Code Review

CS 691 / SWE 699
Fall 2025

GEORGE MASON UNIVERSITY

Logistics

- Reflection 4, Project Checkpoint, Lecture 10 reading questions due today
- Lecture 11 reading questions due today next week at 4:30pm
- Lecture 10 activity (in class today), due by 11/7 at 4:30pm
- Project presentations in 4 weeks

Today

- Discussion: Experiences from Lecture 9
- Discussion: Reading questions for Lecture 10
- Lecture
  - Code Review
- In-Class Activity

Discussion: Experiences from Lecture 9 Activity

- How did you use LLM to help understand a codebase?
- How did you build trust in the answers?
- What was it good or bad it?
- How did experience of using LLM compare to not using an LLM?

Discussion: Reading questions for Lecture 10

- What questions did you have from readings for Lecture 10
  - Discuss questions & possible answers in group of 3 or 4
  - Come back with 1 question you want to discuss w/ whole class

Code Review

Modern Code Review: Goals

- Ensure that code can be read by others
- Ensure consistency of style and design
- Ensure adequate tests
- Accident prevention: find defects & other quality issues
- Education: ensure that multiple developers are familiar with code to be able to maintain
- Records code history, enabling future auditing of changes when understanding how and why defects introduced

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. ICSE-SEIP.

Don't Crash When Reading Bad Defines #129

Merged huderlem merged 3 commits into `huderlem:master` from `garakmon:testing` on May 9, 2019

Conversation 11 | Commits 3 | Checks 0 | Files changed 6

garakmon commented on May 5, 2019 • edited — Collaborator

but warn the user (give a file name and line number) about bad expressions.

Reading tilesets now doesn't give warnings about the secret base tileset pointers.

Added colorful log output for Linux and macOS users (and probably this applies to WSL). Windows I can't reliably test and it would require a refactor that is probably not worth it.

A

garakmon added 2 commits on May 5, 2019

- improve tileset label reading to silence unnecessary warnings — fc407b6
- readCDefines() - don't crash on invalid expressions, add better debug… — 6303068

B

huderlem requested changes on May 5, 2019 — View changes

src/project.cpp

```
1628  -        *definesToSet = definesInverse.values();
1629  -    } else {
1630  -        logError(QString("Failed to read C defines file: '%1'").arg(filepath));
      1597 + void Project::readCDefinesSorted(QString filename, QStringList prefixes, QStringLis
```

C

huderlem on May 5, 2019 — Owner

I feel like the `readCDefines` functions should live in `parseUtil.cpp`. Agree?

👍 1

D

garakmon on May 5, 2019 — Author — Collaborator

yes

E

- move source parsing functions from project to parseutil — 81c937a

F

readCDefines() - don't crash on invalid expressions, add better debug...

B

garakmon committed on May 5, 2019 — commit 6303068

272 src/project.cpp

```
1622      -        logError(QString("Failed to read C defines file: '%1'").arg(filepath));
      1597 + void Project::readCDefinesSorted(QString filename, QStringList prefixes, QStringList* definesToSet) {
      1598 +     QString filepath = root + "/" + filename;
      1599 +
      1600 +     QMap<QString, int> defines = readCDefines(filename, prefixes);
      1601 +
      1602 +     // The defines should to be sorted by their underlying value, not alphabetically.
      1603 +     // Reverse the map and read out the resulting keys in order.
      1604 +     QMultiMap<int, QString> definesInverse;
      1605 +     for (QString defineName : defines.keys()) {
      1606 +         definesInverse.insert(defines[defineName], defineName);
1623  1607      }
      1608 +     *definesToSet = definesInverse.values();
1624  1609  }
```

C

move source parsing functions from project to parseutil

F

garakmon committed on May 9, 2019 — commit 81c937a8

349 src/project.cpp

```
1597  - void Project::readCDefinesSorted(QString filename, QStringList prefixes, QStringList* definesToSet) {
1598  -     QString filepath = root + "/" + filename;
1599  -
1600  -     QMap<QString, int> defines = readCDefines(filename, prefixes);
1601  -
1602  -     // The defines should to be sorted by their underlying value, not alphabetically.
1603  -     // Reverse the map and read out the resulting keys in order.
1604  -     QMultiMap<int, QString> definesInverse;
1605  -     for (QString defineName : defines.keys()) {
1606  -         definesInverse.insert(defines[defineName], defineName);
1607  -     }
1608  -     *definesToSet = definesInverse.values();
```

