AI-Assisted Code Refactoring: Where Can it be Helpful and Where Do Humans Outperform it?

Apostolos Ampatzoglou[1], Elvira-Maria Arvanitou[2], Stavros Almpanopoulos[1], Nikolaos Mittas[3], and Alexander Chatzigeorgiou[1]

[1] Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
[2] Department of Information & Electronic Engineering, International Hellenic University, Greece
[3] Hephaestus Laboratory, School of Chemistry, Faculty of Sciences, Democritus University of Thrace, Greece

Corresponding Author: apostolos.ampatzoglou@gmail.com

Abstract. The use of Generative AI, and more specifically Large-Language Models (LLMs), is becoming an essential aid in the software development process. An increasing number of software engineers are using general-purpose or code-trained LLMs for writing code, formulating requirements or deriving test cases. However, since this practice has come so abruptly into the daily routines of developers, the research community is still lacking an in-depth evaluation of its effectiveness. A major aspect of software engineering using LLMs that is rather unexplored is the quality of the code that is generated. In this paper, we explore the ability of GenAI to assist developers in performing refactoring activities, employing well-established Object-Oriented Programming "*good-practices*" like GoF Design Patterns and SOLID principles. To achieve this goal, we have performed a controlled experiment on junior developers, relying on a cross-over experimental design, and asked them to complete development tasks with and without the use of an LLM. The results suggested that GenAI-Assisted solutions outperformed Humans-Only ones in terms of the correctness of implementing the selected practice (pattern or principle), whereas Humans-Only solutions were superior in cognitive steps of the refactoring process such as the identification of the problem and the compromised quality attributes.

1. INTRODUCTION

In recent years the terms SE4AI and AI4SE (Moreschini et al., 2025) are becoming increasingly popular, referring to research on how Software Engineering (SE) practices need to be tailored to effectively develop Artificial Intelligence (AI) driven software; and how AI can be used to boost the productivity of software development. Regarding AI4SE, which is the focus of this work, most of the published research focuses on the creation of software artifacts using LLMs (see Section 2.3). In this work, we dig further into this aspect and shed light on the effectiveness of LLMs to provide GenAI-Assisted solutions that appropriately apply "good" object-oriented practices. The general belief of both academics and practitioners is that LLMs are producing code very quickly, easily, and in general "*correctly*", especially for small-scale and easy programs (Vaithilingam et al., 2022; Lyu et al., 2025). However, there are significant concerns if the resulting code is understandable by humans, and if it can adhere to well-known programming practices (Imai, 2022; Ziegler et al., 2022), as for instance GoF Design Patterns (DP) (Gamma et al., 1995) and SOLID Principles (SP) (Martin, 2003) [***motivation-1***]. The benefits of using DPs and SPs have been exhaustively discussed in the literature and are briefly reported in Section 2.1.

In practice, there are two ways of applying DPs and SPs: forward and reverse engineering, i.e., either develop the software in the first place using the corresponding practice (during greenfield development) or refactor existing code to effectively adopt a best practice (as part of brownfield development), respectively (Charalampidou et al., 2017). In this work, we focus on refactoring an existing code to take advantage of DPs or SPs. An additional concern in AI-Assisted brownfield development is the lack of guidelines on effective prompting for guiding GenAI to apply DPs or

SPs [***motivation-2***]. In practice, a basic flow for refactoring to DP or SP, requires the execution of the following steps, as introduced by Kerievsky (2004) and Smiari et al. (2022). The refactoring steps have been inspired by the "*engineering cycle*" of the design science methodology, introduced by Wieringa (2014). The steps are summarized below:

- A deep ***understanding of the problem constraints*** (e.g., "*there can be only one ball in a basketball game*"—urging for the use of the Singleton design pattern) which need to be specified in the prompt of the LLM, to enforce its application in the GenAI solution.
- ***Understand the quality attributes that are compromised*** if the effective solution is not applied (e.g., "*a class has more than one reasons to change" —* urging to conform to the Single Responsibility Principle, the lack of which is usually reflected on low cohesion).
- ***Find whether a specific DP or SP is applicable to the identified problem***: e.g., Singleton or Single Responsibility Principle in the two previous examples.
- ***Instantiate the DP or apply the SP correctly*** so that the DP or SP are introduced without deviations from their original definition (when needed). In this work we deliberately ignore DP variants or alternatives.
- ***Effectively identify the DP or SP extension points***, so that the maintenance is performed as dictated by the DP or the SP respectively, avoiding deviations from the definitions of extension axes the DP or the SP.

To explore the effectiveness (in terms of completeness and correctness) of Humans in carrying out the refactoring steps with and without AI assistance, we have performed a controlled experiment with 10 junior developers. The reason that we focused this study on junior developers is the belief that they are more prone in "*blindly*" relying on GenAI for code generation (Haindl and Weinberger, 2024)—the effectiveness of senior developers is probably different and is considered as a future work (see Section 5.4). The goals of the experiment, linked to the motivation statements are as follows: (***g1)***: Explore the effectiveness of Human-only and GenAI-Assisted solutions when refactoring existing code to DP or SP; (***g2)***: Explore the prompt structures while interacting with GenAI, that can lead to more "*correct*" GenAI-Assisted refactoring solutions. The experimental setup (see Section 3) relies on the crossover design as adopted in the domain of SE (Vegas et al., 2016), being tailored from medical research for studying the effect of different treatments on patients. The main benefits of a crossover design with repeated measurements are: (a) the need for a smaller sample size in terms of participants; (b) a statistical analysis that mitigates as much as possible the threats to validity that emerge from differences in the participants' backgrounds; and (c) the ability to control the variability in terms of tasks' order. The results of this study have enriched the body of knowledge on the effect of AI4SE on software quality, led to interesting research directions fir future work, and provided useful implications to practitioners on how and for which tasks to use GenAI.

2. BACKGROUND INFORMATION & RELATED WORK

In this section, we outline the background and related work that frame our study. In Section 2.1 and 2.2 we provide background information on GoF Design Patterns and SOLID Design Principles, as well as the application of crossover experimentation in SE, respectively. Finally, in Section 2.3 we summarize recent related work on the use of Large Language Models (LLMs) across the software development lifecycle, whereas in Section 2.4 we present the main contributions of our study.

2.1 Background Knowledge on GoF Design Patterns and SOLID Design Principles

The application of GoF Design Patterns (DP) and SOLID design principles (SP) has long been associated with improvements in various dimensions of software quality (Wellek and Blettner, 2012; Singh and Hassan, 2015). More specifically, according to Ampatzoglou et al. (2013), 40% of the studies on the effect of GoF Design Patterns on

software quality attributes investigate the effect of GoF Design Patterns on software maintainability whereas, according to Zhang and Budgen (2012), GoF Design Patterns offer a framework for maintainability and future research efforts should be more focused on maintainability. DPs have been introduced in software engineering literature by Gamma et al. (1995). The notion of patterns in software development represents a collection of well-known design solutions to common design problems. Gamma et al. (1995) provided 23 Design Patterns across three broad categories: creational, structural, and behavioural. Their primary benefit lies in providing a share vocabulary and proven templates that enhance code understandability and communication among software engineers. Similarly, SPs offer a set of five guidelines—Single Responsibility, Open-Closed, Liskov Substitution, Interface Segregation, and Dependency Inversion—aimed at improving class design, promoting high cohesion, low coupling and clear responsibility assessment (Martin, 2003). Empirical studies (Izurieta and Bieman, 2013; Ampatzoglou et al., 2015; Ampatzoglou et al., 2019) have shown that systems incorporating well-applied DPs and SPs exhibit lower fault-proneness and higher maintainability scores compared to those without such practices. Consequently, integrating DPs and SPs into the development process is a recognized strategy for improving both the internal and external quality attributes of object-oriented software.

In our study, we focused on three well-established GoF Design Patterns and two core SOLID principles, selected for their clear and distinct impact on software quality. The selected DPs span the three categories defined by Gamma et al. (1995):

- ***Abstract Factory*** is a creational design pattern that provides an interface for creating families of related objects without specifying their concrete classes. This promotes consistency among products and enables switching between product families with minimal code changes, thereby supporting maintainability and scalability.
- ***Bridge*** is a structural design pattern that decouples an abstraction from its implementation, allowing both to vary independently. By separating interface and implementation hierarchies, Bridge recuses coupling and facilitates changes to either side without impacting the other, which enhances maintainability.
- ***Template Method*** is a behavioural design pattern that defines the skeleton of an algorithm in a base class while letting subclasses override specific steps of the algorithm without altering the overall structure. This promotes code reuse and simplifies maintenance.

In addition to these patterns, we selected two SOLID principles defined by Martin (2003):

- ***Single Responsibility Principle (SRP)*** states that each class should have only one responsibility and should have only one reason to change. Adhering to SRP improves code readability, testability, and maintainability.
- ***Open-Closed Principle (OCP)*** states that each class should be open for extension but closed for modifications. This principle encourages the design of modules that can be extended with new functionality through inheritance or composition without changing existing source code.

2.2 Background Knowledge on Crossover Designs in Software Engineering Experiments

Crossover designs are a family of experimental arrangements in which each participant is exposed to multiple treatments in different sequences. This within-subject structure reduces variability due to individual differences and can yield higher statistical power than between-subject designs (Wellek and Blettner, 2012). Crossover designs are popular in medical and psychological research. In software engineering (SE) research, crossover designs have been increasingly adopted to compare alternative techniques, tools, or processes under controlled conditions (Wohlin et al, 2010).

The benefits and potentials pitfalls of this design in the SE context are well-documented. Vegas et al. (2016) examined the use of crossover designs in SE experiments. More specifically, the authors introduced the core concepts of

crossover experimentation—where participants receive multiple treatments in different sequences—and focused particularly on the four-group AB/BA variant. The authors highlighted key benefits of this approach, notably that such designs require fewer participants and effectively control for inter-subject variability, since each participant acts as their own control. However, the authors cautioned that carryover effects—where the impact of one treatment persists and influences subsequent treatments (Cleophas, 1999)—pose significant threats to internal validity. The paper concluded with recommendations for careful planning, explicit modelling of threats and transparent reporting to ensure the reliability of findings derived from crossover designs in SE.

Frattini et al. (2024) conducted a literature review to assess the state of practice for the analysis of crossover designs in SE experiments. In particular, the study focused on identifying: (a) the prevalence of crossover designs in recent SE research; (b) the statistical methods used to analyze such experiments; (c) the extent to which threats to validity; and (d) the availability of research artifacts for reproducibility. The authors used for the search process forward snowballing of primary studies citing from Vegas et al. (2016) and targeted studies published between 2015 and March 2024. After applying selection criteria, 48 empirical crossover experiments involving human participants were identified from an initial set of 136 candidate papers. The results suggest that only 29.5% of all validity threats were appropriately addressed: maturation effects were modeled in 35.8% of studies, sequencing effects in 38.8% and carryover effects in only 3%. Finally, Frattini et al. (2024) provided a critical assessment for the research community, highlighting a methodological flaw that threatens the validity of many published findings and calling for researchers to adopt systematic and transparent analysis protocols.

Madeyski and Kitchenham (2018) presented a methodological study focused on how to calculate and interpret effect sizes for the AB/BA crossover design experiments in SE. As effect size, the authors mean the indicators that measure the magnitude of a treatment effect. In particular, the paper presented formulas in order: (a) to measure both non-standardized mean difference effect sizes and standardized mean difference effect sizes; (b) to estimate the variances of the non-standardized and standardized effect sizes; and the paper explained what descriptive statistics are necessary to enable these calculations. Madeyski and Kitchenham (2018) provided two small examples to calculate crossover study effect sizes and their variances: one using real empirical data from Scanniello et al. (2014), and another with simulated data under ideal model assumptions. The paper presented how researchers can derive effect sizes and their associated variances using R even without access to raw data. The results of this study suggest that careful calculation and transparent reporting of effect sizes in crossover designs not only improve reproducibility but also enable more accurate meta-analyses of experimental outcomes in SE research.

