UFMG

lab-soft
software engineering laboratory

From Requirements to Code:
Assessing the Impact of Foundation Models on Software Development

Eduardo Figueiredo
Federal University of Minas Gerais (UFMG)

05 March 2026

About Me

- Associate professor at UFMG (since 2010)
- Head of the Software Engineering Laboratory (LabSoft)
- Academic Background
  - [2006 - 2009] PhD in Computer Science
    Lancaster University, UK
  - [2017 - 2018] One-Year Sabbatical
    Carnegie Mellon University (CMU), US
  - [2025 - 2026] One-Year Sabbatical
    University of Ottawa (uOttawa), Canada


Jon Whittle


Alessandro Garcia


Christian Kastner


Lionel Briand

LabSoft Research

- Our research spams a broad range of SE topics
  - AI and foundation models for Software Engineering
  - Software metrics and static program analysis
  - Software testing, verification, and validation
  - Development for mobile devices
  - Self-adaptive and IoT systems
  - Mining software repository
  - Open-source software development

lab-soft
software engineering laboratory

LabSoft Current Members

- 3 Post-Doc Researchers
  - Filipe Fernandes (SE for the Metaverse, Quantum SE)
  - Gustavo Vale (Quality of AI-generated Code, Merge Conflicts)
  - Fischer Ferreira (Software Testing, Resource-Intensive Systems)
- 6 PhD Candidates
  - Joao Paulo Diniz (Mutation Testing)
  - Amanda Santana (Code Smell Detection)
  - Henrique Nunes (LLMs for Architecture and Code Smells)
  - Cleiton Tavares (Quality of AI-generated code)
  - Pedro Garcia (Impact of AI on Education)
  - Saymon Souza (LLMs for Refactoring Code Smells)
- 7 MSc and a few undergrade students

lab-soft
software engineering laboratory

Some LabSoft Members

Amanda Santana (PhD)

Cleiton Tavares (PhD)

Filipe Fernandes
(Post-Doc)

Henrique Nunes (PhD)

João Paulo Diniz (PhD)

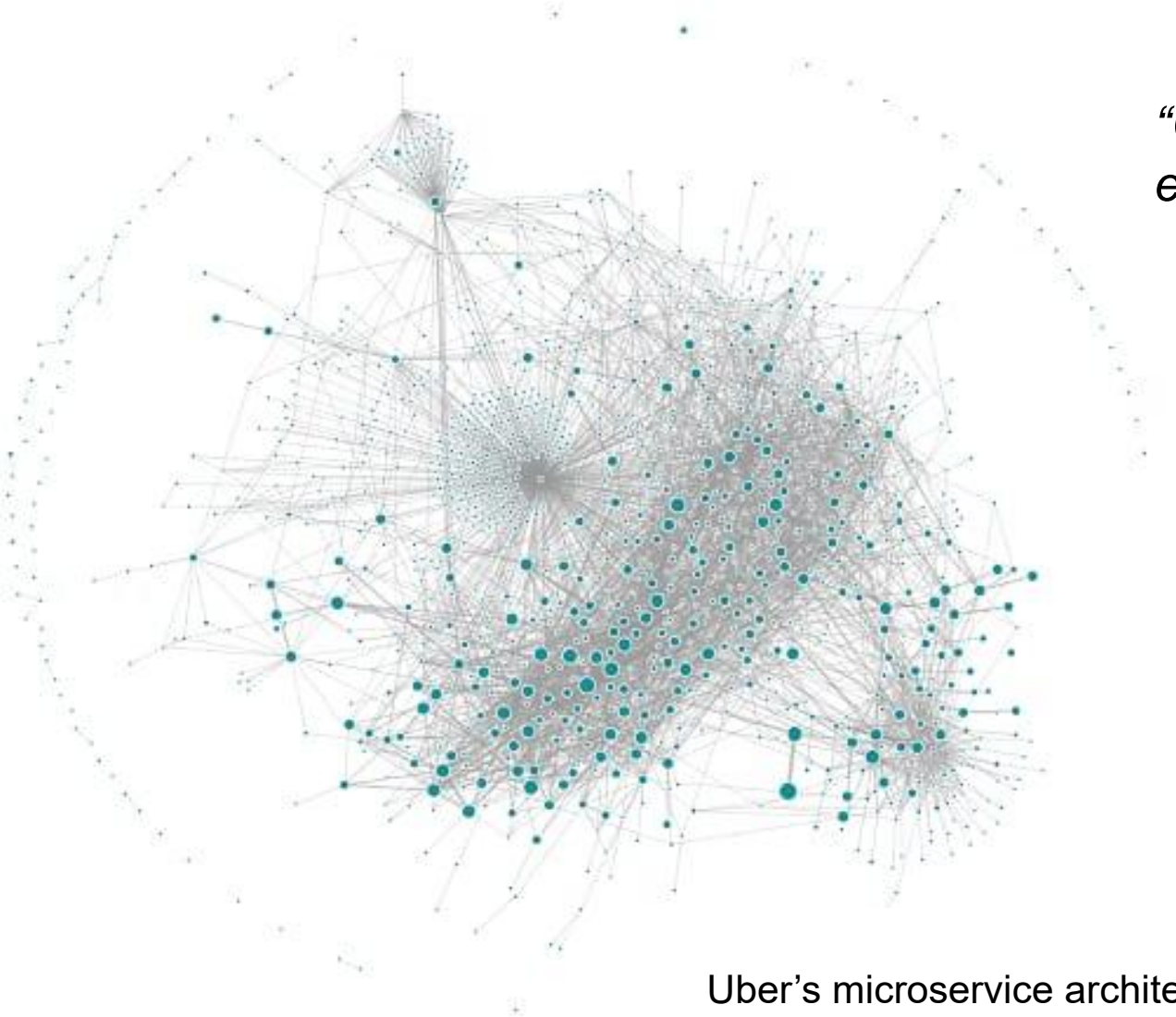
Fischer Ferreira
(Post-Doc)