243 src/core/parseutil.cpp

```
      294 + void ParseUtil::readCDefinesSorted(QString filename, QStringList prefixes, QStringList* definesToSet)
      295 +     QMap<QString, int> defines = readCDefines(filename, prefixes);
      296 +
      297 +     // The defines should to be sorted by their underlying value, not alphabetically.
      298 +     // Reverse the map and read out the resulting keys in order.
      299 +     QMultiMap<int, QString> definesInverse;
      300 +     for (QString defineName : defines.keys()) {
      301 +         definesInverse.insert(defines[defineName], defineName);
      302 +     }
      303 +     *definesToSet = definesInverse.values();
```

Practices

- Ownership
  - Reviewers should have ownership of relevant part of the code, ensuring consistency and awareness of priorities and practices
- Readability
  - Ensure knowledge in code style & best practices for language

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. ICSE-SEIP.

Process

- Creation: create a change request
- Preview: review diff & results from automated quality checks
- Commenting: reviewers comments on change, create action items to address by author
- Addressing feedback: author address comments
- Approval: reviewers mark change as good to go after action items have been addressed

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. ICSE-SEIP.

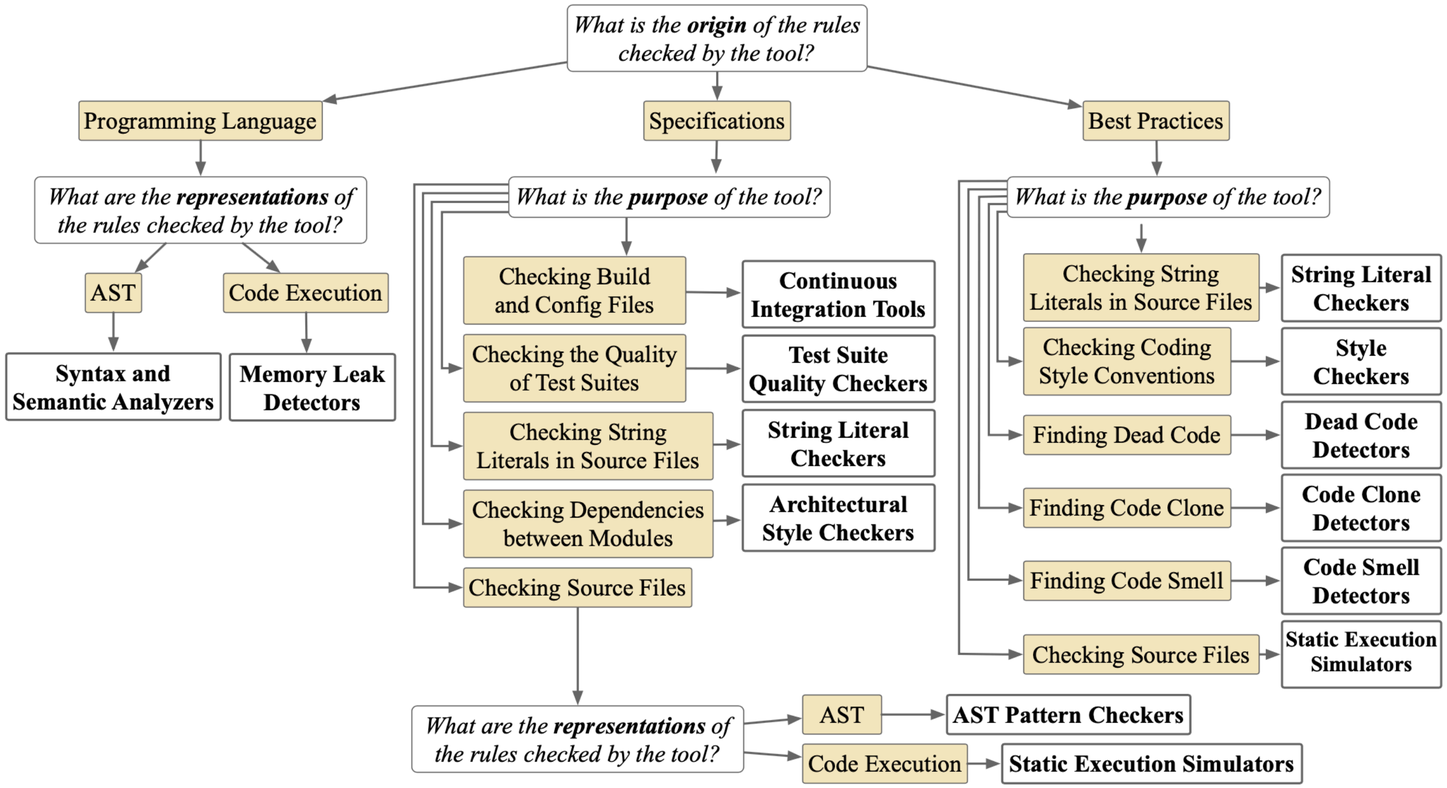
Challenges in Code Review

- Distance: geographical & organizational, lead to delays & misunderstandings
- Social interactions: tone & power dynamics can make developers uncomfortable
- Design: should design be reviewed before or during code review
- Context: need adequate understanding of motivation for change

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. ICSE-SEIP.