2.3 Related Work: GenAI and Software Development

In recent years, the application of Large Language Models (LLMs) across the Software Engineering lifecycle has increased substantially, moving from a niche topic to a mainstream tool used by developers worldwide. A recent survey by Salem et al. (2025) provides a comprehensive overview of how LLMs have been integrated into various stages of the software engineering lifecycle, including requirement gathering, system design, coding, debugging, testing, and documentation. This wealth of tools has created an urgent need for the research community to understand their impact, effectiveness and potential drawbacks. Since our study focuses on both Design Patterns and SOLID principles, this section reviews related work on how LLMs have been applied not only to pattern-oriented tasks but also to principle-driven code quality and design support. Complementing this broad perspective, Laue et al. (2024) conducted a focus group at EuroPLoP 2024 that explore how ChatGPT could be used to recommend DPs. Their findings illustrate the growing trend of employing LLMs not only for automating programming tasks but also for supporting higher-level architectural and design decision-making.

A growing body of work explores the extent to which LLMs can recognize, apply, and validate GoF Design Patterns. Kim (2025a) conducted a comparative study on the validity of pattern implementations produced during LLM-based code refactoring. More specifically, Kim (2025a) proposed an evaluation framework that uses three metrics (Property Satisfaction Rate, Critical Property Coverage, and Pattern Implementation Quality Score) to assess whether refactored code correctly adheres to the intent and structure of GoF Design Patterns (Factory Method, Strategy, Composite, Observer, and Singleton patterns). The results suggest that while LLMs can assist in pattern-oriented refactoring, their reliability varies across models. Similarly, Pandey et al. (2025) performed an empirical study to evaluate the ability of five LLMs (Code2Vec, CodeBERT, CodeGPT, CodeT5, and RoBERTa) to recognize five different GoF patterns (Abstract Factory, Builder, Factory Method, Prototype, and Singeleton) in Java. The authors used 9 projects from the P-MARt dataset[1] and generated code embeddings using the respective models. Evaluation with precision, recall and F1-score showed that RoBERTa is the top performer followed by CodeGPT and CodeBERT. Kim (2025b) proposed a method to measure LLMs ability to implement DPs using Role-Based Metamodeling Language. The method evaluates whether an LLM should apply a DP, feeding relevant code the LLM, and then checking the output for conformance to the solution specification and completeness of transformation steps. For the evaluation purposes, Kim (2025b) used the Visitor pattern applied to three case studies in ChatGPT 4. The results of this study demonstrate high performance (avg. 98% conformance and 87% completeness), showing the model's potential for design-pattern aware code generation. Finally, Pan et al. (2025) conducted an empirical study evaluating code LLMs in their capacity to recognize, comprehend and generate software design patterns. Pan et al. (2025) designed three experiments using Java and Python repositories annotated with twelve GoF patterns. The results suggest that existing code LLMs exhibit systematic biases, often failing to adhere to design patterns or only doing so superficially.

A complementary line of research explores the broader role of LLMs in software refactoring, code quality improvement, and design support. Liu et al. (2025a) performed an empirical study to assess the capabilities of LLMs (i.e., ChatGPT and Gemini) in automating software refactoring tasks. They constructed a dataset comprising 180 real-world refactorings from 20 Java projects. The results indicated that without specific prompts, ChatGPT and Gemini identified only 28 and 7 refactoring opportunities respectively, out of 180. In parallel, White et al. (2024) proposed a catalog of prompt patterns designed to enhance the effectiveness of LLMs (like ChatGPT) in automating various software engineering activities. These prompt patterns address common challenges such as ensuring code modularity, decoupling from third-party libraries, etc. The study suggests that by employing these prompt patterns, software engineers can leverage LLMs more effectively, leading to improved code quality, streamlined development process and better alignment with software design principles.

Recent research indicates that the intersection of LLMs and software design patterns is expanding beyond the classical GoF patterns into several new areas. For example, Andrade et al. (2025) proposed a novel framework that leverages LLMs to detect and mitigate security risks associated with software design patterns and antipatterns. The proposed framework analyzes relationships and behavioral dynamics in code, identifying issues such as unauthorized state changes, insecure communication and inappropriate data handling. Similarly, Liu et al. (2025b) performed a systematic literature review to provide a comprehensive catalogue of 18 architectural patterns designed to guide the development of foundation model-based agents. The patterns address key challenges such as goal-seeking, plan generations explainability and accountability. This catalogue aims to assist software engineers in designing effective agent architectures by facilitating informed decision-making and promoting best practices in the field. Additionally, Postle and Salingaros (2025) proposed a hybrid design framework that integrates Christopher Alexander's pattern language with generative AI models. To address the limitations of Alexander's static text and the ungrounded nature

[1] https://www.ptidej.net/tools/designpatterns/

of standalone LLMs, the framework leverages the original 253 patterns as a foundational knowledge base. Upon receiving a specific design prompt, the tool identifies relevant patterns and employs an LLM to synthesize, adapt, and generate a new, context-specific pattern language.

In addition to design patterns, researchers have also begun investigating how LLMs can support adherence to fundamental design principles such as SOLID. Martins et al. (2024) investigated the use of LLMs in automating code reviews, specifically focusing on their ability to detect violations of the five fundamental SOLID principles of object-oriented design. The study reveals a notable performance gap: while the LLMs were adept at identifying violations of more straightforward, syntax-oriented principles (like the Single Responsibility Principle), and their reliability diminished when faced with abstract principles requiring deeper semantic understanding (such as the Liskov Substitution Principle).

2.4 Main Contributions

While the related work reviewed in Section 2.3 demonstrates growing interest in applying LLMs to various software engineering tasks, this study addresses specific gaps that have not been adequately covered by existing literature. We can classify the differentiation of our work from existing literature, as shown below. Regarding the ***Scope of Interaction***, most related works focus on the capability of LLMs to either autonomously recognize design patterns (Pandey et al., 2025; Pan et al., 2025) or implementation quality (Kim, 2025a; Kim, 2025b). Our work focuses *on refactoring existing code*, investigating how GenAI assists software engineers in transforming code smells into high-quality object-oriented solutions that adhere to best practices. Furthermore, we explicitly compare Human-only versus GenAI-Assisted workflows, *placing the software engineer in the driver's seat*, instead of other studies that treat LLMs as autonomous agents (e.g. Liu et al., 2025a). With respect to the ***Granularity of the Evaluation***, existing studies primarily assess the syntactic correctness or the functional validity of the final code produced by LLMs (Kim, 2025a; Pan et al., 2025). Our study systematically evaluates *the entire refactoring process through five distinct cognitive and technical steps*. This allows us to identify *where exactly humans outperform AI* and *where AI excels*, rather than providing a simple "*better* or *worse*" comparisons. Additionally, the ***Scope of Applied Practices*** existing studies have explored either GoF patterns (Kim, 2025a; Kim, 2025; Pandey et al., 2025; Pan et al., 2025) or SOLID principles in isolation (Martins et al., 2024). Our study investigates *both within the same experimental framework*. This allows for a comparative analysis of how GenAI handles concrete implementation templates versus abstract design guidelines, revealing distinct performance gaps between structural tasks and conceptual compliance. Finally, the ***Experimental Methodology***, we employ *an AB/BA crossover experimental design with repeated measures* (Vegas et al., 2016), combined with LME modeling for statistical analysis. By ensuring that each participant acts as their own control, this approach effectively mitigates threats to validity regarding developer skills and reduces the noise inherent in subject variability.

3. EXPERIMENTAL SETUP

The experiment has been designed based on the guidelines of Vegas et al. (2016) for performing crossover experiments in the field of software engineering. Based on the literature, a systematic methodology is essential for ensuring the validity and replicability of crossover experiments. Vegas et al. (2016), proposed a set of good practices for software engineers to perform a crossover experiment in SE. The methodology begins at the design stage by establishing the experiment's core architecture. Software engineers must first ***define the periods***, which involves deciding how many times each participant (subject) will perform the experimental task under different treatments and study the implications. For a standard AB/BA design, this means two periods. Next, they must ***define the sequences*** by specifying the order in which treatments are applied. A critical design-time activity is to proactively ***deal with car-***

ryover at design time. The software engineers must select a strategy to minimize the chance that the first treatment experience influences performance in the second treatment. Once the data is collected, the analysis must correctly leverage the design's strengths. The chosen data analysis technique must ***consider subject variability***, because each subject provides repeated measures, and statistical methods that handle dependent data such as a paired t-test (repeated measures on the same subject). Ensuring within-subject variability, the statistical power will be increased. Next, the analysis must begin by verifying the design's assumptions. Software engineers must ***deal with carryover at analysis time***, by performing a statistical test to check for its presence. Regarding the ***match analysis with design***, the variables included in the analysis must be consistent with design decisions (e.g., treatment, period, sequence, carryover, etc.). Finally, reporting must be comprehensive. The guideline to ***beware of effect size*** proposed that depending on the design and/or results determine whether effect size can be meaningfully calculated. Reporting it provides a complete picture of the treatment's impact and is essential for future meta-analyses. Inspiration for the reporting has been borrowed from the seminal guidelines for conducting experiments introduced by Wholin et al. (2010).

3.1 Research Goal and Research Questions

The goal of this study is to explore the effectiveness (in terms of completeness and correctness) of refactoring to DPs and SPs, when developers are assisted by LLMs compared to humans that rely only on their expertise. To address this goal, we have set two main research questions:

[**RQ$_1$**] Is there a difference in the correctness of GenAI-Assisted and Human-only solutions when refactoring code to apply DPs or SPs?

[**RQ$_{1.1}$**] Are there any differences in the correctness of GenAI-Assisted and Human-only solutions in specific steps (problem identification, compromised quality attribute selection, treatment selection, treatment implementation, identification of extension mechanism) of the refactoring process?

[**RQ$_{1.2}$**] Is there a difference in the in the correctness of GenAI-Assisted and Human-only solutions, when refactoring code to apply DPs compared to when they apply SPs?

RQ$_1$ (and its sub-questions) are related to ***g1***; thus, by answering this question we aim to extend the body of knowledge on code refactoring with GenAI support. Specifically, we want to explore which parts of the refactoring process can be assisted by GenAI, and which parts Humans must handle without GenAI support. We also plan to explore if the type of the refactoring is a factor that influences the decision of whether or not to use GenAI support.
HYPOTHESIS: Different approaches (Human-only and Gen-AI Assisted) are likely to be better suited for different steps of the refactoring process.

[**RQ$_2$**] What is the relation between the prompts used while interacting with Gen AI and the quality of the provided solutions?

RQ$_2$ is related to ***g2***; thus, by answering this question we aim to identify characteristics of the prompts, while interacting with GenAI for refactoring support, that can lead to more correct results. Such knowledge can lead to the proposition of some high-level prompt input guidelines, which can act as input for future studies that will qualitatively and more accurately study this phenomenon.
HYPOTHESIS: Prompt inputs are related to the correctness of the GenAI-Assisted solutions.

3.2 Study Setup

The experiment was conducted with 10 junior / entry-level developers (6 months to 3 years of working experience). For participant selection we have launched a call to collaborating companies in Thessaloniki. Interested entry-level developers have reacted to the call volunteering to participate. Out of these applications we have selected 10 that have followed related courses, as follows: software engineering (at least 1 semester), quality management (at least 1 semester), and Java development (at least 2 semesters). Out of the applicants, we have selected those with the highest grades in the quality management course, supposing a better background knowledge on DP or SP. However, the personal knowledge and self-studying of participants could not have been controlled. In that sense, it is considered as a random effect. Some basic demographics of the group of participants are presented in Figure 1.