Pedro Garcia (PhD)

Saymon Souza (PhD)

Gustavo Vale
(Post-Doc)

Software is Complex



“Computer software is the most complex entity among human-made artifacts.”

Frederick P. Brooks, 1987

“Good programmers write code that humans can understand.”

Martin Fowler, 2008

Uber’s microservice architecture (2018)

How much software is created by humans?

Over 25% of Google's code is now written by AI —and CEO Sundar Pichai says it's just the start

BY **GREG MCKENNA**
NEWS FELLOW

October 30, 2024 at 11:14 AM EDT

TECH

Satya Nadella says as much as 30% of Microsoft code is written by AI

PUBLISHED TUE, APR 29 2025•9:33 PM EDT | UPDATED TUE, APR 29 2025•9:58 PM EDT

Jordan Novet @IN/JORDANNOVET/ | **Jonathan Vanian** @IN/JONATHAN-VANIAN-B704432/

SHARE

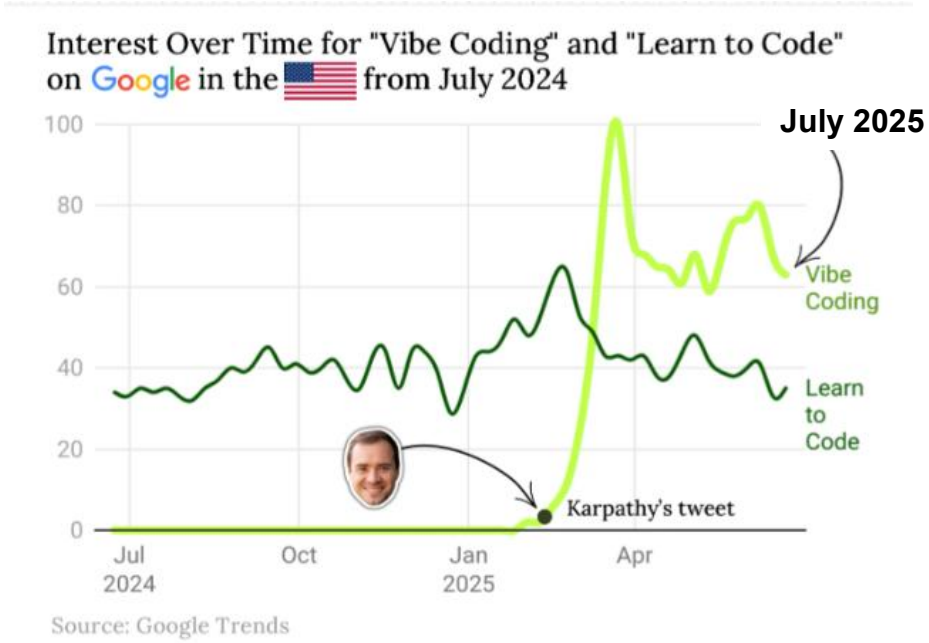
Mark Zuckerberg wants AI to do half of Meta's coding by 2026

The comments came during a discussion between Zuckerberg and Microsoft CEO Satya Nadella.

By Cecily Mauran on April 29, 2025

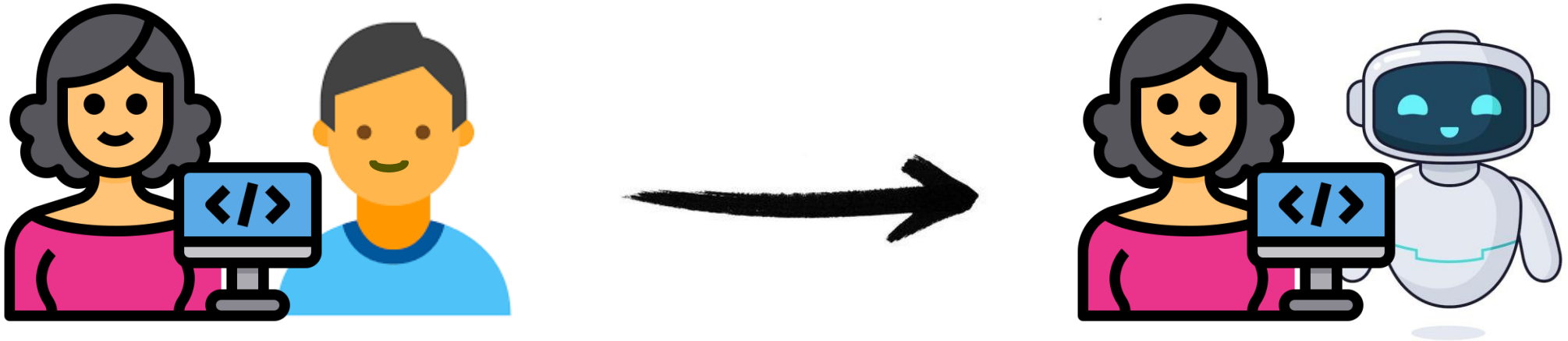
What do we expect for the near future?