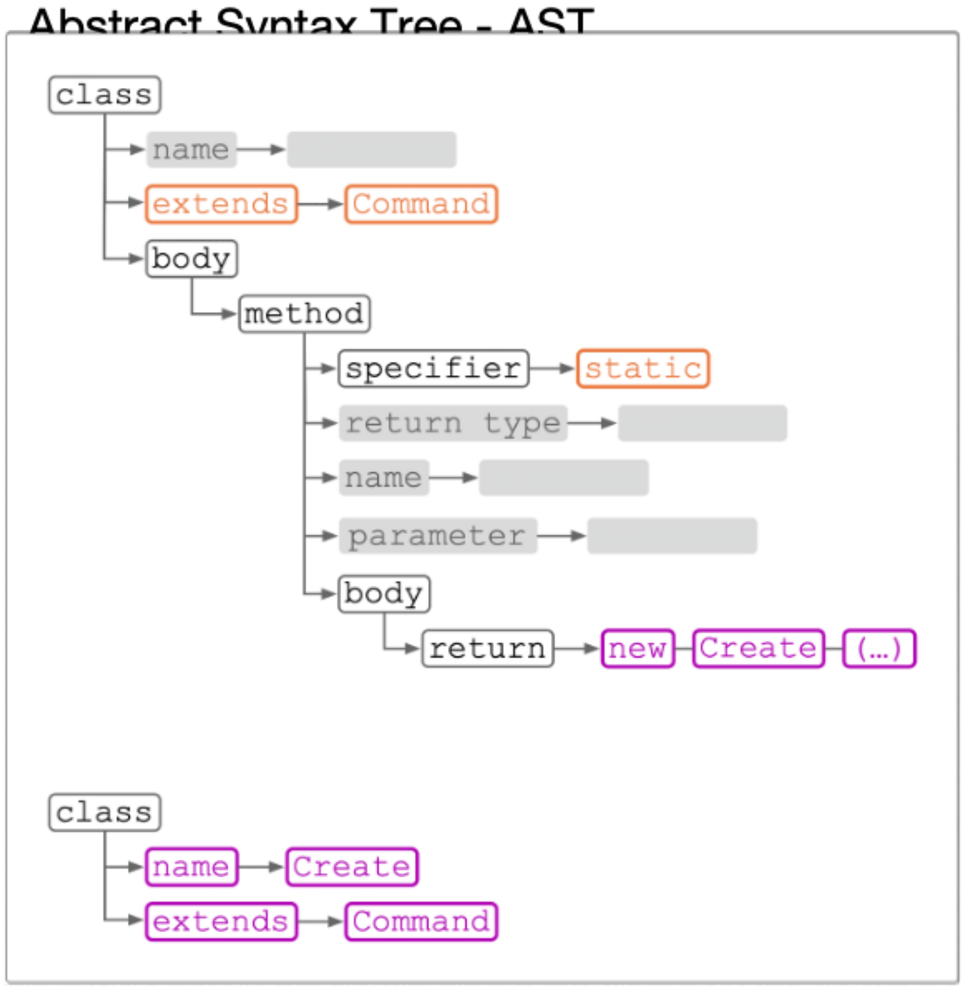
Automated Code Review Tools

- Rather than rely on human reviewers to find all issues, long been emphasis on using automated tooling
- Increased in popularity with rise of Continuous Integration / Continuous Deployment workflows
-



Sahar Mehrpour and Thomas D. LaToza. 2022. Can static analysis tools find more defects? A qualitative study of design rule violations found by code review. Empirical Softw. Engg.

Example: AST-based Rules



All microtask commands must be handled by command subclasses.

IF a method is a static method on Command
THEN it should implement its behavior by constructing a new Command subclass instance.

```
class ACommand extends Command {
   static ACommand create(...) {
      return new Create(...);
   }
}

class Create extends ACommand {

}
```

Sahar Mehrpour and Thomas D. LaToza. 2022. Can static analysis tools find more defects? A qualitative study of design rule violations found by code review. Empirical Softw. Engg.

Types of defects

- Maintainability: impact quality of code
- Implementation: code does not satisfy its requirements
- Build config: may cause build & integration to break, through missing scripts of config settings
- Test suite: incorrect & ineffective tests, such as incorrect assertions of insufficient code coverage
- User Interface: visual appearance of software as displayed to the user
- Requirements: missing or misinterpretation of project requirements
- Performance: memory or runtime issues, such as unnecessary computation or poorly optimized memory allocation (e.g., unnecessary database queries)

Sahar Mehrpour and Thomas D. LaToza. 2022. Can static analysis tools find more defects? A qualitative study of design rule violations found by code review. Empirical Softw. Engg.

