CS846 Week 2 Summary Review

Asim Waheed

Towards AI-Native Software Engineering (SE 3.0): A Vision and a Challenge Roadmap

Problem

The presentation questions whether the current software development pipeline is broken. In SE 2.0, AI functions mainly as an assistant through advanced code completion. Despite its usefulness, the pipeline remains inefficient: humans bear high cognitive load, training is inefficient, and code quality is suboptimal.

New Idea

SE 3.0 envisions AI-native software engineering built around an *intent-first*, conversation-oriented approach. Humans focus on thinking, while AI systems handle implementation. The roadmap introduces:

- **Teammate.next:** A personalized, self-evolving mentor with a form of Theory of Mind that adapts to human thinking patterns and ideally makes its reasoning understandable to humans.
- **IDE.next:** A conversation-based UI where code is hidden by default. Conversations are treated as first-class assets, complete with version control.
- **Compiler.next:** Translates conversations into code, balancing objectives such as accuracy, latency, cost, security, readability, and scalability.
- **Runtime.next:** An AI-specific runtime (*FMWare*) where models and software coexist. Resources are usage-managed, fine-tuning happens opportunistically, and edge-computing routes simple requests to smaller models.
- **FM.next:** Curriculum engineering for foundation models—curating high-quality, domain-specific knowledge and externalizing it from the model.

Positive Points

- Reimagines what it means to be a software developer: humans focus on ideas and value creation rather than low-level technical details.
- Rethinks the entire SE pipeline from the ground up, with detailed discussion of each component.
- Pushes toward democratization of software creation, enabling broader participation in development.

Negative Points

- **Obfuscation of code:** With code hidden, questions arise about the future of manual reviews, vulnerability discovery, and skill retention.
- **Democratization vs. resources:** Although more people could build software, SE 3.0 may be resource-intensive, increasing the gap between high- and low-resource developers.
- **Monopolization:** The best foundation models risk being monopolized, reinforcing trends already present today.
- **Feedback loops:** If AI trains on AI-generated code, will all software converge toward the same form? The presentation raises this but offers no solutions.

Future Work

(1) Incorporate explicit security reviews: how would tradeoffs between security and utility play out in SE 3.0, and would AI clarify or obscure them? *(2)* Survey research effort requirements across the roadmap: components like Teammate.next may be near-term, while Runtime.next requires more fundamental advances.

Rating

3/5 — an average contribution. The vision is ambitious and thought-provoking, but some aspects are underspecified and speculative.

While my rating was lower, the class in general had a more positive view of the paper.

Discussion Points

How would SE curricula adapt to SE 3.0?

We had an interesting discussion here. Jacie brought up how even right now the CS curriculum does teach us the very basics even if we do not use them in our day-to-day life. For example, most CS majors would take an OS course, or security course where they might be doing things they would normally never do in real life.

Prof. Godfrey had an interesting point on how universities and professors are already discussing how to incorporate AI-based coding into the curriculum. It cannot be completely ignored. However, the focus has to shift towards problem solving skills rather than programming skills.

Most of the class seemed to agree that in the future the most important skill would be problem solving.

How can harmful feedback loops (AI on AI-generated code, prompts, or systems) be prevented?

I brought up the idea that with AI generating all of the code, it might lead to a homogeneity of code. This could prove harmful if security vulnerabilities are introduced in this homogeneous code.

Michael talked about how even now in AI research there are ways to reduce the chance of feedback loops by limiting how much a model can update itself.

In general, the class seemed to agree that the kind of code we will see in SE 3.0 might be quite different from the kind of code we see today. If code is never meant to be read by humans, it may, eventually, not be very readable at all.

Does SE 3.0 lead more toward democratization or monopolization of software creation?

We had an interesting debate on this. On the one hand, Jacie talked about how monopolization is a risk with any new technology. However, the focus of this technology to reduce the skill-barrier required to start developing apps. Reducing skill-barriers will lead to some level of democratization.

On the other hand, training and serving FMs is prohibitively expensive. Most of the FMs would likely be in the hands of a few large corporations.

Overall, we had an interesting discussion of what the future of software engineering potentially looks like. Everyone agreed that the focus will be more on problem solving skills instead of strong technical skills.