- Vibe Coding?
  - Vibe coding searches are up 6,700% in 3 months (Mar-May)



- AI Coding Agents?
  - According to 2025 Stack Overflow Developer Survey, ~32% of professional developers use AI Agents

A Twist in Pair Programming

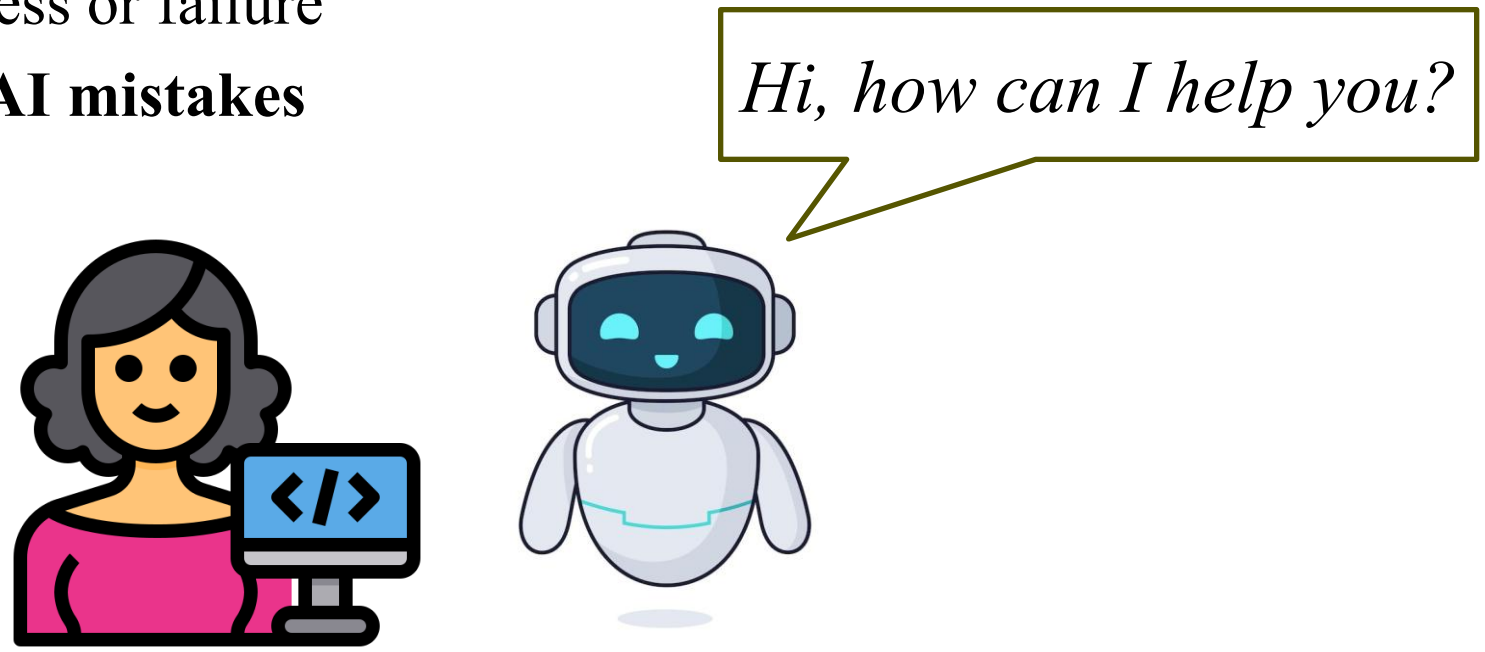


- Human-Human
  - Knowledge sharing
  - Social dynamics
- Human-AI
  - Development speed
  - 24/7 availability

Defining Roles in the Human-AI Partnership

Human

- Human is good for requirements
- Responsible for success or failure
- **Verify and correct AI mistakes**



1st Study: AI-supported Requirements

Documenting User Stories: Does LLM Help?

Pedro Garcia[1], João Vitor Depollo[1], Laura Barroso[1], Eduardo Figueiredo[1],
Kattiana Constantino[2], Filipe Fernandes[1], Filipe Roseiro Côgo[3]
[1]Federal University of Minas Gerais (UFMG), Belo Horizonte, Brazil
[2]Federal University of Vale Jequitinhonha e Mucuri (UFVJM), Diamantina, Brazil
[3]Huawei Technologies, Kingstong, Canada

ABSTRACT

The increasing capabilities of large language models (LLMs) offer a promising avenue for enhancing human-intensive software engineering activities, such as requirements elicitation. However, we lack strong empirical evidence on the impact of LLMs, such as ChatGPT, not only on the productivity and quality of requirement artifacts, such as user stories, but also on the learning of requirements elicitation. To address this gap, this exploratory study investigates the impact of ChatGPT used by 28 undergraduate students from two universities on documentation of user stories. Based on the Latin Square design, students created user stories with and without support of the LLM. We then analyzed the effect

as Connextra, defines the following structured format [11]: As a <role>; I want to <action>; so that <benefit>.