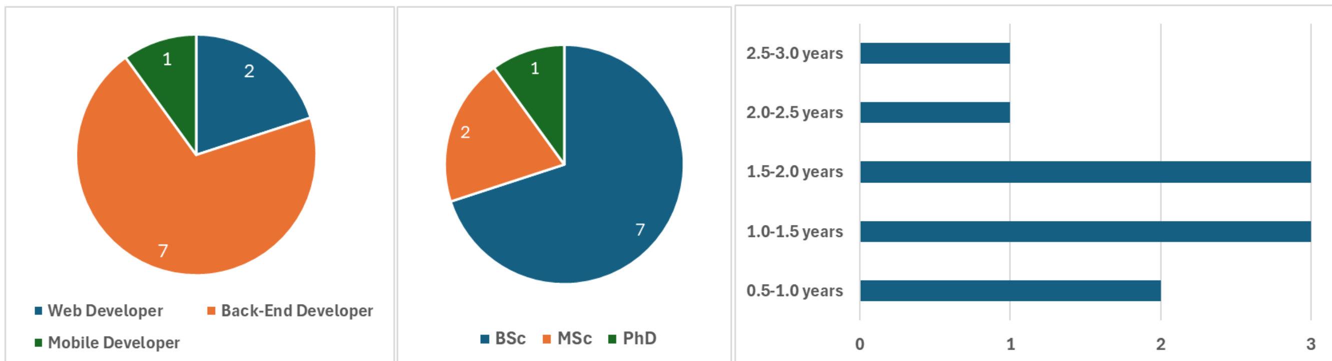

Figure 1: Demographics of Participants

The participants were split into two groups of balanced working experience in programming. The two groups (5 participants each), in each session, were assigned a task of the following form. We have selected simplified scenarios for tasks, so that their solution can be framed in the 1-hour slot that was given to the participants (per task). The tasks were designed based on the experience of the researchers as instructors of DPs or SPs, to be feasible to solve in the timeframe. The scenarios were functional variations of textbook examples from DP or SP literature.

> We wish to model a system of a clothing shop. The shop (for now) sells only pants and shirts. There are two types of clothing, Vintage and Modern. Each clothing contains a void `printInfo` method which takes no parameters and prints a message of the type: `"A <Type> <Clothes>"` (e.g. "A Modern Pants"). *It is worth noting that each clothing shop can sell only one of the clothing types, that means that a Modern Shop cannot sell Vintage clothing and vice versa*.

The constraint of initializing only a specific family of products suggests that the Abstract Factory DP must be used. The two groups were given two different instructions, based on the treatment that we wanted them to use for exploring the design space. For the Human group, the instructions were as follows:

> Study the requirements and the attached initial code and answer the following questions. For answering the questions, please consult the lecture notes on metrics, design principles, and design patterns (attached).

The GenAI group was using the free version of ChatGPT that was available at the time of the experiment, i.e., GPT-4.1-mini. We note that with the term GenAI group, we refer to a group of people that are getting assistance from ChatGPT to provide an answer, since pure AI solutions are not available (see Section 6 for more details on the relevant threat to validity). For the GenAI group the instructions were as follows:

Study the requirements and the attached initial code and answer the following questions. While using GenAI, please record the complete prompting (AI-human conversation) record. You are free to provide to GenAI any kind of prompt that you feel is relevant.

The starting solution for the specific task was the following:

```
public abstract class Clothes{
	public abstract void printInfo();
}
public abstract class Shirt extends Clothes{
	public abstract void printInfo();
}
public abstract class Pants extends Clothes{
	public abstract void printInfo();
}
public class VintageShirt extends Shirt{
	public void printInfo(){
		System.out.println("A vintage shirt");
	}
}
public class ModernShirt extends Shirt{
	public void printInfo(){
		System.out.println("A modern shirt");
	}
}
public class VintagePants extends Pants{
	public void printInfo(){
		System.out.println("A vintage pair of pants");
	}
}
public class ModernPants extends Pants{
	public void printInfo(){
		System.out.println("A modern pair of pants");
	}
}
public class Shop{
	private List<Clothes> clothing;
	public Shop(){
		this.clothing = new ArrayList<Clothes>();
	}
	public void addShirt(Clothes c){
		clothing.add(c);
	}
	public void addPants(Clothes c){
		clothing.add(c);
	}
}
public class MainClass{
	public static void main(String[] args){
		Shop vintageShop = new Shop(); // Create a Vintage shop
		vintageShop.addShirt(new VintageShirt());
		vintageShop.addPants(new VintagePants());
		Shop modernShop = new Shop(); // Create a Modern shop
		modernShop.addShirt(new ModernShirt());
		modernShop.addPants(new ModernPants());
	}
}
```

The questions to be answered by both groups were the following, answered as a Google Form:

- Describe the bad smells that you identify / What problems is the code suffering from?
- Describe the metrics / quality attributes that are compromised
- What DP or SP would you use to treat this problem?
- Please upload the code that implements the treatment
- Note at least one extension scenario that shows that the treatment provides a more extensible solution, compared to the starting code.

The experimental study material has been piloted with junior researchers from the research group of the authors. These researchers were not part of the final experiment to avoid bias. The piloting has led to modifications and clarifications. During the experiment execution, there were no complaints about the task descriptions or the prompts for data collection.

The crossover experiment was held for one week (early June 2025). The experiment was conducted over five consecutive days in the afternoon, so that participants were not at work. Each day, two sessions took place, each one lasting for 1 hour (per task). Participants could leave earlier if they had finished their tasks. The allocation of tasks to sessions, days, and participants are depicted in Figure 2. The columns in the figure correspond to time (sessions, within days), and the rows are participants. The cells present dual information: the color corresponds to whether the participant was allowed to use GenAI or not, whereas the text to the task that they were assigned to (through a problem-id). The first part of the problem-id shows if it corresponds to a DP or a SP, the second part to a specific practice (Abstract Factory—AF, Bridge—BR, Template Method—TM, Open-Closed Principle—OCP, or Single Responsibility Principle—SRP). Finally, the index 1 or 2 at the end of the problem-id indicates the version of a task solved through the same practice. For instance, regarding AF, DP_AF1 was referring to the clothing task, whereas a DP_AF2 to a task with creating factories of game characters. We preferred to use different versions of the problems in the two sessions, so that we avoid having an extra knowledge boost in the second session of the day. For shuffling the task to days, sessions and participants, we have obeyed to an AB/BA sequence among groups (i.e., one group started with GenAI and the other without in the 1st session, and vice versa in the 2nd session). In each day, each participant encountered only one problem type (DP or SP) and specific practice (e.g., AF).

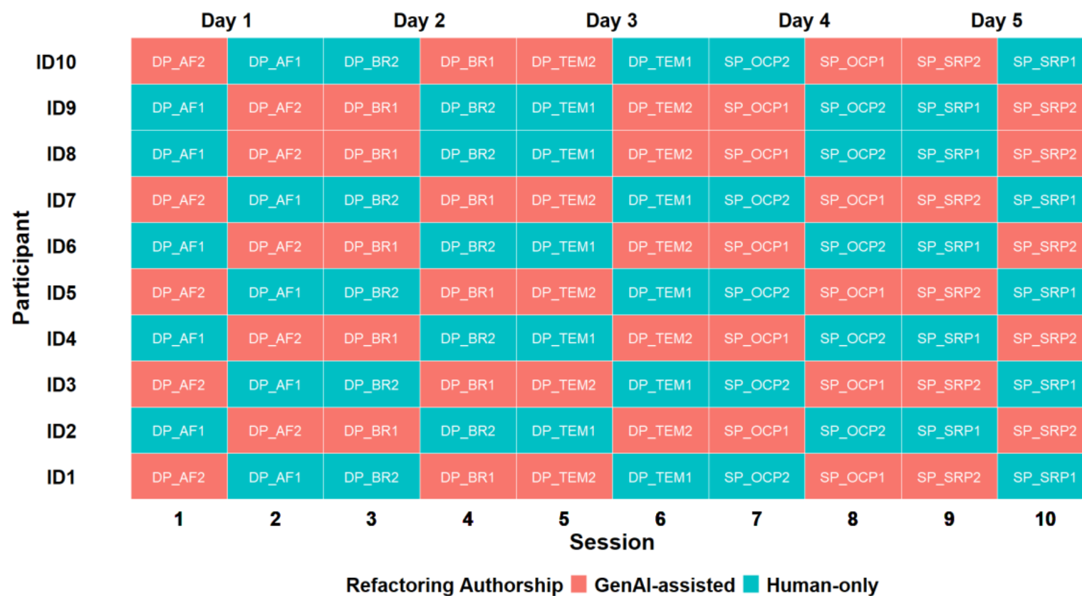

Figure 2: Allocation of tasks across participants, sessions, and study days

3.3 Data Collection

Following the study setup, described in Section 3.2, to answer RQ_1, we have concluded with a dataset of 100 rows (one row for each junior developer, solving each task), and 8 columns:

- ***Participant-ID***: An id for each participant to be considered as a random effect, since we believe that each developer has different skills on DPs and SPs.
- ***Refactoring Authorship***: GenAI-assisted or Human-only solution
- ***Refactoring Type***: Design Pattern (DP), SOLID Principle (SP)
- ***Refactoring Practice***: DP_AF, DP_BR, DP_TM, SP_SRP, and SP_OCP
- ***Refactoring-ID***: DP_AF1, DP_AF2, DP_BR1, DP_BR2, DP_TM1, DP_TM2, SP_SRP1, SP_SRP2, SP_OCP1, SP_OCP2
- ***Correctness Problem Identification***: A score from 1-10 being the grade of the first and the second author of this work, on the correctness of the problem identification (the grade is an average). The scoring guide for this variable was as follows: 10=the answer is targeted, and uses examples from the code to describe the symptoms; 8=the answer includes the correct bad smell, but it includes also other less important (or incorrect) problems, it uses examples from the code to describe the symptoms; 5=the answer includes the correct bad smell, but it includes also other less important (or incorrect) problems, it does not uses examples from the code to describe the symptoms; 3=the answer includes relevant bad smells, but not the correct one; 0=the answer is irrelevant. In case of a difference of more than 20%, the fifth author of this work has functioned as a mediator. This process has been initiated in 2 cases.
- ***Correctness Compromised QAs***: A score from 1-10 being the grade of the first and the second author of this work, on the correctness of selecting the compromised QAs (the grade is an average). The scoring guide for this variable was as follows: 10=the answer is targeted, and the selected metrics / QAs are related to the symptoms; 6=the answer includes the correct metrics / QAs, but it includes also other incorrect metrics / QAs; 3=the answer includes relevant metrics / QAs, but not the correct one; 0=the answer is irrelevant. In case of a difference of more than 20%, the fifth author of this work has functioned as a mediator. This process has been initiated in 3 cases.
- ***Correctness Selected Practice***: A score from 1-10 being the grade of the first and the second author of this work, on the correctness of the selected treatment (the grade is an average). The scoring guide for this variable was as follows: 10=the answer is targeted, and uses examples from the code to describe the solution; 8=the answer is targeted, but it is suboptimal (e.g., Strategy instead of Template Method, Strategy instead of OCP, etc.) and uses examples from the code to describe the solution; 5= the answer includes the solution, but it includes also other suboptimal (or incorrect) solutions, it uses examples from the code to describe the solution; 3=the answer includes relevant solutions, but not the correct one; 0=the answer is irrelevant. In case of a difference of more than 20%, the fifth author of this work has functioned as a mediator. This process has been initiated in 1 case.
- ***Correctness Practice Implementation***: A score from 1-10 being the grade of the first and the second author of this work, on the correctness of the implementation of the treatment (the grade is an average). The scoring guide for this variable was as follows: 10=the code is functional, the treatment is well implemented; 6=the code is not functional, but the treatment is well implemented; 3= the code is not functional, the treatment is not correctly implemented, but there is a general grasp of the implementation details (e.g., abstraction, polymorphism, etc.); 0=the answer is incorrect. In case of a difference of more than 20%, the fifth author of this work has functioned as a mediator. This process has been initiated in 8 cases.
- ***Correctness Extension Mechanism Identification***: A score from 1-10 being the grade of the first and the second author of this work, on the correctness of identifying the correct extension mechanisms (the grade is an average).

