Methods_Research_Combining_both_qualitative_and_quantitative_approaches. Accessed: 22 October 2025.

Becker, J., Rush, N., Barnes, E. & Rein, D. 2025. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. URL: https://doi.org/10.48550/arxiv.2507.09089.

CENELEC 2018. European Standards (EN IEC). URL: https://boss.cenelec.eu/fadel/pages/eniec/pages/. Accessed: 10 September.

Chen, V., Zhu, A., Zhao, S., Mozannar, H., Sontag, D. & Talwalkar, A. 2025. Need Help? Designing Proactive AI Assistants for Programming. CHI '25: Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems, New York, pp. 1–18. URL: https://doi.org/10.1145/3706598.3714002.

Commission Implementing Decision (EU) 2022/757 of 11 May 2022 amending Implementing Decision (EU) 2021/1182 as regards harmonised standards for quality management systems, sterilisation and application of risk management to medical devices. Official Journal of the European Union, L 138/27–29, 17 May 2022.

Commission Implementing Decision (EU) C(2021) 2406 final of 14 April 2021 on a standardisation request to the European Committee for Standardization and the European Committee for Electrotechnical Standardization as regards medical devices in support of Regulation (EU) 2017/745 of the European Parliament and of the Council and in vitro diagnostic medical devices in support of Regulation (EU) 2017/746 of the European Parliament and of the Council. European Commission. Official Journal of the European Union (mandate document; not published in OJ), Brussels, 14 April 2021.

"Dave, D., Naik, H., Singhal, S. & Patel, P. 2020. Explainable AI meets Healthcare: A Study on Heart Disease Dataset. URL: https://doi.org/10.48550/arXiv.2011.03195.

European Commission 2019. Ethics Guidelines for Trustworthy AI. URL: https://digital-strategy.ec.europa.eu/en/library/ethics-guidelines-trustworthy-ai. Accessed: 12 October 2025.

European Commission 2025. AI Act. URL: https://digital-strategy.ec.europa.eu/en/policies/regulatory-framework-ai. Accessed: 12 October 2025.

European Parliament 2025. Healthcare sector: addressing labour shortages and working conditions. URL: https://www.europarl.europa.eu/news/en/agenda/plenary-news/2025-02-10/14/healthcare-sector-addressing-labour-shortages-and-working-conditions. Accessed: 1 Spetember 2025.

European Union s.a. Harmonised Standards. URL: https://single-market-economy.ec.europa.eu/single-market/goods/european-standards/harmonised-standards_en. Accessed: 30 September 2025.

Ferino, S., Hoda, R., Grundy, J. & Treude, C. 2025. Junior Software Developers' Perspectives on Adopting LLMs for Software Engineering: a Systematic Literature Review. URL: https://arxiv.org/abs/2503.07556v1.

Garouani, M., Mothe, J., Barhrhouj, A. & Aligon, J. 2024. Investigating the Duality of Interpretability and Explainability in Machine Learning. 2024 IEEE 36th International Conference on Tools with Artificial Intelligence (ICTAI), Herndon, pp. 861–867. URL: https://doi.org/10.1109/ictai62512.2024.00125.

Guaman, D., Quezada-Sarmiento, P.A., Barba-Guamán, L., Cabrera, P. & Enciso, L. 2017. SonarQube as a Tool to Identify Software Metrics and Technical Debt in the Source Code through Static Analysis. Proceedings of 2017 the 7th International Workshop on Computer Science and Engineering, Beijing, pp. 171-175. URL: https://doi.org/10.18178/wcse.2017.06.030.

Huang, D., Zhang, J.M., Harman, M., Zhang, Q., Du, M. & Ng, S.-K. 2025. Benchmarking LLMs for Unit Test Generation from Real-World Functions. URL: https://doi.org/10.48550/arXiv.2508.00408.

IEC 62304:2006. Medical device software — Software life cycle processes. International Electrotechnical Commission. Geneva, 2006.

IEC 62304:2006/Amd 1:2015. Medical device software — Software life cycle processes. International Electrotechnical Commission. Geneva.

IEC 81001-5-1:2021. Health software and health IT systems safety, effectiveness and security — Part 5-1: Security — Activities in the product life cycle. International Electrotechnical Commission. Geneva.

IEC 82304-1:2016. Health software — Part 1: General requirements for product safety. International Electrotechnical Commission. Geneva.