Recent advances in Large Language Models (LLMs) have boost interest in their use for software engineering problems, such as code generation [26], software quality [28], and software documentation [22]. In fact, their ability to understand complex contexts allows LLMs to generate flexible and adaptable solutions [28]. In particular, LLMs offer a promising avenue for enhancing human-intensive software engineering activities [24], such as requirements elicitation. For instance, the popularity of user stories among practitioners and their simple yet strict structure make them ideal candidates for automatic generation by LLMs [13]. However, we lack strong

Pedro Garcia (PhD)

Goal and Research Questions

- Goal
  - Evaluate the use of an LLM (ChatGPT) on **requirements engineering** activities, such as user story documentation

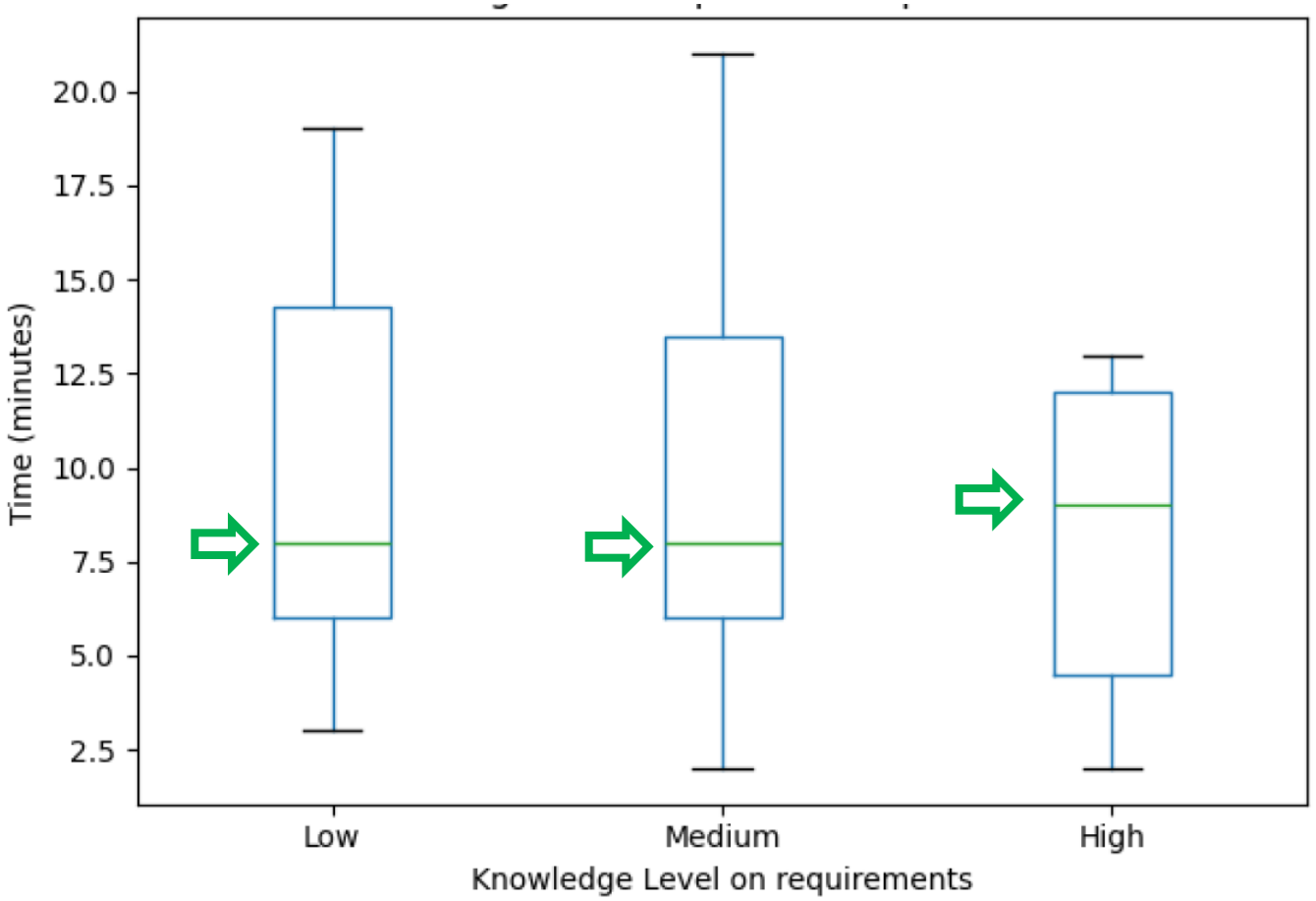
- Research Questions
  1. How LLMs impact the **time** required to create user stories?
  2. How LLMs impact the **quality** of the documented user stories?
  3. What are the **challenges** and **benefits** of using LLMs for requirements documentation?