The scoring guide for this variable was as follows: 10=the answer is targeted to both the problem context and solution, and uses examples from the code to describe the extension; 8=the answer is targeted to both the problem context and solution, but does not use examples from the code to describe the extension; 5=the answer is targeted to solution, but not the problem context, does not use examples from the code to describe the extension; 3=the answer is targeted to problem context, but not the solution, does not use examples from the code to describe the extension; 0=the answer is irrelevant. In case of a difference of more than 20%, the fifth author of this work has functioned as a mediator. This process has been initiated in 3 cases.

To answer RQ_2 the dataset comprised of 5 sets of 6 columns, each set corresponding to the correctness score of each refactoring set. The dataset included 50 rows, filtering from the dataset of RQ_1 only GenAI-Assisted refactoring authorships. The six columns represent on the one hand the correctness of the refactoring step, and on the other hand 5 prompt input metrics, representing the interaction of the subject with the LLM, in each step of the process. In particular, we selected a set of quantitative and qualitative metrics based on recent empirical frameworks for AI-assisted programming (Ross et al., 2023; Zamfirescu-Pereira et al., 2023; Liu et al., 2023). Specifically, we aim to capture three distinct dimensions of the prompting process: (a) *Conversational Effort*—following Ross et al. (2023), we measure Initial Prompt Length, Total Prompt Length, and Number of Turns; (b) *Prompt Quality and Specificity*—adapting Zamfirescu-Pereira et al. (2023), we employee Instruction Detail to assess the semantic depth of user inputs; and (c) *Technical Constraints*—refereeing Liu et al. (2024), we track Context Window Utilization to monitor the cumulative token load. The columns of the dataset are presented below:

- ***Refactoring Step Correctness***: The score of each step as described above
- ***Initial Prompt Length***: Measured as the token count of the very first prompt submitted by the participant for each refactoring task. This metric serves as a proxy for the developer's ability to initially frame the problem.
- ***Total Prompt Length***: Measured as the cumulative token count of all user-submitted prompts within a single conversational session for a given task. This metric reflects the total effort and information provided by the developer throughout the interaction.
- ***Number of Turns / Interactions***: Measured as the extent of the dialogue between the developer and the LLM, we counted the Number of Turns for each refactoring task. This metric allowed us to quantify the conversational effort required to reach a solution and to differentiate between efficient one-shot interactions and more complex, iterative dialogues.
- ***Context Window Utilization***: Measured as the cumulative usage of the model's context window throughout a conversation session. The rationale is that in multi-turn interactions, the context available to the LLM expands with every turn, encompassing the entire conversation history (all prior prompts and responses). It is calculated as the percentage of the model's capacity consumed at each step, by dividing *Input Tokens Used* (the sum of the tokens in the current prompt plus the tokens from the entire conversation history up to that point) to the *Max Context Capacity* (to the official context window limit of the model used—in this study (GPT-4o mini), which is 128k tokens).
- ***Instruction Detail:*** Evaluates the qualitative nature of the developer's guidance rather than its mere length. It assesses the technical specificity and information richness of the prompt. To measure this, we established a 3-level ordinal scale to classify each user prompt: (1) *Vague/Goal-Oriented*, for general requests without technical details (e.g., "improve this code"); (2) *Specific/Descriptive*, for prompts that identify a concrete problem (e.g., "this method is too long"); and (3) *Prescriptive/Technical*, for prompts that specify a software engineering technique or solution (e.g., "apply the Strategy Pattern"). To ensure reliability, each

prompt in our dataset was independently rated on this scale by two researchers. The final metric value was calculated as the average of the two scores. The inter-rate agreement was high, with a disagreement rate of approximately 3%, indicating strong consistency in the evaluation of the prompts.

The prompts that have been recorded can be found in the dataset in two languages; either English or Greek. Although this decision imposes a confounding factor, we have allowed participants to use their mother tongue as well, due to: (a) the perception that people are more likely to participate in an experiment without any language constraints (Maha et al., 2020); (b) the likelihood that improper prompting might be due to no fluency of some participants in English language, which would also be a threat; and (c) the ability of modern LLMs to work multilingually (Qin et al, 2025).

3.4 Data Analysis

Answering RQ₁: Continuous variables for correctness score variables were summarized using mean (standard deviation-SD) and median (range). Given the within-subject repeated-measures design with daily AB/BA counterbalancing, where each participant completed similar paired versions of the same refactoring practice for both refactoring types (DP or SP) with (GenAI-assisted) and without (Human-only) Gen-AI assistance across five consecutive days (Day 1 to Day 5) (see Figure 2), we employed Linear Mixed Effects (LMEs) models to address the posed RQs (Bates et al. 2015). To investigate the fixed effects of the two experimental factors of interest (`Refactoring Authorship` and `Refactoring Type`), separate LME models were fitted for each correctness score variable (Table 1). Each model included both `Refactoring Authorship` (GenAI-assisted vs. Human-only) and `Refactoring Type` (DP vs. SP) as fixed main effects, along with their interaction term (`Refactoring Authorship × Refactoring Type`), to examine whether the effect of refactoring authorship on correctness scores varied between refactoring types. The model with the interaction term was compared to the model containing only the main effects (`Refactoring Authorship` and `Refactoring Type`) using a Likelihood Ratio Test (LRT). If the interaction term was not statistically significant, the simplified main-effects model was chosen as the final model.

A random intercept for each participant (`Participant-ID`) was incorporated to account for intra-subject correlation due to repeated measurements and to systematically control for variability due to individual differences. Additionally, a random intercept for each refactoring practice (`Refactoring Practice`) was included to account for variability in difficulty or structural complexity across the refactoring practices that could affect correctness independently of the refactoring authorship, refactoring type or participant. To further assess potential learning/fatigue effects across the experiment (10 Sessions) and sequence effects (i.e., whether participant began each day with or without GenAI assistance), the final model was compared with extended version of the final model that included `Session Order` (1 to 10) and `Sequence` (GenAI-first/Human-first) as additional covariates (fixed effects) via LRT. The extended model was retained for further examination when these covariates significantly improved model fit accounting for learning/fatigue and sequence effects. Post-hoc comparisons of estimated marginal means were conducted on the fitted models using the Tukey correction to appropriate adjust the p-values and to construct 95% Confidence Intervals (CI) error plots. All statistical analyses were performed using R (R Core Team, 2025) through *lme4*, *lmerTest* and *emmeans* packages. All statistical tests were non-directional, and a difference was considered significant if the p-value was less than 0.05 ($a = 0.05$).

Table 1. Description of Model Components used in the analysis

Model Component	**Description**	**Goal**
Correctness Score	Dependent variable	
Refactoring Authorship	Main fixed effect term indicating who	Tests the overall main effect of refac-

Model Component	Description	Goal
	produced the code (GenAI-assisted vs. Human-only) for solving a task	toring authorship on correctness ignoring the effect of refactoring type
Refactoring Type	Main fixed effect term representing the refactoring type (DP vs. SP) of task being solved	Tests the overall main effect of refactoring type on correctness ignoring the effect of refactoring authorship
Refactoring Authorship × Refactoring Type	Interaction term between refactoring authorship and refactoring type	Tests whether the effect of refactoring authorship on correctness scores depends on refactoring type
Participant-ID	Random effect for participants	Models the variability due to differences between participants
Refactoring Practice	Random effect for refactoring practices of tasks being solved	Models the variability due to differences across refactoring practices

Answering RQ₂: To answer RQ_2, we computed the correlation between the 5 prompt input metric scores and the correctness of each refactoring step. Considering the setting up of the study it is necessary to opt for a correlation calculation procedure suitable for repeated measurement. Thus, we have calculated the repeated measures correlation coefficient, as presented by Bland and Altman (1995). Similar to any other kind of correlation analysis we will report the correlation coefficient and the p-value.

4. RESULTS

In this section we present the results of our experiment and answer the two research questions that have been set during our experimental design. We note that in this section we present the results and explain them in detail. However, their interpretation, comparison against state-of-the-art, and implications for research and practice, as well as future research opportunities are discussed in Section 5. Additionally, we note that we have checked that all Gen-AI assisted solutions have utilized the input that they got from the models; and that no participants ignored the Gen-AI suggestions.

4.1 Differences in the Correctness of GenAI-Assisted and Human-only Refactoring Solutions

Table 2 summarizes the descriptive statistics for correctness scores across the five steps of the refactoring process for each combination of the `Refactoring Authorship` and `Refactoring Type` factors, whereas Figure 3 presents the score distributions to provide further insights. The central tendency measures along with the graphical inspection of the distributions suggest that performance between GenAI-assisted developers and developers relying solely on their own expertise differs across the five refactoring steps, with some differences depending on the refactoring type. Specifically, during Problem Identification, non-assisted developers generally achieved higher scores compared to GenAI-assisted developers regardless of the refactoring type, indicating that developers tend to grasp the essence of the problem more effectively. Given the score of almost 6.5 against 5.5 for Human-only and GenAI-Assisted solutions, we can observe that the average solution of both `Refactoring Authorship` strategies tends to also identify both relevant and irrelevant solutions; however, the Human-only answers are better justifying the choice by linking the answer to the given source code. In contrast, GenAI-assisted solutions outperformed Human-only solutions in Practice Implementation for both refactoring types implying that the LLM is more capable of generating code that satisfies specified requirements. By examining the average scores of the solutions (Human-only: ~6.5 and GenAI-Assisted: ~8.5), we can conclude that the main difference of the solutions stands in the working functionality of the solution, since both capture the context and the feeling of the `Refactoring Practice`. With

respect to Compromised QAs and Extension Mechanism Identification, differences in correctness scores varied by the refactoring type being solved, since non-assisted developers exhibited higher performance in SP tasks, but lower performance to GenAI-assisted developers in DP tasks. Finally, both GenAI-assisted and non-assisted developers demonstrated similar performance in selecting the right practice (Selected Practice step) for DP tasks, whereas slightly differences were noted for SP tasks.

Table 2. Descriptive statistics for Correctness Scores across the five steps of the refactoring process

Correctness Score	**Refactoring Type**	**Refactoring Authorship**	***N***	***M***	***SD***	***Mdn***	***min***	***max***
Problem Identification	DP	GenAI-Assisted	30	5.23	2.74	5.00	0	10
		Human-only		6.81	2.72	7.00	0	10
	SP	GenAI-Assisted	20	5.95	2.58	6.00	3	10
		Human-only		6.33	3.08	6.66	0	10
Compromised QAs	DP	GenAI-Assisted	30	5.03	2.62	5.00	0	10
		Human-only		6.85	3.04	8.00	0	10
	SP	GenAI-Assisted	20	6.10	2.55	6.50	3	10
		Human-only		5.72	2.93	5.72	0	10
Selected Practice	DP	GenAI-Assisted	30	5.90	3.22	7.00	0	10
		Human-only		6.41	3.08	7.00	0	10
	SP	GenAI-Assisted	20	5.20	3.79	6.00	0	10
		Human-only		4.11	3.67	3.56	0	10
Practice Implementation	DP	GenAI-Assisted	30	8.00	3.06	10.00	0	10
		Human-only		6.41	3.85	7.50	0	10
	SP	GenAI-Assisted	20	8.50	2.52	10.00	2	10
		Human-only		6.17	2.89	6.08	2	10
Extension Mechanism Identification	DP	GenAI-Assisted	30	6.73	3.92	8.50	0	10
		Human-only		7.78	3.33	10.00	0	10
	SP	GenAI-Assisted	20	7.55	3.20	9.50	2	10
		Human-only		5.78	3.36	5.39	2	10

The columns *N, M, SD, Mdn, min, max* denote, respectively, the total number of DP or SP tasks, the mean, the median, the minimum and the maximum of correctness score variables

Although graphical inspection of correctness scores provides a straightforward overview of the problem, it cannot be used alone to draw generalizable conclusions on the phenomenon under examination. Therefore, as described in Section 3.4, five separate LME models were fitted to systematically investigate the factors affecting correctness scores, while controlling for additional variability arising from differences among participants and refactoring practices. To facilitate readability, the following paragraphs present only the key results: (a) LRTs examining whether the effect of `Refactoring Authorship` on correctness scores depends on `Refactoring Type` , (b) Analysis of Variance (ANOVA) using Satterthwaite's method to evaluate fixed effects (main and interaction terms) and (c) LRTs examining whether `Session Order` and `Sequence` should be included as covariates in the final models. The detailed results from the fitting of the LME models including fixed effects coefficients and random intercepts can be found in Appendix.

