LLMs for Software Engineering
State of the Art & Open Problems

Aug 26, 2025

Hou et al. (2024) & Fan et al. (2023)

Family


adidas

Occupation:
Faculty in Software Engineering

Change is the heart of software development

Programming is program transformation

Q1: Analyze what software changes occur in practice?

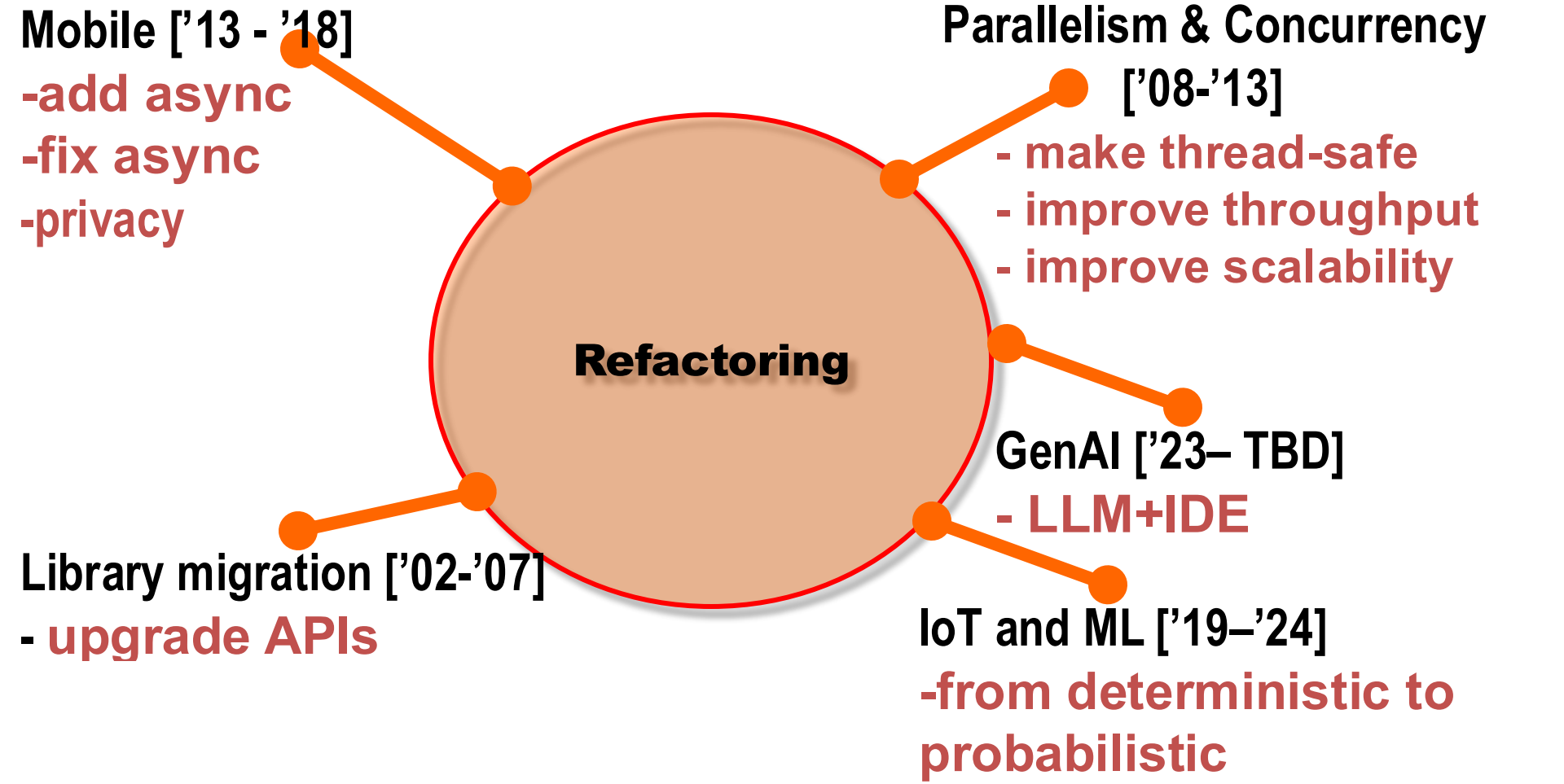
Q2: How can we automate them?

Q3: Can we represent programs as transformations? Archive, retrieve, and visualize them?

Q4: Can we infer higher-level transformations?

Visual Studio NetBeans IDE eclipse Android Studio

Work in Your Strength Zone but Reinvent Yourself



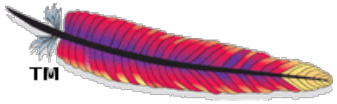
Principles for changing between different programming models

My dream: Practical Impact on SW Development

Automating

-ship with official

- hundreds of accepted patches



- first open-source refactoring

Refactoring

Inferring

- used at 

- dozen labs

founded Workshop on Refactoring Tools, HotSwUp, Dagstuhl S.

Understanding

- shaped APIs in Java and .NET official concurency libraries

-learnparallelism.net
150,000+ visitors

Testing

Recreation


TRIPLE ZERO

On Aug 5, 2015 …

From personal success to significance

From a ladder climber to a ladder holder

Motivation

Dominicana se transforma

Discussion Roadmap

1. Surprise Insights — 5–7 min

2. Models, Data, Gaps — 10–12 min

3. Optimization & Hybridization — 10 min

4. Evaluation & Non-determinism — 10–12 min

5. Tasks & Future Directions — 15 min

6. Hallucinations: reducing techniques

7. Breakout room activity

1. First Reactions & Surprise Insights (5–7 min)

What was the most surprising finding from these two surveys?

Did any result contradict your expectations?

Which SE task/domain did you expect to be more mature?

2. Models, Data, and Practice Gaps (10–12 min)

- Why are decoder-only LLMs dominating SE tasks? What do we lose by underusing encoder(-decoder) models?
- Only 6/395 studies used industrial datasets. How does this bias research?
- Should SE research prioritize lighter SE-specific LLMs over mega-models?

- YourQ: How do the fundamental architectural differences between encoder, decoder, and encoder-decoder models dictate their suitability for distinct SE tasks like code generation versus understanding?

Model Architectures for SE Tasks

🔷 Encoder-only (BERT, RoBERTa)
- Strength: Understanding, classification, embeddings
- SE Use: Bug detection, code search, clone detection, traceability

🔷 Encoder–Decoder (T5, CodeT5)
- Strength: Sequence-to-sequence transformations
- SE Use: Code translation, doc generation, refactoring, test generation

🔷 Decoder-only (GPT, CodeX, Code Llama)
- Strength: Generative, autoregressive, few/zero-shot
- SE Use: Code completion, repair, requirement-to-code, design brainstorming

✅ **Rule of Thumb:**
  - **Understanding → Encoder**
  - **Transformation → Encoder–Decoder**
  - **Generation → Decoder**