Study Participants and Steps

- Participants: 28 CS students from two universities
  - We collect their levels of experience in software development activities, such as requirements engineering

- Research steps
  1. Participants create user stories with and without LLM support
  2. They provide feedbacks about the use of LLMs
  3. We evaluate the quality of the user stories
  4. We also analyze the participants' feedbacks

One Interesting Result



With AI-support, participants with higher knowledge take longer to complete their tasks.

These requirements do not seem right.

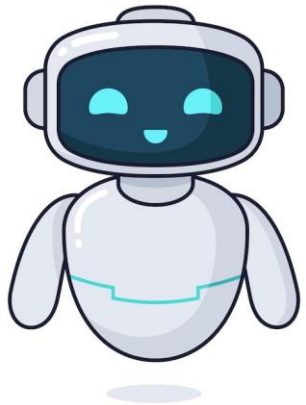
Defining Roles in the Human-AI Partnership

Human

- Human is good for requirements
- Responsible for success or failure
- Verify and correct AI mistakes

AI (Foundation Models)

- AI is good for coding
- Faster and scalable
- **Create quick, but may be unreliable**



2nd Study: Detecting Code Smells with LLMs

IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. XX, NO. XX, XX 2025

Beyond Strict Rules: Assessing the Effectiveness of Large Language Models for Code Smell Detection

Saymon Souza, Amanda Santana, Eduardo Figueiredo, Igor Muzetti, João Eduardo Montandon, and Lionel Briand

Saymon Souza (PhD)

***Abstract*—Code smells are symptoms of potential code quality problems that may affect software maintainability, thus increasing development costs and impacting software reliability. Large language models (LLMs) have shown remarkable capabilities to support different software engineering activities, but their use to detect code smells is still underexplored. However, unlike the rigid rules of static analysis tools, LLMs can support flexible and adaptable solutions tailored to different code smells. This paper evaluates the effectiveness of using four LLMs to detect nine code smells in 30 Java projects. For the empirical evaluation, we automatically and manually create ground truths; the latter relies on 76 developers inspecting 268 code smell candidates. Our results indicate that the agreement between LLMs and**

smells [21]. However, to the best of our knowledge, they do not provide strong empirical evidence to indicate, for instance, that LLMs perform better than traditional code smell detection tools.

Moreover, early investigations of LLMs for code smell detection typically employ controlled scenarios and small code samples [22]–[24]. Despite initial results, it is essential to understand the effectiveness of LLMs in detecting code smells in real-world software projects [18]. Such software projects introduce numerous challenges for LLMs, including the need to understand and navigate codebases, adhere to

Goal and Research Questions

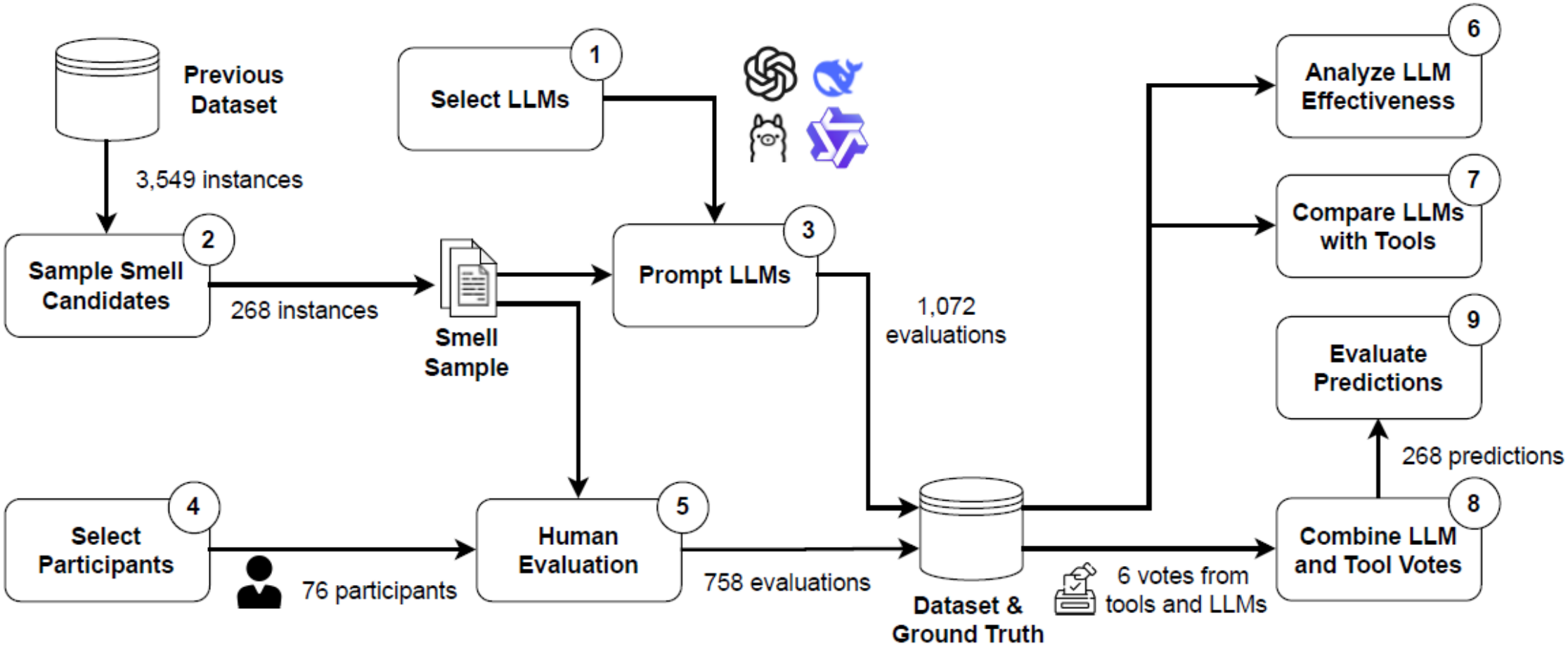
- Our goal is to investigate the effectiveness of **LLMs** in detecting **code smells** in Java projects