ISO 13485:2016. Medical devices — Quality management systems — Requirements for regulatory purposes. International Organization for Standardization. Geneva.

ISO 14971:2019. Medical devices — Application of risk management to medical devices. International Organization for Standardization. Geneva.

ISO s.a. Standards. URL: https://www.iso.org/standards.html. Accessed: 10 September 2025.

ISO/IEC 42001:2023. Information technology — Artificial intelligence — Management system. International Organization for Standardization / International Electrotechnical Commission. Geneva.

Juristo, N. & Moreno, A.M. 2013. Basics of Software Engineering Experimentation. 2001st ed. Springer Science & Business Media. New York.

Kashif, S.M., Liang, P. & Tahir, A. 2025. On Developers' Self-Declaration of AI-Generated Code: An Analysis of Practices. ACM Transactions on Software Engineering and Methodology. URL: https://doi.org/10.1145/3771937.

King, J., Klyman, K., Capstick, E., Saade, T. & Hsieh, V. 2025. User Privacy and Large Language Models: An Analysis of Frontier Developers' Privacy Policies. Proceedings of the AAAI/ACM Conference on AI, Ethics, and Society, 8, 2, pp. 1465–1477. URL: https://doi.org/10.1609/aies.v8i2.36646.

King, N. & Brooks, Joanna.M. 2017. Template Analysis for Business and Management Students. SAGE Publications Ltd. London.

Licorish, S.A., Wang, F., Bajpai, A., Arora, C. & Tantithamthavorn, K. 2024. Comparing Human and LLM Generated Code: The Jury is Still Out!. URL: https://doi.org/10.48550/arXiv.2501.16857.

Markus, A.F., Kors, J.A. & Rijnbeek, P.R. 2021. The role of explainability in creating trustworthy artificial intelligence for health care: A comprehensive survey of the terminology, design choices, and evaluation strategies. Journal of Biomedical Informatics, 113, 103655. URL: https://doi.org/10.1016/j.jbi.2020.103655.

Mashuri, S., Sarib, M., Alhabsyi, F., Syam, H. & Ruslin, R. 2022. Semi-structured interview: A methodological reflection on the development of a qualitative research instrument in educational studies. URL: https://www.researchgate.net/publication/358893176_Semi-

structured_Interview_A_Methodological_Reflection_on_the_Development_of_a_Qualitative_Research_Instrument_in_Educational_Studies. Accessed: 22 October 2025.

Niţă, N.-M., Stan, S.-E. & Ţîţu, A.M. 2024. Synergizing Quality and Legislative Software Requirements: A Holistic Approach Across the Software Product Lifecycle. International conference KNOWLEDGE-BASED ORGANIZATION, 30, 1, pp. 261–268. URL: https://doi.org/10.2478/kbo-2024-0036.

Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B. & Karri, R. 2025. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions. Communications of the ACM, 68, 2, New York, pp. 96–105. URL: https://doi.org/10.1145/3610721.

Peng, S., Kalliamvakou, E., Cihon, P. & Demirer, M. 2023. The Impact of AI on Developer Productivity: Evidence from GitHub Copilot. URL: https://doi.org/10.48550/arxiv.2302.06590.

Perry, N., Srivastava, M., Kumar, D. & Boneh, D. 2023. Do Users Write More Insecure Code with AI Assistants?. CCS '23: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security, New York, pp. 2785 - 2799. URL: https://doi.org/10.1145/3576915.3623157.

Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). Official Journal of the European Union, L119/1, 4 May 2016, pp. 1–88.

Regulation (EU) 2017/745 of the European Parliament and of the Council of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC. Official Journal of the European Union, L 117/1, 5 May 2017, pp. 1–175.

Regulation (EU) 2017/746 of the European Parliament and of the Council of 5 April 2017 on in vitro diagnostic medical devices and repealing Directive 98/79/EC and Commission Decision 2010/227/EU. Official Journal of the European Union, L117/176, 5 May 2017, pp. 176–332.

Regulation (EU) 2024/1689 of the European Parliament and of the Council of 13 June 2024 laying down harmonised rules on artificial intelligence and amending Regulations (EC) No 300/2008, (EU) No 167/2013, (EU) No 168/2013, (EU) 2018/858, (EU) 2018/1139 and (EU) 2019/2144 and Directives 2014/90/EU, (EU) 2016/797 and (EU) 2020/1828 (Artificial Intelligence Act). Official Journal of the European Union, L 2024/1689, 12 July 2024.

