Code Review Comprehension: Reviewing Strategies Seen Through Code Comprehension Theories

11/12/25

Presented by

Youssef Souati

UNIVERSITY OF WATERLOO

What problem is being solved?

We rely on code review, but we still do not understand the actual mental process reviewers use.

Prior work often assumes linear reading and small changes, which misses the messy reality of real pull requests.

- RQ1. How do reviewers scope comprehension during code review?
- RQ2. What process and strategies do reviewers use while reviewing?
- RQ3. What are the roles of information sources, the knowledge base, and mental models in code review?

Study design

- 25 review sessions with 10 experienced reviewers, observed on real PRs.
- Think-aloud during the review, short interviews after, transcripts analyzed.
- Reviews covered GitHub, GitLab, Gerrit, and Phabricator.

The new idea: CRCM

Code Review Comprehension Model (CRCM)

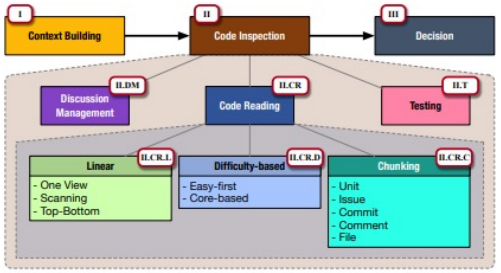
- Extends Letovsky's code comprehension model to PR review and integrates constructivist learning (Piaget) and self-discrepancy ideas.
- Treats review as an opportunistic learning loop: construct a mental model from information sources and knowledge; compare actual vs. expected and ideal; evaluate and give feedback; update knowledge and enrich sources.

Theoretical foundations (what “classic theory” means)

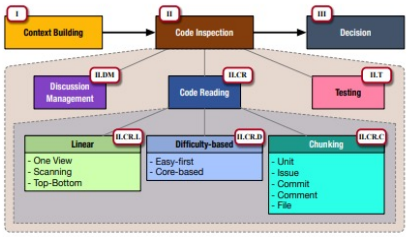
- The models behind CRCM
  - Letovsky’s model: code comprehension = knowledge base + information sources → mental model.
  - Constructivism (Piaget): understanding is incremental and scoped; capacity limits drive chunking and staged learning.
  - Self-discrepancy: reviewers compare the actual PR to expected and ideal mental models, which shapes feedback and verdicts.

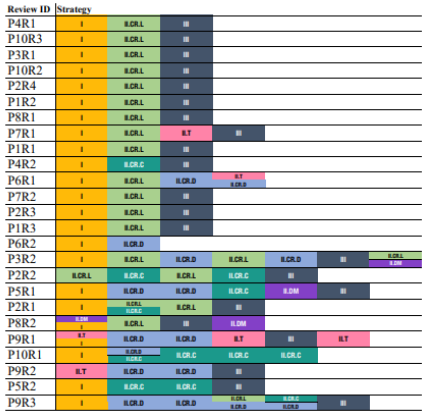
CRCM components

1. Code review process: context → inspection → decision.
2. Information sources: PR text, issues, discussion, CI, codebase tests, docs, local runs.
3. Knowledge base: language, patterns, domain, team conventions.
4. Mental model: specification, implementation, annotation.



CRCM components





Results (RQ1): Scoping comprehension

Four depths reviewers choose

- Full: seek complete understanding now.
- Focused: go deep on specific risks or areas.
- Partial: review selected sections for now.
- Shallow: a surface pass, leaning on tests and other reviewers.

Results (RQ2): Process & strategies

How reviewers move through a PR

- Chunk by commit when history is clean.
- Chunk by file or functional area when history is noisy.
- Start with tests to learn intent or finish with tests to validate behavior.
- End with a quick linear skim to catch inconsistencies.

Authors found that most sessions ended with comments or request changes, fewer with direct approval.

Results (RQ3): Information sources & testing

What reviewers consult

- PR title and description, linked issues, prior rounds, discussion threads.
- Codebase, tests, documentation, CI results, and local experiments.

Results (RQ3): Mental model layers

Three layers of the PR mental model

- Specification: goal, why, scope of addressed cases.
- Implementation: what changed, how it behaves, and where it lives.
- Annotation: links that tie goal to code through names, commits, tests, and traces.

Results (RQ3): Expected vs. Ideal

Two comparators reviewers use

- Expected: what they thought they would see based on context and prior rounds.
- Ideal: an optimal solution from experience and best practice.

Positives

- The paper captures reviewers saying the quiet part out loud. (code -> expectations of the reviewer)
- Figure four shows real session-by-session compositions of activities and ordered by change size.
- They propose very specific affordances: visually separate meaningful chunks, surface the “core” of the change, integrate navigation/testing with review, and let reviewers *toggle* comments to reduce bias. Not just “future work” shrug ¯_(ツ)_/¯.

Negatives

- Sample skew: Only experts were observed. Generalizability to novice or median reviewers is limited.
- Social layer missing: Norms, status, and prior threads shape reviews, but interactions were not analyzed.
- Method constraints: Video setting and Hawthorne effect noted. Little probing of comment rationales, so causal links are thin.

Future work

- 1- Bias-aware review flow
  - What: Compare pre-seeded context (diff + notes/comments visible) vs blind-first then reveal.
  - Learn: Does early context trade accuracy for speed? Does a blind-first pass reduce anchoring without hurting throughput?
  - Outcome: If blind-first improves true defect finds with only a small time cost, make it the default first pass.
- 2- Novice vs expert reviewers
  - What: Same PR set for both groups; measure scope choices, navigation strategy, time to verdict, and defects found; test CRCM prompts/guidance.
  - Learn: Where novices differ from experts and whether CRCM cues close gaps in quality and time.
  - Outcome: If gaps shrink meaningfully, ship CRCM prompts in onboarding and the review UI; otherwise, target training to the weak spots.

Rating

I would give this paper a 4 out of 5. Code review is very important and this paper upgrades how we talk about review comprehension in a way that tool builders and teams can actually use.

Discussion points

- Scope vs risk debt: When is a shallow or focused review acceptable, and who owns the risk if multiple reviewers scope down on the same PR?
- "Expected" vs "Ideal" tension: Does privileging "meets expectations" entrench sub-optimal designs and team folklore over better solutions?
- Commit hygiene as gate: Should tools *enforce* atomic commits?
- Would the ownership model we saw in the google case study improve comprehension efficiency?

UNIVERSITY OF WATERLOO