- Research Questions
  - **RQ1**: How effective are LLMs in detecting code smells?
  - **RQ2**: How do LLMs compare with static analysis tools?
  - **RQ3**: How effective is combining LLMs and Tools?

Used Dataset

- We analyzed 1,182 classes for code smells from 30 open-source Java projects
- Nine code smells
  - Data Class, Dispersed Coupling, Feature Envy, Intensive Coupling, Long Method, Large Class, Long Parameter List, Refused Bequest, and Shotgun Surgery
- Four LLMs and four static analysis tools
  - GPT-5 min, Llama-3.3, DeepSeek-R1, and Qwen2.5-Coder
  - JDeodorant, JSpIRIT, Organic, and PMD

Study Steps


Previous Dataset
3,549 instances
1
Select LLMs
2
Sample Smell Candidates
268 instances
Smell Sample
3
Prompt LLMs
1,072 evaluations
4
Select Participants
76 participants
5
Human Evaluation
758 evaluations
Dataset & Ground Truth
6 votes from tools and LLMs
6
Analyze LLM Effectiveness
7
Compare LLMs with Tools
8
Combine LLM and Tool Votes
268 predictions
9
Evaluate Predictions

Summary of Results

Smell	Large Language Models				Static Analysis Tools				Combined Predictions
	DeepSeek	GPT	Llama	Qwen	JDeodorant	JSpIRIT	Organic	PMD	
Data Class	😐	😐	🏅😊	😐			😐	😐	🏅😊
Dispersed Coupling	☹️	☹️	😐	😐		😐	😐		🏅😊
Feature Envy	☹️	☹️	🏅😐	☹️	😐		😐		😐
Intensive Coupling	🏅😊	😐	😐	😐		😐	😐		😐
Large Class	😊	😐	🏅😊	😊	😐	😐			🏅😊
Long Method	😐	🏅😊	😊	😐	😊		😊		🏅😊
Long Parameter List	😐	😐	😐	☹️			🏅😐	☹️	😐
Refused Bequest	☹️	☹️	☹️	☹️		🏅☹️	☹️		☹️
Shotgun Surgery	😐	😐	😐	☹️		🏅😐	😐		🏅😐

Legend: 😊 (F1 $\geq$ 0.80) 😐 (0.51 $\leq$ F1 $<$ 0.80) ☹️ (F1 $\leq$ 0.50) 🏅 Best strategies

Some Findings

- LLMs are **effective for "easily detectable" code smells**, such as Data Class, Large Class, and Long Method

- **Static analysis tools is better** than LLMs in some more complex smells, such as Refused Bequest and Shotgun Surgery

- Combining LLMs and static analysis tools has shown **promising results**

3rd Study: AI Mistakes in Coding

Evaluating the Effectiveness of LLMs in Fixing Maintainability Issues in Real-World Projects

Henrique Nunes
Federal University of Minas Gerais
Belo Horizonte, Brazil
henrique.mg.bh@gmail.com

Eduardo Figueiredo
Federal University of Minas Gerais
Belo Horizonte, Brazil
figueiredo@dcc.ufmg.br

Larissa Rocha
State University of Bahia
Alagoinhas, Brazil
larissabastos@uneb.br

Sarah Nadi
New York University Abu Dhabi
Abu Dhabi, United Arab Emirates
sarah.nadi@nyu.edu

Fischer Ferreira
Federal University of Itajubá
Itajubá, Brazil
fischer.ferreira@unifei.edu.br

Geanderson Esteves
Federal University of Minas Gerais
Belo Horizonte, Brazil
geandersonesteves@gmail.com

Henrique Nunes (PhD)

***Abstract*—Large Language Models (LLMs) have gained attention for addressing coding problems, but their effectiveness in fixing code maintainability remains unclear. This study evaluates LLMs capability to resolve 127 maintainability issues from 10 GitHub repositories. We use zero-shot prompting for Copilot**