Potential use of static analysis tools to find defects

SAT Type and Defect Type	**All Defects** Count	**All Defects** Overall %
All SATs	1009	76.27 %
AST Pattern Checkers	340	25.70 %
Maintainability	83	
Implementation	188	
Test Suite	8	
User Interface	33	
Requirement	20	
Performance	8	
Style Checkers	339	25.62 %
Maintainability	339	
Continuous Integration Tools	67	5.06 %
Build Config	67	
Static Execution Simulators	65	4.91 %
Maintainability	16	
Implementation	43	
Requirement	1	
Performance	5	
Architectural Style Checkers	60	4.54 %
Maintainability	57	
Test Suite	3	
Test Suite Checkers	29	2.19 %
Test Suite	29	
Code Clone Detectors	28	2.12 %
Maintainability	27	
Test Suite	1	
Dead Code Detectors	27	2.04 %
Maintainability	27	
Syntax and Semantic Analyzers	20	1.51 %
Implementation	20	
String Literal Checkers	20	1.51 %
Maintainability	10	
Implementation	10	
Code Smell Detectors	10	0.76 %
Maintainability	10	
Memory Leak Detectors	4	0.30 %
Implementation	4	

Sahar Mehrpour and Thomas D. LaToza. 2022. Can static analysis tools find more defects? A qualitative study of design rule violations found by code review. Empirical Softw. Engg.

Examples of defects

Example 1. "we will want to emit the `clicked` signal no matter whether the topic is sensitive or not."

If any instance of an object is created, then a specific method should be called regardless of the object properties.

Example 3. "please add a `target="_blank"` and a descriptive `title` tag to all the external links"

If an HTML tag contains an external link, then the tag should also include a specific property and a proper title.

Example 4. "Why are we keeping these [persisted data] in memory at all? We could just as well load them on demand when a new `channeld` gets started"

If persisted data is used, it should be loaded on demand rather than retained in-memory.