Saravanan, V., Kavitha, S., Ravi, S., Seetha, A., Rambabu, C. & Rajani Kanth, T.V. 2025. Generative AI in Software Engineering: Revolutionizing Code Generation and Debugging. International Journal of Computational and Experimental Science and Engineering, 11, 2. URL: https://doi.org/10.22399/ijcesen.1718.

Sjoeberg, Dag.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N.-K. & Rekdal, A.C. 2005. A survey of controlled experiments in software engineering. IEEE Transactions on Software Engineering, 31, 9, pp. 733–753. URL: https://doi.org/10.1109/tse.2005.97.

Stackoverflow 2025. 2025 Stack Overflow Developer Survey. URL: https://survey.stackoverflow.co/2025/. Accessed: 23 September 2025.

Strubell, E., Ganesh, A. & McCallum, A. 2020. Energy and Policy Considerations for Modern Deep Learning Research. Proceedings of the AAAI Conference on Artificial Intelligence, 34, 09, pp. 13693–13696. URL: https://doi.org/10.1609/aaai.v34i09.7123.

Takerngsaksiri, W., Pasuksmit, J., Thongtanunam, P., Tantithamthavorn, C., Zhang, R., Jiang, F., Li, J., Cook, E., Chen, K. & Wu, M. 2025. Human-In-The-Loop Software Development Agents. 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), Ottawa, pp. 342–352. URL: https://doi.org/10.1109/icse-seip66354.2025.00036.

U.S. Food and Drug Administration 2024. Transparency for Machine Learning-Enabled Medical Devices: Guiding Principles. URL: https://www.fda.gov/medical-devices/software-medical-device-samd/transparency-machine-learning-enabled-medical-devices-guiding-principles. Accessed: 22 October 2025.

Watson, N. & Van Italie, M. 2025. From Black Box to Open Book: An Emerging Transparency Imperative in Generative AI Codebases. 2025 IEEE Conference on Artificial Intelligence (CAI), Santa Clara, pp. 1369–1374. URL: https://doi.org/10.1109/cai64502.2025.00266.

Wong, M.-F., Guo, S., Hang, C.-N., Ho, S.-W. & Tan, C.-W. 2023. Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review. Entropy, 25, 6, pp. 888. URL: https://doi.org/10.3390/e25060888.

Xu, W., Gao, K., He, H. & Zhou, M. 2025. Licoeval: Evaluating LLMs on License Compliance in Code Generation. 2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE), Ottawa, pp. 1665–1677. URL: https://doi.org/10.1109/icse55347.2025.00052.

Yuan, Z., Chen, Y., Lou, Y., Wang, K., Cn;, Y., Lou, Liu, M., Ding, S. & Peng, X. 2024. Evaluating and Improving ChatGPT for Unit Test Generation,1 , FSE, New York, pp. 1703 - 1726. URL: https://doi.org/10.1145/3660783.

Appendices

Appendix 1. Interview Guide

Part 1: Background

- What is your role and experience in software development?
- What kind of projects have you worked on that required regulatory compliance, were any of them relevant to the healthcare sector?

Part 2: Coding and QA Practices

- Could you describe your typical coding workflow?
- What quality assurance practices are currently most important in your work? (e.g. testing, reviews, documentation)
- In your experience, what are the biggest challenges in ensuring compliance with healthcare regulations during development?

Part 3: AI-Assisted Coding

- Have you or your team used any kind of AI-assisted coding tools? (e.g. code/test/documentation/review generators)
- What potential benefits do you see behind using AI-assisted coding tools in regulated healthcare environments.
- Which potential risks do you foresee in using AI-assisted coding tools? (e.g. reliability, efficiency, explainability, compliance, security)

Part 4: Adaptations and Future Outlook

- If AI-assisted coding tools were more widely applied in your work, what adaptation would be coding, and QA practices need?
- In your opinion, what guidelines, checklists, or regulations would be most helpful for developers wiring with AI-assisted coding tools?

Closing

- Is there anything you would like to add from your experience with coding, QA, or AI-assisted coding tools?