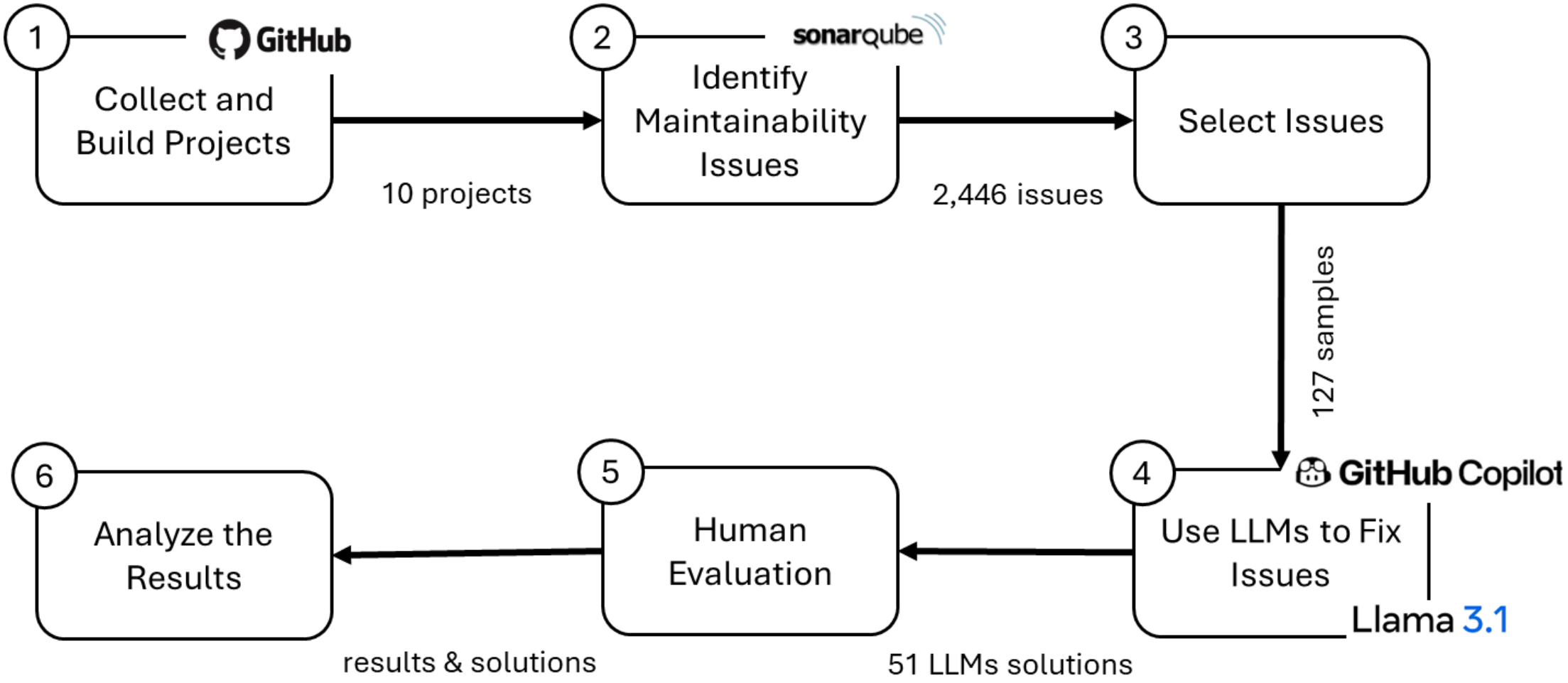
but using LLMs to refactor maintainability issues is underexplored and lacks relevant data [12]. In contrast to traditional automated tools that adhere to rigid rules, LLMs can provide an innovative approach to addressing maintainability issues.

Goal and Research Questions

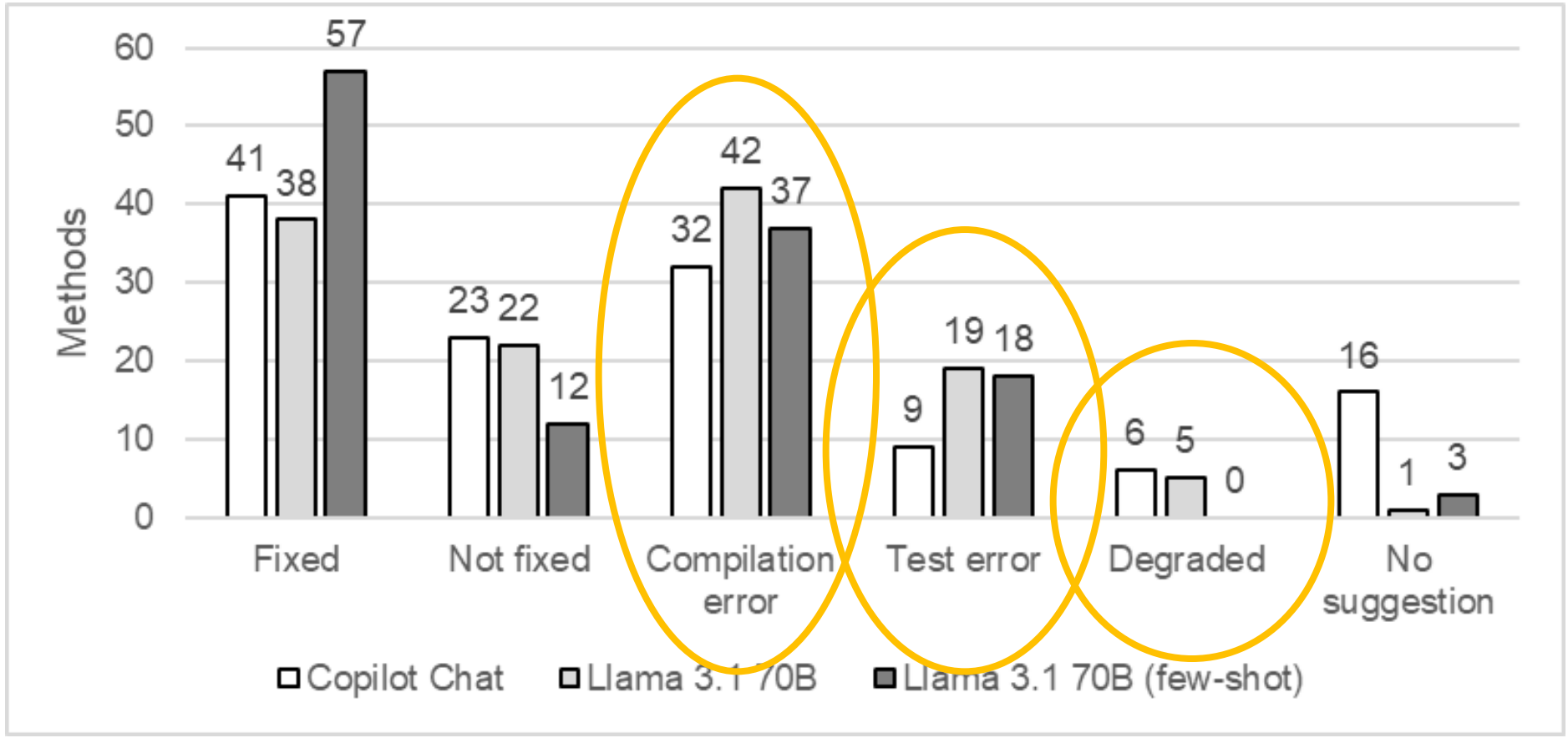
- Goal: Understand the strengths and drawbacks of LLMs in fixing code issues
  - Two LLMs (Copilot and Llama) refactored 127 code issues in 10 open-source Java projects