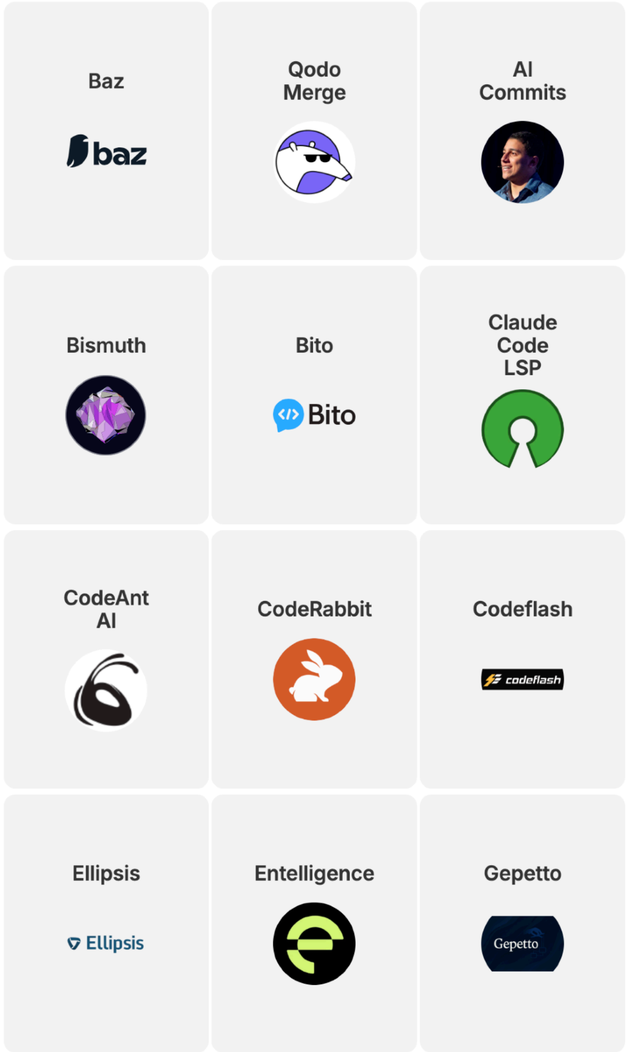
Example 5. "Do not use underscore in method naming. Read [document on] java code convention. Method name should be `getTaskStatus` [instead of `get_task_status`]"

If an identifier is for a method, then it should not include underscores.

Sahar Mehrpour and Thomas D. LaToza. 2022. Can static analysis tools find more defects? A qualitative study of design rule violations found by code review. Empirical Softw. Engg.