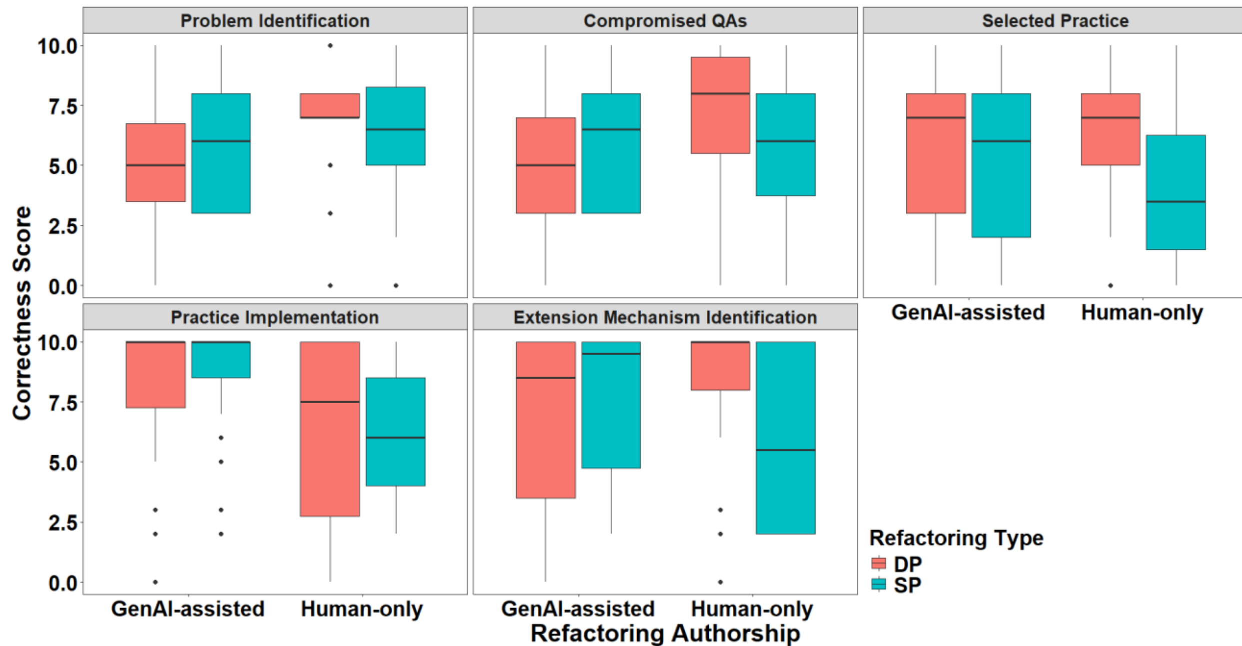

Figure 3: Correctness Score distributions across the five steps of the refactoring process

For ***Problem Identification***, the comparison between the full model with the interaction term of `Refactoring Authorship` × `Refactoring Type` and the model with only the main effects did not reveal a statistically significant interaction ($\chi^2(1) = 1.390, p = 0.238$). Therefore, the more parsimonious main effects model was retained for further analysis. Similarly, the comparison with an extended model including the covariates of `Session Order` and `Sequence` did not reveal a statistically significant difference ($\chi^2(2) = 2.318, p = 0.314$). The ANOVA results showed a statistically significant main effect of `Refactoring Authorship` on correctness scores ($F(1,95) = 4.453, p = 0.037$), but no significant effect of `Refactoring Type` ($F(1,5) = 0.010, p = 0.925$). Specifically, the intercept for `Refactoring Authorship` concerning the Human-only level ($b = 1.100, se = 0.524, p = 0.039$ see Appendix) indicates that correctness scores increase by 1.100 points, when the solution stems from human expertise rather than from GenAI-assisted (baseline category) code. Indeed, the estimated marginal means (see Figure 4a), visually, confirm the above findings, since the lines are parallel indicating no interaction effect. Interestingly, there is a difference between the AI-assisted and Human-only levels for both DP or SP, with higher for Human-only suggesting a main effect of `Refactoring Authorship`, whereas the lines are very close to each other across both DP or SP levels indicating a negligible difference between the two refactoring types.

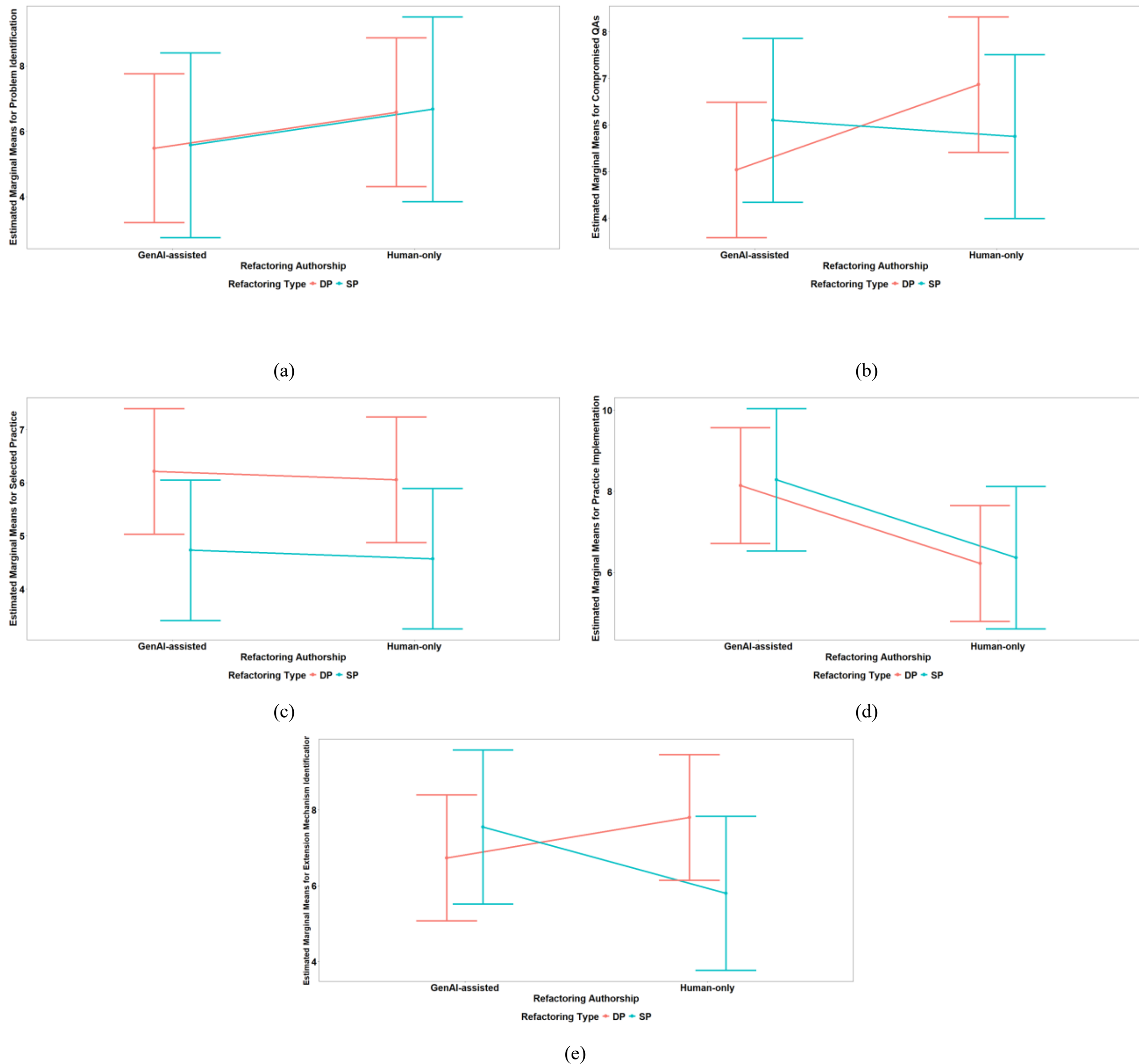

Figure 4: Estimated Marginal Means for Correctness Score

The results derived from the fitting of the LME model for ***Compromised QAs*** bring to surface remarkable findings that deserve further investigation. In this regard, LRT revealed a statistically significant interaction between `Refactoring Authorship` and `Refactoring Type` ($\chi^2(1) = 3.926, p = 0.048$). This interaction is also evident in Figure 4b, in which the lines intersect. Post-hoc analysis on estimated marginal means using the Tukey's correction showed a statistically significant difference only for the paired comparison between GenAI-assisted and Human-only code in DP tasks. Furthermore, the estimate of means suggest that participants performed better when solving DP tasks without using assistance from AI (AI-assisted), with a mean increase in correctness scores of approximately 1.833 points ($p = 0.048$). Again, the inclusion of covariates (`Session Order` and `Sequence`) did not result to a significant improvement of the final model ($\chi^2(2) = 4.171, p = 0.124$).

Concerning ***Selected Practice***, the interaction term (`Refactoring Authorship` × `Refactoring Type`) was not statistically significant $\chi^2(1) = 1.363, p = 0.243$, whereas, the final model revealed, in this case, a statistically significant main effect of `Refactoring Type` on correctness scores ($F(1,88) = 4.741, p = 0.032$) but no

significant main effect of `Refactoring Authorship` ($F(1{,}88) = 0.058, p = 0.811$). This is also depicted in Figure 4c, in which the lines are nearly horizontal (no `Refactoring Authorship` effect) yet, the estimated marginal means are consistently higher for DP compared to SP tasks. The estimated coefficient for the fixed effect of `Refactoring Type` on SP tasks ($b = -1.483, se = 0.681, p = 0.032$ see Appendix) suggests that participants performed significantly worse on SP tasks than on DP tasks, with correctness scores decreasing by approximately 1.5 points, regardless of whether they used GenAI-assisted code or Human-only approaches. LRT showed no evidence of learning/fatigue or sequence effects ($\chi^2(2) = 0.448, p = 0.799$).

For ***Practice Implementation***, the interaction term was dropped out from the final model ($\chi^2(1) = 0.313, p = 0.575$) and the reduced model containing only the main effects was further analyzed. Similarly, to Problem Identification, the lines are very close to each other (Figure 4d) indicating no significant main effect of `Refactoring Type` on correctness scores ($F(1{,}3) = 0.033, p = 0.867$). On the other hand, there was a statistically significant main effect of `Refactoring Authorship` ($F(1{,}94) = 9.193, p = 0.003$) and the estimated coefficient for `Refactoring Authorship` ($b = -1.920, se = 0.633, p = 0.003$ see Appendix) indicates that participants scored higher when using GenAI-assisted code, with an average increase of about 1.92 points. Similar to the above models, there were not noted statistically significant learning/fatigue or sequence effects ($\chi^2(2) = 3.739, p = 0.154$).