- Research questions
  1. To what extent can LLMs **fix** maintainability issues?
  2. What are the main **mistakes** made by LLMs when fixing code issues?
  3. To what extent do developers find LLM solutions more **readable**?

Study Steps


1
GitHub
Collect and Build Projects
10 projects
2
sonarqube
Identify Maintainability Issues
2,446 issues
3
Select Issues
127 samples
4
GitHub Copilot
Use LLMs to Fix Issues
Llama 3.1
51 LLMs solutions
5
Human Evaluation
results & solutions
6
Analyze the Results

Some Results about the AI Mistakes



All LLMs make mistakes, such as compilation errors and test failures

Follow-up: Train LLMs for Better Refactoring

- MaRV: A **Ma**nually **V**alidated **R**efactoring Dataset
  - Goal: train LLMs (e.g., few-shot prompts) to better identify and perform refactoring
  - With a survey, 40 participants evaluated 693 refactorings
  - Each refactoring instance was evaluated by 2 participants

- Results: Both participants agreed that 321 cases (~46%) are good examples of refactoring

Train LLMs with High Quality Data

Henrique Nunes (PhD)

MaRV: A Manually Validated Refactoring Dataset

Henrique Nunes
Federal University of Minas Gerais
Belo Horizonte, Brazil
henrique.mg.bh@gmail.com

Tushar Sharma
Dalhousie University
Halifax, Canada
tushar@dal.ca

Eduardo Figueiredo
Federal University of Minas Gerais
Belo Horizonte, Brazil
figueiredo@dcc.ufmg.br

***Abstract*—Despite the existence of traditional refactoring tools that offer semi-automated assistance, machine learning-based models have shown significant potential to generate refactored code. A comprehensive, manually validated refactoring dataset could help the software engineering community to train such models for effective refactorings. However, the community lacks a manually validated refactoring dataset. This paper introduces the *MaRV* dataset containing 693 manually evaluated code pairs extracted out of 126 GitHub Java repositories, representing four types of refactoring. In addition, the metadata describing the supposedly refactored elements was collected. Each code pair was manually evaluated by two reviewers out of 40 participants. *MaRV* dataset is constantly evolving with a web-based**

create refactoring datasets without evaluating the quality of the tool output [18, 19].

In this study, we create and propose Manually Validated Refactoring (*MaRV*) dataset, containing code snippets before and after a change, along with metadata such as manually annotated refactoring techniques, affected code elements, and commit details. To prepare this dataset, we use RefactoringMiner to identify applied refactorings from 126 popular Java repositories from GitHub. We focus on four refactoring techniques (*i.e., rename method*, *rename variable*, *extract method*, and *rename parameter*) to strike a balance

Take Aways

- Humans and AI models collaborate in nowadays software development teams
  - We need to **evaluate** and **define** their roles
- Humans are responsible for the successes and failures
  - Experienced developers are **more careful** in verifying the outputs of LLMs
- Current AI models mostly succeed in **easy coding tasks**
  - They make **a lot of mistakes** in real development tasks

UFMG

lab-soft
software engineering laboratory

Thank You!

From Requirements to Code: Assessing the Impact of Foundation Models on Software Development

Eduardo Figueiredo

Federal University of Minas Gerais (UFMG)

05 March 2026