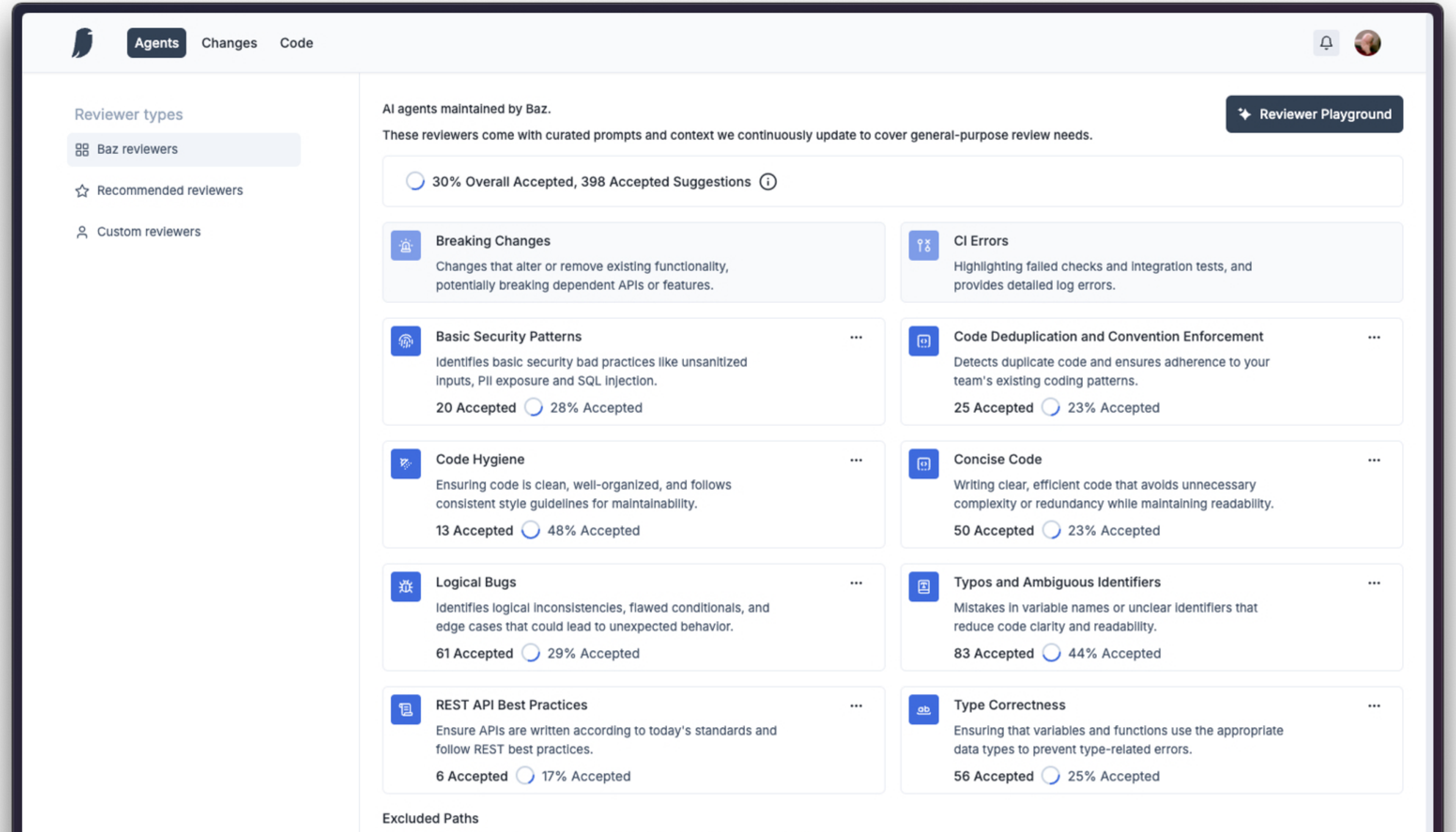
Automated Code Review Tooling with LLMs

- Interest in using LLMs to find defects
- Many ways to build it
  - Run traditional tools (e.g., linter, spell checker, defect detectors), use LLM to explain or propose fixes
  - Have LLM infer rules, which are then checked
  - Agentic workflows



https://ainativedev.io/landscape

Example: Agentic Code Review Workflow in Baz



Logical Bugs Agent in Baz

- Uses whole-repo retrieval: pulls call graphs, helper functions, and test fixtures via similarity search to reconstruct the logical context for a change.
- Combines AST-based program understanding with agentic reasoning to propose concrete execution scenarios that violate invariants (for example, inconsistent state transitions, missed early returns that lead to surprising side effects, or incorrect assumptions about data shapes).
- Correlates dynamic evidence (if available) such as existing test traces or recorded failing CI logs to strengthen hypotheses.
- Outputs Findings that include the reasoning trace, implicated files and lines, and suggested repro steps or minimal code locations to inspect.
- Constraints: this agent is compute-heavy and tuned to reduce false positives by requiring multi-source evidence (retrieval + AST +, where possible, runtime traces).

https://docs.baz.co/agents/baz-reviewers

Spec Review Agent

- Validate change meets issue requirements
- The agent is automatically initated when a pull request is linked to a Jira ticket and optionally a live preview environment. If the ticket contains references to a Figma design the agent will include it in its context.
- The agent builds a sandboxed browser session and delegates UI interactions and visual checks to sub-agents. The agent runs test cases, compares rendered UI to design artifacts, and verifies ticket requirements.
- Output: a single PR comment grouping unmet requirements first and met requirements collapsed, as well as a GitHub check with pass/fail/neutral.
- Constraints: requires a reachable or credentialed preview environment. Inline annotated screenshots are planned but not yet implemented. The agent is designed to reduce noise by summarizing issues in one place rather than producing many inline comments.