Finally, although both LRT ($\chi^2(1) = 3.966, p = 0.046$) and ANOVA results ($F(1{,}93) = 3.965, p = 0.046$) indicate a statistically significant interaction between `Refactoring Authorship` and `Refactoring Type` on correctness scores for ***Extension Mechanism Identification*** (Figure 4e), the post-hoc analysis did not reveal any significant pairwise differences between the estimated marginal means after p-value adjustments. Hence, further experimentation is needed, as there is an indication of varied performance between GenAI-assisted and Human-only solutions across the two tasks but insufficient empirical evidence to support generalization. LRT also indicated no learning/fatigue or sequence effects ($\chi^2(2) = 4.747, p = 0.094$).

> Some steps of the refactoring process are more probable to be effectively assisted by GenAI, compared to others. In particular: Human-only solutions outperform AI-Assisted solutions for problem identification, identification of compromised QA (only for DPs), (b) GenAI-Assisted solutions outperform Human-only solutions for practice implementation, whereas (c) for practice selection and identification of extension mechanisms, the two treatments exhibit no difference.

4.2 Prompt Engineering for GenAI-Assisted Refactoring

In Figure 5, we present the correlation coefficients between the prompt input metrics and the correctness scores of each refactoring step. Positive correlations are marked with blue bars, negative correlations with red bars, and statistically significant correlations are reported with the use of two stars (**) for correlations with a p-value<0.01 and one star (*) for correlations with a p-value<0.05. Out of the prompt input metrics analysis, we can easily identify that ***Instruction Detail*** is the only metric that is strongly correlated with the correctness scores for all refactoring steps. Out of the other metrics, the only statistically significant correlation is between ***Total Prompt Length*** and the Correctness of Practice Implementation. Regarding Instruction Detail, we can see that it is mostly related to the Correctness of ***Problem Identification***, Correctness of ***Practice Implementation***, and Correctness of ***Extension Mechanism Identification***. Regarding the signs of the relations (it was hypothesized that all relations would be positive, based on the definition of the metrics), we can see that the ***Correctness of Problem Identification*** and ***Correctness of Extension Mechanism Identification*** are negatively correlated with the prompt input metric scores. However, these correlations have high p-values and are not statistically significant.

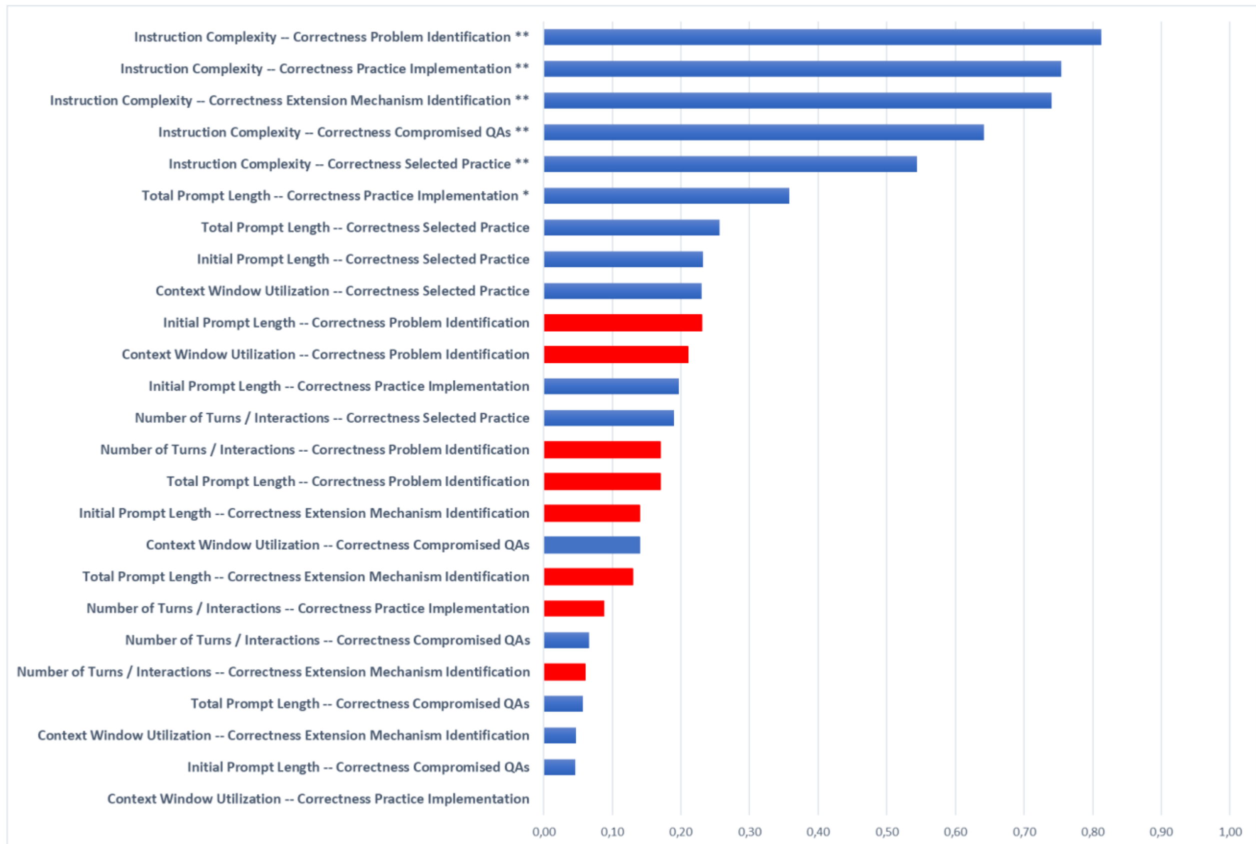

Figure 5: Correlation Analysis of Prompt Input Metric Scores and Refactoring Steps' Correctness

Instruction Detail is the only prompt input metric that seems to be correlated to the correctness of refactoring steps. Practice Implementation Correctness is also correlated to Total Prompt Length. Correctness of Problem Identification and Correctness of Extensions Mechanism Identification seem to be negatively correlated to prompt input metrics (except Instruction Detail).

5. DISCUSSION

5.1 Answering the Research Questions and Interpretation of Results

The findings of this study suggest that we cannot uniformly answer RQ_1, since *there is no clear "winner" between Human-only and AI-Assisted solutions in the refactoring of code to DPs or SPs*. The differences can be attributed to both the *nature of each step of the refactoring process* (in some Human-only outperformed AI-Assisted, and for other the vice versa)—$RQ_{1.1}$, but also the *nature of the practice (patterns or principles) under consideration*—$RQ_{1.2}$. This fact can be considered intuitive, in the sense that human skills and "*AI skills*" are expected to be different, context-dependent and sensitive to the given task when it comes to software quality improvement. We can observe that: (a) ***Human-only*** solutions outperform AI-assisted solutions for *problem identification, identification of compromised QA* (only for DPs), (b) ***GenAI-assisted*** solutions outperform Human-only solutions for *practice implementation*, whereas (c) for *practice selection* and *identification of extension mechanisms*, the two treatments showcase no difference. These findings are considered very interesting in the sense that the quality improvement tasks, seem to "*resist*" the belief that "*AI is eating the world*", at least with the existing technologies. On the one hand, from our analysis it seems that even junior software engineers can be more specific in the identification of quality problems, if they are given well-established guidance / reading material to consult. This task is by nature more cognitive and re-

quires critical thinking, which seems to be better possessed by humans, compared to AI, which possesses access to a huge pool of knowledge thereby providing a lot of information, without necessarily prioritizing what is important. This aligns with the observations of Martins et al. (2024) regarding code reviews, where LLMs struggled with abstract principles requiring deep semantic understanding. Additionally, our findings confirm the work of Liu et al. (2025a), who found that without specific prompting (which corresponds to our "*problem identification"* phase), LLMs fail to autonomously identify refactoring opportunities in complex scenarios. This is also evidenced by the superiority of Humans in identifying the pain points in each system, expressed by quality attributes that are compromised by an inefficient design or code. On the other hand, the ability of AI to produce more accurately a technical solution (such as writing code) to a well-defined problem based on numerous patterns that is has been exposed to, is non-negligible, and it seems that humans cannot compete with it. The findings of this study reveal that because design patterns and design principles have been well documented in the literature, with numerous examples and templates, AI-generated implementations excel in effectively applying the requested solution, as specified by the (human) user. This result supports the general consensus in the literature (Vaithilingam et al., 2022; Lyu et al., 2025) that LLMs are highly effective at generating syntactically correct code once the context is clear. This finding is further verified by Kim (2025b), who reported exceptionally high performance metrics for the Visitor pattern implementation using ChatGPT. Our results suggest that from standard textbook definitions of GoF patterns, modern models (like GPT-4.1 mini) have reached a level of maturity that outperforms junior developers in mechanical implementation. Another issue that cannot be negotiated for AI-assisted tasks is the absence of very low scores in their performance, meaning that through AI every given task pertaining to quality improvement of software can be accomplished decently. It should be noted that processing time for completing each task was not a factor in this experiment. In terms of specific practices, apart from the *identification of compromised QAs*, we have observed that SOLID principles (SPs) are "*harder*" to identify using both treatments, compared to the identification of design patterns (DPs). This finding can be explained in two ways: (a) DPs are more widely known among practitioners as a means for exploring the space of object-oriented design compared to SPs; or/and (b) that SPs are broader in scope than DPs and are usually more abstractly defined in educational and web sources and therefore, the relevant knowledge on the subjects is not easily applicable for both humans and AI. This observation extends the findings of Martins et al. (2024), who identified a performance gap between syntax-oriented principles and abstract principles requiring deeper semantic understanding. In more detail, we could argue that SPs provide the foundational rules of good designs, whereas DPs provide the concrete tools and structures to implement those rules in practice, solving specific problems while adhering to the principles (McLean, 2017) (Singh and Hassan, 2015). Apart from an academic point of view, these statements also seem to be supported by developers' forums and blogposts[2,3,4]. Finally, relating our findings back to the methodological background presented in Section 2.2, the use of the crossover design (Vegas et al., 2016) proved instrumental. By controlling inter-subject variability, we were able to isolate these nuanced differences between cognitive and technical steps.

With respect to RQ2 the main findings can be summarized as follows: (a) Instruction Detail is the only prompt input metric that seems to be correlated to the correctness of refactoring steps; (b) Practice Implementation Correctness is also correlated to Total Prompt Length; and (c) Correctness of Problem Identification and Correctness of Extensions Mechanism Identification seem to be negatively correlated to prompt input metrics (except Instruction Detail). The first finding suggests that qualitative evaluations of prompts as captured by conceptual aspects of the posed question, such as the concreteness of the prompt, the specificity and the descriptiveness of the prompt and the domain

[2] https://blog.savetchuk.com/the-difference-between-design-patterns-and-design-principles
[3] https://www.quora.com/Should-you-learn-SOLID-before-design-patterns
[4] https://medium.com/@kenslearningcurve/design-patterns-vs-design-principles-d1c9fe032dc

knowledge that is passed through the prompt to the LLM are more important compared to structural metrics (e.g., length and number of iterations). However, these structural properties of prompts need to be examined in larger tasks. Regarding specific refactoring steps, the results suggested that Practice Implementation Correctness, which is the step that is mostly assisted by GenAI is related to two metrics is also important, suggesting that the place where GenAI is most useful can be further exploited by correct and accurate prompts. On the contrary, the steps that the neutral steps (Problem Identification and Extensions Mechanism Identification) are negatively correlated to lengthy prompts. This finding suggests that a different prompting style in the specific steps could improve their effectiveness; and maybe lead to GenAI-Assisted solution excel compared to Human-only solutions. However, this also needs further investigation.