Appendix 2. Participation Consent Forms

Participant's Consent

Name of the study: Adapting Coding and QA Practices for Using AI-Assisted Coding Tools in the European Regulated Healthcare Environments

The author of the study: Blazej Goszczynski, Haaga-Helia University of Applied Sciences,
Email: [redacted] Phone: [redacted] or [redacted]

With my signature, I confirm that I give my consent to participate in the research.

_____
Name of the Consent Giver

_____ _____
Date Signature

Appendix 3. Participant Information Sheets

Participant Information Sheet

Title of the Thesis: Adapting Coding and QA Practices for Using AI-Assisted Coding Tools in the European Regulated Healthcare Environments

Student's Name and Contact Information: Blazej Goszczynski Email: [redacted]

Commissioning Party: Haaga-Helia University of Applied Sciences.

Objective of Data Collection: The purpose of this interview is to gain understanding of current coding and QA practices in healthcare or other regulated environments, the challenges in meeting the regulations, and viewpoints and opinions on adopting AI-assisted coding in those environments.

Method and Phases of Data Collection: The semi-structured interviews will be conducted online (via Microsoft teams or similar) the interviews will be recorded with participants' permission and transcribed for analysis.

Duration of Participation: 30-45 minutes

Pre-preparation: No pre-preparations are required, only willingness to share professional experiences and perspectives.

Benefits of Participation for the Participant or their Represented Organization: Participants will contribute to research that supports safe and reliant AI-supported software development in healthcare and other regulated environments. The outcome in the form of a framework may provide practical value for organizations by offering guidelines for safe, efficient, and compliant software development.

The interviews do not ask for personal data.

Controller: Name: Heikki Hietala, Role: Supervisor Responsibility assessor, E-mail: [redacted]

Publication of Results: The thesis report will be published in the Theseus online library.

Funding and Potential Conflicts of Interest: This thesis is conducted as part of student's degree studies. There is no external funding. No conflict of interest is expected.

Additional Information: Blazej Goszczynski Email: [redacted]

Voluntary participation and withdrawal of consent: participation in the interview is voluntary. Consent can be withdrawn at any time without giving a reason, for example by stopping the interview if wished. Consent may be withdrawn at any time, without giving any reason, or by simply withdrawing consent at any time. Please note that once the results of a study have already been analyzed, it is not possible to remove the contribution of one respondent afterwards.

Appendix 4. Initial AI Prompt

Build a low-sale, non-medical “Cardio Report Generator” Python project that takes the JSON below, generates simple mock data in JSON, produces a lightweight analysis summary (descriptive, NOT diagnostic), renders an HTML report, saves it as a PDF, and includes tests plus basic project files.

JSON example (use as baseline input):

{ "patient_id": "12345", "name": "adam cook", "timestamp": "2025-10-29", "context": { "age_years": 40, "sex": "unknown", "notes": "Mock record" }, "vitals": { "CCA": { "psv_cm_s": 149.3, "edv_cm_s": 20.7, "imt_mm": 0.67 }, "ICA": { "psv_cm_s": 195.5, "edv_cm_s": 31.7, "imt_mm": 0.89 }, "ECA": { "psv_cm_s": 58.9, "edv_cm_s": 23.0 }, "ica_cca_ratio": 1.31 } }

Requirements:

- Language: Python 3.12.7, pytest 7.4.4, weasyprint 66.0, type hints, docstrings.
- Random mock generator creates plausible ranges for PSV/EDV/IMT and fills the same JSON schema.
- Analysis (non-medical): compute min/max/mean for PSV/EDV/IMT where present; list notable values (e.g., highest PSV vessel); echo ica_cca_ratio
- HTML → PDF: simple, readable template with a small table for vitals and a “Findings (descriptive only)” section. Save the .pdf.
- Tests: pytest for the whole project, analysis functions, and HTML render (e.g., template fills without errors).

Repo files to generate:

- main.py, analysis.py, data_gen.py, report_template.html
- tests/test_analysis.py, tests/end_to_end, tests/test_render.py
- README.md (how to run, examples), requirements.txt (jinja2, weasyprint), LICENSE (MIT), .gitignore

No external services; keep it simple and local.

Include sample data.json based on the provided JSON.

Deliverables:

- All source files, runnable CLI, passing tests, generated sample out/report.html and out/report.pdf.

Keep copy clear and strictly non-diagnostic. Attach each file separately