CS846 Week 9 Summary Reviews: Code Review Comprehension: Reviewing Strategies Seen Through Code Comprehension Theories

Youssef Souati

1 Problem Being Solved

Teams rely on code review, yet we still lack a clear account of how reviewers actually read and understand real pull requests. Prior work often imagines small, linear changes, while real reviews are messy and cross–cutting. The paper asks how reviewers scope their understanding, which strategies they use, and how information sources and prior knowledge shape the mental model they build.

2 The Authors Proposed Idea

The paper proposes the Code Review Comprehension Model, CRCM. It extends classic code comprehension to pull requests and treats review as a loop of learning. Reviewers gather signals, combine them with what they already know, form a working mental model, compare what they see with what they expected and with an internal ideal, then decide and leave feedback. The loop repeats as they navigate and test.

CRCM has four parts that work together. The review process moves from context to inspection to decision, and the inspection step relies on chunking to keep the work manageable. Reviewers slice the change into chunks that fit the situation: by commits when history is clean, by files when the change is wide, or by starting from tests to learn intent before diving into implementation. Alongside the process, reviewers pull from many information sources such as the PR text, linked issues, discussion, CI, the codebase, tests, docs, and local runs. They also lean on a personal knowledge base that includes language fluency, common patterns, domain facts, and team conventions. With those inputs, they build a layered mental model that covers the goal the change aims to satisfy, how it is implemented, and the annotations that tie intent to code through names, commits, tests, and traces.

A key insight in CRCM is that reviewers use two comparators while they read. One is what they expected to find based on the context and any prior rounds. The other is an internal ideal shaped by experience and good practice. The tension between these explains why a change can meet expectations and still attract comments that push it toward a cleaner design.

3 Positive points

The paper gets reviewers to say the quiet part out loud. It shows how what they read in the code connects directly to the expectations they carry into a pull request. That framing makes the feedback feel less mysterious and more traceable to a reviewer's mental model.

It also backs claims with concrete evidence. Figure 4 lays out real, session-by-session mixes of review activities and orders them by change size, so you can actually see how work patterns shift as the diff grows.

The authors offer specific tool ideas rather than vague promises. They suggest visually separating meaningful chunks, surfacing the core of a change, integrating navigation and testing into the flow, and letting reviewers toggle comments to reduce bias. It reads like a to-do list that a platform could ship, not just a future-work shrug.

4 Negative points

The sample is all experts. That limits what we can say about novice or median reviewers, who may scope differently and lean harder on tests or automation.

The social layer is thin. Status, norms, and prior threads shape tone and outcomes, yet interactions are not analyzed in depth here. That leaves open how team culture intersects with the model.

The method has limits. Reviews were observed in a recorded setting, which can trigger the Hawthorne effect and nudge people to behave differently.

5 Future work

Bias–aware first pass. Compare a normal review where notes and prior comments are visible from the start to a blind–first pass where a reviewer records an initial verdict and notes, then reveals the discussion. Measure time, true defects, and reversals. If blind–first improves signal with a small cost, ship it as a default first step. This can help developers reduce anchoring.

Novice uplift with CRCM prompts. Give the same PRs to novices and experts, instrument navigation, and insert prompts like "start with tests," "chunk by commit," and "end with a skim." Track scope choices, time, and defects. If gaps close, bake the prompts into onboarding and the review UI.

6 Rating

4 out of 5. It reframes review as intentional comprehension with practical levers for tools and process, while leaving room to strengthen evidence and broaden beyond experts.

7 Discussion Points

Scope versus risk debt: When is a shallow or focused review acceptable, and who owns the risk if multiple reviewers scope down on the same PR? The class agreed that shallow or focused reviews can be acceptable, especially when you are onboarding less-experienced reviewers. The key is clear communication about depth. Platforms should let people label the kind of pass they are doing, for example "shallow review" or "focused on security aspects," so expectations and risk assignment are explicit. If several shallow passes land on the same PR, the gaps are visible instead of everyone assuming full coverage. Risk ownership should be distributed but transparent. One student put it well: "I wouldn't say blame the reviewer, but who owns the risk, I think that's easier to kind of point down." As a process fix, then I pointed to Google-style file ownership. When reviewers already know the area, context building is faster and time pressure is less likely to force shallow passes, and also easier to point down who owns the risk.

"Expected" vs "Ideal" tension: Does privileging "meets expectations" entrench sub-optimal designs and team folklore over better solutions? Some argued we should erase the gap by aligning expectations with a clear team ideal so the tension disappears. Others said context matters. If the goal is speed or a short-term release, meeting expectations can be the right call; for long-lived systems, the ideal should carry more weight. A concrete example was offered: if something exists to make quick money or ship quickly, an optimal design might not be needed. In the long run, strong team alignment shrinks the gap and reduces friction in review.

Commit hygiene as gate: Should tools enforce atomic commits? The room favored influence over enforcement. People noted that it is mentally taxing to review a tangled change and that developer capacity is limited, so tools should nudge toward smaller, coherent commits rather than block merges. Flexibility is important because some changes simply do not slice cleanly. The instructor added the historical note about time-boxed formal inspections and reminded us that modern review is also social and instructional, so tools should handle the easy stuff and leave space for human judgment.

Would the ownership model we saw in the Google case improve comprehension efficiency? Assigning reviewers by file or component ownership improves quality and speed because owners carry context and can spend energy on evaluation rather than orientation. That reduces cognitive load and ties directly to the paper's point about status and energy shaping outcomes. Even with ownership, reviewers should say when they lack capacity for a deep pass so others do not assume full coverage.