5.2 Implications for Researchers

First, through this study we have confirmed that cross-over experimental designs can provide answers to interesting research questions, employing the Linear Mixed Effects analysis method. The identification of relations between treatments and possible confounding factors were accurately identified, thus we encourage the use of the same study design setup, for fitting purposes. More specifically, cross experimental designs reveal whether factors (such as `Treatment`, that is AI-assisted or Human-only approach and `Problem Type`, i.e. DPs or SPs, in our study), act independently or jointly on the correctness of the obtained solutions when refactoring code to improve its quality. Second, with respect to AI4SE researchers, we encourage the further development and exploration of code-specific LLMs rather than general-purpose ones (like ChatGPT) for software engineering tasks, since general-purpose models might exhibit inferior performance to humans. Finally, with respect to the academic community, we encourage the development and deployment of additional sources (books, blogs, papers, etc.) on SPs, which currently do not seem sufficient for either LLMs or Humans to use and lead to the suitable identification for the need to apply an SP (at least compared to DPs). In any case, we urge prospective software engineers to build strong foundations in core principles; otherwise, they may soon lack the knowledge to validate the output of AI-generated solutions, particularly in terms of quality.

5.3 Implications for Practitioners

The major implication for practice out of this experiment has to do with general guidance to practitioners to not rely blindly on AI-assistance, especially for merely cognitive tasks, like problem identification. The suggested practice would be to use AI-assistance for compiling a list of possible problems, and then dive into details of each suggestion, either by self-studying on web or traditional sources to evaluate and prioritize them. Of course, a viable alternative to this would be to not "*stay*" at a shallow "*discussion*" with AI and get suggestions, but ask AI to explain each problem, prioritize and in finally co-decide which identified problem is the one that is the most important or influential given the requirements of the task. We see this path as productive both for reaching a high quality in the software but also for educating software engineers through the process. Furthermore, we strongly encourage software engineers to make use of AI for the "*actual*" refactoring process, i.e., when the problem is clear, when the compromised QAs are identified, and the selection of a solution is made in a collaborative manner, the human can rely on AI for the actual writing of the code. However, even in this step, the execution of tests and manual inspection of code is required. Finally, the analysis of prompts gives two main takeaways to practitioners when using LLMs for code refactoring to DPs and SPs. The first has to do with the provision of as much context as possible; providing domain knowledge, e.g., on the fact that the solution must use a specific DP or SP or guiding with respect to the goals of the envisioned refactoring. The second high-level guideline is to use smaller prompts for each step of the refactoring process, i.e., the identification of the problem and the identification of extension mechanisms. However, both recommendations need to be further explored.

5.4 Future Work

We believe that an interesting extension to this work would be a full-fledged qualitative (lexical and semantic) study of prompts that are used for interacting with LLMs when performing refactoring tasks and producing guides of interaction through prompt engineering. Although this work has performed an initial analysis towards this direction, we have not performed analysis of the content of prompts, which we believe can substantially improve the level of detail in the prompting guidelines. Equally interesting would be to further explore parts of the refactoring process that are currently indecisive. Since in many cases we observed comparable performance between the refactoring task that is carried out with and without the help of GenAI, it would be interesting to explore the time required for reaching each solution, and especially if there are cases where humans opt for directly drafting a solution avoiding the overhead of effectively structuring a prompt to an LLM. Also, replications with different practices, senior developers and other LLMs are strongly encouraged to achieve generalizability.

6. THREATS TO VALIDITY

Although the primary objective of this study is to shed light into the effectiveness of code produced with and without GenAI assistance in carrying out refactoring steps to improve software quality through design patterns and SOLID principles, it is crucial to acknowledge potential threats to the validity of findings and categorize them to the well-established schema proposed by Wohlin et al. (2024). Regarding study ***replicability***, we have developed a full replication package including: (a) the material provided to human participants; (b) the task descriptions and code; (c) the dataset of evaluations; (d) the prompts that have been used when using GenAI; and (e) the submitted responses. The replication package is available online in GitHub[5].

Regarding ***external validity***, the reported findings stem from a controlled experiment involving 10 junior developers and as a result caution should be taken in the generalization of the results to different scenarios involving senior software engineers, programming paradigms that urge for organization in teams, different application domains, programming languages, environments, etc. Such threats can only be mitigated by replicating the study; something which is imperative given the pace by which GenAI capabilities advance. In addition, the generalizability of the results to the unknown population of junior developers may also be hindered by the relatively limited sample size used for inferential purposes. The sample size in the experiment, although quite low for quantitative analysis, is justified by the effort required for conducting experiments in the domain of software engineering. The experiment was time-consuming, requiring the participation of professional (even junior) developers for a complete week every afternoon. To this end recruiting volunteers for such a task was non-trivial. According to Falezzi et al. (2017) it is very difficult to attract professional software engineers as participants in experiments, with only 33% of invited professionals ending up to participate in the studies. In addition to this, according to Sjoberg et al. (2010) the typical participation in SE experiments varies from 4 to 60 professional software engineers, with a mean value of 20. Finally, according to Vegas et al. (2016) the sample size issue is mitigated by the repeated measures design, enabling the gathering of 100 data points from the participants; a number which renders possible further statistical analysis. Although a priori power analysis would result in an approximated sample size needed for fulfilling the intended scopes of the current study, we decided not to conduct a priori power analysis due to both practical and technical reasons. First, due to the originality of the topic discussed in this study and, to the best of our knowledge, there was no prior empirical evidence available to reliably estimate key parameters, such as the expected effect size and variance components of random effects, which are necessary inputs for conducting a priori power analysis in order to determine the required sample size. Second, analytical formulae for sample size calculation are only available for simple ex-

[5] https://github.com/stavrosal/jss-paper/

perimental designs and traditional statistical hypothesis procedures, such as parametric t-tests, analysis of variance (ANOVA), or chi-square test for independence. For more complex designs, such as repeated measures designs etc., with multiple factors being examined and complex variance structures, there are no available analytical formula, and the only option would be to conduct power analysis via simulation approaches. In this regard, such approaches involve the specification of plausible effect sizes and conduction of power analyses on these predefined levels (small, medium and high). While such simulation-based approaches provide flexibility, it does not guarantee that the final selected effect size accurately reflects the true magnitude. Summarizing, although we acknowledge the limitation associated with the relatively small sample size used for inferential purposes, we opted to use a robust statistical modeling technique to appropriately exploit the available information contained in the collected data and mitigate potential external validity threats. Specifically, we adopted Linear Mixed Effects models that are considered well-suited to account for participant-level (developer-level in our case) and refactoring practice-level random effects modeling the individual heterogeneity and refactoring practice variability. The insertion of random effects safeguards the estimation of the parameters (coefficients) for the primary factors of interest (fixed-effects) ("Refactoring Authorship" and "Refactoring Type" along with their interaction in our study) while complementing formal hypothesis testing and uncertainty quantification via the construction of 95% confidence intervals.

Additionally, the adopted within-subject repeated-measures design with daily AB/BA counterbalancing, where each participant completed 10 sessions of distinct refactoring task combinations across 5 days, may introduce two important ***internal validity*** threats. The first internal validity is associated to the potential learning or fatigue effects, as repeated exposure to similar refactoring tasks could lead to improvement in performance (learning) or declines due to fatigue, particularly in the later sessions, affecting in turn, the correctness scores. The second internal validity threat concerns sequence effects, where the order in which participants executed daily refactoring tasks may introduce bias to the extracted findings. Although such effects are inherent in repeated-measures designs and longitudinal studies, the use of Linear Mixed Effects models enabled us to examine such effects by including and testing appropriate covariates in the final models.

In terms of experimental design, a ***construct validity*** threat that is valid pertains to the fact that the GenAI group does not assume a pure AI solution, in the sense that the user guides the employed LLM based on his/her experience. Unavoidably, this might lead to bias as the software engineer through his/her prompts might shift the model's output towards one or the other direction. For this reason, we mainly stick with the term AI-assisted solutions as this is the most probably scenario of AI4SE in the coming years, that is, a human software engineer will act the driver in human-AI interactions. Additionally, the construct validity of the study may be threatened by the design of the refactoring tasks that were assigned to participants. The tasks have been designed to match the timeframe of the experiment (1 hour) and to be as close as possible to the rationale of the specific refactoring, being inspired from textbook examples. Moreover, although the five scores (Problem Identification, Compromised QAs, Selected Practice, Practice Implementation, Extension Mechanism Identification) were carefully designed to quantify the effectiveness of applying a refactoring process, different team might approach refactoring in a more ad/hoc way. Finally, the subjectivity of the raters in assessing correctness scores may pose a threat to internal validity, considering that the grading scheme provides some flexibility (i.e., not all possible scores are included in the schema, but only some landmark scores). This was mitigated by involving a third assessor, in cases where the scores from the two primary raters (first two authors of this study) differed by more than 20%; and the use of detailed grading mechanism.

As far as ***conclusion validity*** is concerned, it has been already mentioned that the combination of a cross-over repeated measures design with Linear Mixed Effects models helps mitigate threats to conclusion and statistical validity. This adopted approach accounts for within-subject variability, while controlling for participant- and task-specific

random effects. Thus, the different sources of variability are appropriately taken into consideration, providing in turn robust estimation of the fixed-effects parameters (coefficients), while quantifying the uncertainty inherent in complex experimental setups.

7. CONCLUSIONS

Triggered by the omnipresent use of LLMs in everyday software development and concerns about the quality of software solutions delivered by GenAI, the goal of this study was to investigate the ability of GenAI to assist humans in refactoring to DPs and SPs. Through a controlled experiment involving junior developers, we compared the correctness of five refactoring steps (understanding problem constraints, reasoning about compromised quality aspects, identifying applicable patterns or principles, instantiating the solution and identifying an extension scenario benefiting from the refactoring) between Human-only and AI-Assisted solutions. The use of a cross-over experimental design proved powerful for identifying potential relations between treatments and possible confounding factors. The experiment revealed a nuanced landscape in the collaboration between Human-only and AI-Assisted solutions across different refactoring tasks. While Human-only solutions excelled in cognitive and problem-identification steps, especially for design principles, AI-Assisted solutions demonstrated superior performance in implementing well-defined solutions. These findings highlight the complementarity of humans and AI and underline the importance of acquiring solid foundational knowledge for engineers to be able to effectively validate AI outputs.

REFERENCES

A. Ampatzoglou, A. Chatzigeorgiou, S. Charalampidou and P. Avgeriou, "The Effect of GoF Design Patterns on Stability: A Case Study", Transactions on Software Engineering, 41 (8), pp. 781-802, 1 Aug. 2015.

A. Ampatzoglou, S. Charalampidou, and I. Stamelos, “Research state of the art on GoF design patterns: A mapping study,” Journal of Systems and Software, 86 (2), pp. 1945–1964, July 2013.

A. Ampatzoglou, A.-A. Tsintzira, E.-M. Arvanitou, A. Chatzigeorgiou, I. Stamelos, A. Moga, R. Heb, O. Matei, N. Tsiridis, and D. Kehagias, “Applying the Single Responsibility Principle in Industry: Modularity Benefits and Trade-offs”, 23rd International Conference on Evaluation and Assessment in Software Engineering (EASE '19), Copenhagen, Denmark, April 15 - 17, 2019.

R. Andrade, J. Torres, and I. Ortiz-Garcés, “Enhancing Security in Software Design Patterns and Antipatterns: A Framework for LLM-Based Detection”, Electronics, 14 (3), 586, 2025.

D. Bates, M. Mächler, B. Bolker, and S. Walker, “Fitting linear mixed-effects models using lme4. Journal of statistical software”, 67, pp. 1-48, 2015.

J. M. Bland and D.G. Altman, “Calculating correlation coefficients with repeated observations: Part 1 -- correlation within subjects”. BMJ, 310, 446., 1995a

J. M. Bland and D.G. Altman, “Calculating correlation coefficients with repeated observations: Part 2 -- correlation within subjects”, BMJ, 310, 633, 1995b.

T. J. Cleophas, “Human Experimentation. Methodologic Issues Fundamental to Clinical Trials”, Norwell, MA, USA: Kluwer, 1999.

S. Charalampidou, A. Ampatzoglou, P. Avgeriou, S. Sencer, E. M. Arvanitou, and I. Stamelos, “A theoretical model for capturing the impact of design patterns on quality: the decorator case study”, Symposium on Applied Computing (SAC '17), 2017.

M. Fowler, “Refactoring: Improving the Design of Existing Code”, 2nd Edition, Addison-Wesley, 2018.

J. Frattini, D. Fucci, and S. Vegas, “Crossover Designs in Software Engineering Experiments: Review of the State of Analysis”, 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24), Barcelona, Spain, October 24 - 25, 2024.

D. Falessi, N. Juristo, C. Wohlin, et al. “Empirical software engineering experts on the use of students and professionals in experiments”, Empirical Software Engineering, 23, 452–489, 2018.

E. Gamma, R. Helms, R. Johnson, and J. Vlissides, “Design patterns: elements of reusable Object-Oriented software”, Addison-Wesley Professional, 1995.

P. Haindl and G. Weinberger, "Does ChatGPT Help Novice Programmers Write Better Code? Results from Static Code Analysis", IEEE Access, 12, pp. 114146-114156, 2024.

S. Imai, “Is GitHub Copilot a Substitute for Human Pair-programming? An Empirical Study”, 44th International Conference on Software Engineering: Companion Proceedings, (ICSE Companion 2022), Pittsburgh, PA, USA, May 22-24, 2022.

C. Izurieta and J. M. Bieman, “A multiple case study of design pattern decay, grime, and rot in evolving software systems”, Software Quality Journal, 21, pp. 289–323, 2013.

J. Kerievsky, “Refactoring to Patterns”, Addison-Wesley, 2004.

D.-K. Kim, "Comparative analysis of design pattern implementation validity in LLM-based code refactoring", Journal of Systems and Software, 230, 2025a.

D.-K. Kim, “Quantitative Assessment of Generative Large Language Models on Design Pattern Application”, Computers, Materials & Continua, 82 (3), pp.3843-3872, March 2025b.

R. Laue, J. J. Maranhão, and E. M. Guerra, “Asking ChatGPT for Pattern Recommendations: EuroPLoP 2024 Focus Group Report”, 29th European Conference on Pattern Languages of Programs, People, and Practices (EuroPLoP '24), Irsee, Germany, July 3 - 7, 2024.

B. Liu, Y. Jiang, Y. Zhang, et al., “Exploring the potential of general purpose LLMs in automated software refactoring: an empirical study”, Automated Software Engineering, 32, 26, 2025a.

N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang, “Lost in the Middle: How Language Models Use Long Contexts”, Transactions of the Association for Computational Linguistics, 173, pp. 12-157, 2024.

Y. Liu, S. K. Lo, Q. Lu, L. Zhu, D. Zhao, X. Xu, S. Harrer, and J. Whittle, "Agent design pattern catalogue: A collection of architectural patterns for foundation model-based agents", Journal of Systems and Software, 220, 2025b.

M. R. Lyu, B. Ray, A. Roychoudhury, S. H. Tan, and P. Thongtanunam, “Automatic Programming: Large Language Models and Beyond”, ACM Transactions on Software Engineering and Methodology, 34 (5), Article 140, pp. 1-33, June 2025.

L. Madeyski and B. Kitchenham, “Effect sizes and their variance for AB/BA crossover design studies”, Empirical Software Engineering, 23, pp. 1982–2017, 2018.

A. Maha, M. S. Mohamud, E. Masuadi, M. A. Altowejri, A. Farraj, H. G. Schmidt, “The Effect of Using Native versus Nonnative Language on the Participation Level of Medical Students during PBL Tutorials”, Health Professions Education, Elsevier, 6 (4), 2020.

R. C. Martin, “Agile software development: principles, patterns, and practices”, Prentice Hall PTR, Upper Saddle River, USA, 2003.

G. McLean, “Adaptive Code: Agile coding with design patterns and SOLID principles (Developer Best Practices)”, Microsoft Press, 2nd Edition, 2017.

G. F. Martins, E.C.M. Firmino, and V. P. De Mello, “The Use of Large Language Model in Code Review Automation: An Examination of Enforcing SOLID Principles”, Degen, H., Ntoa, S. (eds) Artificial Intelligence in HCI. HCII 2024. Lecture Notes in Computer Science(), vol 14736, 2024.

S. Moreschini, E. M. Arvanitou, E.-P. Kanidou, N. Nikolaidis, R. Su, A. Ampatzoglou, A. Chatzigeorgiou, and V. Lenarduzzi, “The Evolution of Technical Debt from DevOps to Generative AI: A Multivocal Literature Review”, Journal of Systems and Software, August 2025.

Z. Pan, X. Song, Y. Wang, R. Cao, . Li, Y. Li, and H. Liu, “Do Code LLMs Understand Design Patterns?”, International Workshop on Large Language Models for Code (LLM4Code’25), Co-Located with ICSE 2025, Ottawa, Canada, 2 April – 3 May 2025.

S. K. Pandey, S. Chand, J. Horkoff, et al., “Design pattern recognition: a study of large language models”, Empirical Software Engineering, 30, 69, 2025.

B. Postle and N. A. Salingaros, “LLM and Pattern Language Synthesis: A Hybrid Tool for Human-Centered Architectural Design” Buildings, 15 (14), 2400, 2025.

L. Qin, Q. Chen, Y. Zhou, Z. Chen, Y. Li, L. Liao, M. Li, W. Che, P. S. Yu, “A survey of multilingual large language models”, *Patterns*, Elsevier, 6 (1), 2025.

R Core Team, “R: A language and environment for statistical computing. R Foundation for Statistical Computing”, R Foundation for Statistical Computing, 2025.

S. I. Ross, F. Martinez, S. Houde, M. Muller, and J. D. Weisz, “The Programmer’s Assistant: Conversational Interaction with a Large Language Model for Software Development”, 28th International Conference on Intelligent User Interfaces (IUI '23), Sydney, Australia, March 27 - 31, 2023.

N. Salem, A. Hudaib, K. Al-Tarawneh, H. Salem, A. Tareef, H. Salloum, and M. Mazzara, “A survey on the application of large language models in software engineering”, Computer Research and Modeling, 16 (7), pp. 1715–1726, 2025.

G. Scanniello, C. Gravino, M. Genero, J. A. Cruz-Lemus, and G. Tortora, “On the impact of UML analysis models on source-code comprehensibility and modifiability”, Transactions on Software Engineering and Methodology, 23 (2), pp. 13:1–13:26, 2014.

D. I. K. Sjoeberg et al., "A survey of controlled experiments in software engineering," IEEE Transactions on Software Engineering, vol. 31, no. 9, pp. 733-753, Sept. 2005

H. Singh and S. I. Hassan, “Effect of SOLID design principles on quality of software An empirical assessment”, International Journal of Scientific & Engineering Research, 6 (4), April 2015.

P. Smiari, S. Bibi, A. Ampatzoglou, and E. M. Arvanitou, "Refactoring Embedded Software: A Study in Healthcare Domain", Information and Software Technology, Elsevier, February 2022.

P. Vaithilingam, T. Zhang, and E. L. Glassman, “Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models”, CHI Conference on Human Factors in Computing Systems (CHI EA '22), New Orleans, LA, USA, 29 April - 5 May 2022.

S. Vegas, C. Apa and N. Juristo, "Crossover Designs in Software Engineering Experiments: Benefits and Perils," Transactions on Software Engineering, 42 (2), pp. 120-135, 1 Feb. 2016.

F. Wedyan and S. Abufakher, “Impact of design patterns on software quality: a systematic literature review”, IET Software, 14 (1), pp. 1–17, February 2020.

S. Wellek and M. Blettner, “On the proper use of the crossover design in clinical trials: part 18 of a series on evaluation of scientific publications”, Deutsches Ärzteblatt international, 109 (15), pp. 276-81, April 2012.

R. J. Wieringa, “Design science methodology for information systems and software engineering”. Springer, 2014.

J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, “ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design”, Nguyen-Duc, A., Abrahamsson, P., Khomh, F. (eds) Generative AI for Effective Software Development. Springer, Cham, June 2024.

C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, “Experimentation in Software Engineering”, New York, NY, USA: Springer, 2024.

J.D. Zamfirescu-Pereira, R. Y. Wong, B. Hartmann, and Q. Yang, “Why Johnny Can’t Prompt: How Non-AI Experts Try (and Fail) to Design LLM Prompts”, CHI Conference on Human Factors in Computing Systems (CHI '23), Hamburg, Germany, April 23 - 28, 2023.

A. Ziegler, E. Kalliamvakou, X. A. Li, A. Rice, D. Rifkin, S. Simister, G. Sittampalam, and E. Aftandilian, “Productivity assessment of neural code completion”, 6th International Symposium on Machine Programming (MAPS@PLDI 2022), San Diego, CA, USA, 13 June 2022.

C. Zhang and D. Budgen, “What do we know about the effectiveness of software design patterns”, Transactions on Software Engineering, 38 (5), pp. 1213–1231, Sep./Oct. 2012.

APPENDIX: DETAILED RESULTS

Dependent	**Effect**	**Term**	***Estimate***	***SE***	***t***	***df***	***p***	***Variance***
Problem Identification	Fixed	Intercept	5.473	0.800	6.841	3.760	0.003	
		Refactoring Authorship:*Human-only*	1.102	0.523	2.106	85.003	0.038	
		Refactoring Type:*SP*	0.118	1.194	0.099	3.000	0.928	
	Random	Intercept for Participant-ID						0.014
		Intercept for Refactoring Name						1.369
		Residual						6.841
Compromised QAs	Fixed	Intercept	5.033	0.631	7.980	7.950	0.000	
		Refactoring Authorship:*Human-only*	1.818	0.695	2.617	84.000	0.011	
		Refactoring Type:*SP*	1.067	0.943	1.131	6.807	0.296	
		Refactoring Authorship:*Human-only*×Refactoring Type:*SP*	-2.196	1.099	-1.999	84.000	0.049	
	Random	Intercept for Participant-ID						0.420
		Intercept for Refactoring Name						0.343
		Residual						7.244
Selected Practice	Fixed	Intercept	6.219	0.585	10.630	37.43	0.000	
		Refactoring Authorship:*Human-only*	-0.131	0.667	-0.196	88.00	0.845	
		Refactoring Type:*SP*	-1.498	0.681	-2.202	88.00	0.030	
	Random	Intercept for Participant-ID						0.459
		Residual						11.115
Practice Implementation	Fixed	Intercept	8.148	0.584	13.963	5.991	0.000	
		Refactoring Authorship:*Human-only*	-1.889	0.633	-2.983	94.000	0.004	
		Refactoring Type:*SP*	0.130	0.775	0.167	3.000	0.878	
	Random	Intercept for Refactoring Name						0.220
		Residual						10.022
Extension Mechanism Identification	Fixed	Intercept	6.733	0.718	9.373	7.93	0.000	
		Refactoring Authorship:*Human-only*	1.045	0.896	1.166	93.00	0.247	
		Refactoring Type:*SP*	0.817	1.136	0.719	7.93	0.493	
		Refactoring Authorship:*Human-only*×Refactoring Type:*SP*	-2.817	1.417	-1.988	93.00	0.050	
	Random	Intercept for Refactoring Name						0.344
		Residual						12.041