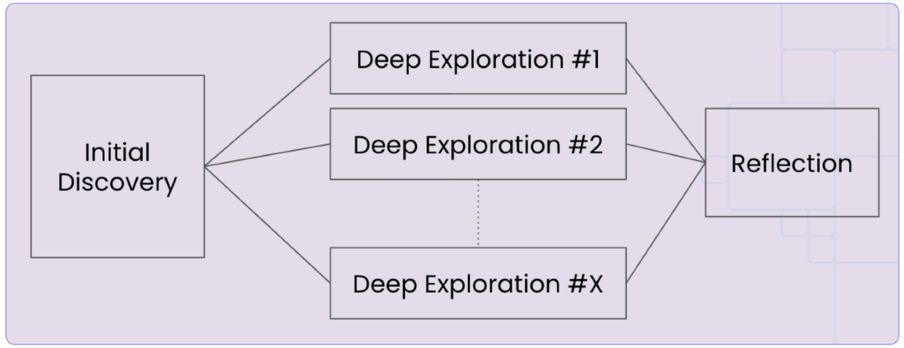
https://docs.baz.co/agents/baz-reviewers

Baz Agent Architecture

1. **Context Mapping**
   Locate where the PR introduces changes in the broader codebase. Identify touched systems, functions, contracts, and consumers.

2. **Intent Inference**
   Infer what the PR is trying to achieve from the title, description, commit messages, ticket - and the code itself. Add input from integrated systems including Jira, Linear and Youtrack, with additional data from Github and Gitlab.

3. **Socratic Questioning**
   Generate probing validation questions that challenge assumptions and common pitfalls:
   - "What happens if the list is empty?"
   - "This API schema changed. Are all consumers updated?"
   - "Are we persisting all required fields to the database?"



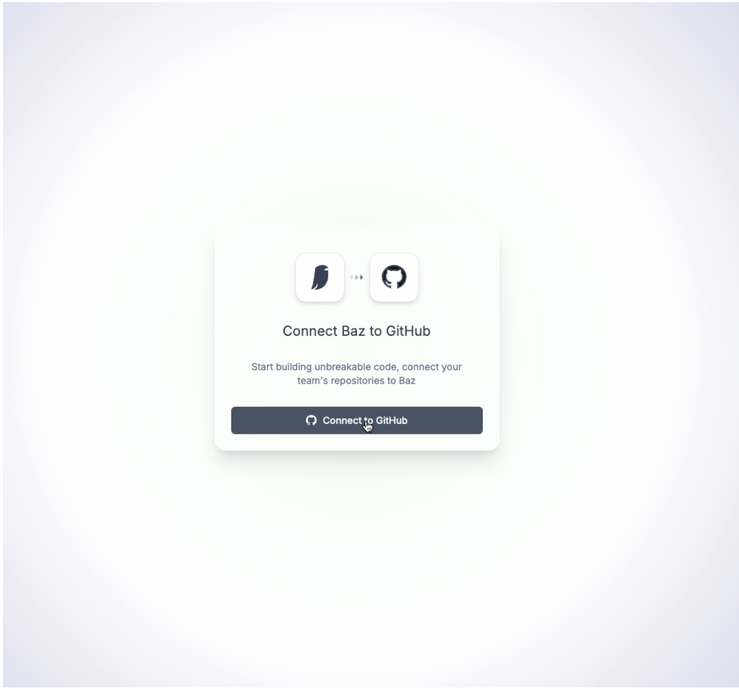
4. **Targeted Investigation (multi-agent)**
   Spawn independent sub-agents, each assigned to **prove or disprove one risk**. They traverse the repo, read diffs and files, run searches, and manage their own task queues.

5. **Reflection & Consolidation**
   Aggregate findings, filter false positives, and surface a concise report: evidence, rationale, and a **yes/no verdict** for each risk.

https://baz.co/resources/engineering-intuition-at-scale-the-architecture-of-agentic-code-review

Using Baz

- Integrated into GitHub / GitLab
- MCP Server



Key questions about Agentic Code Review

- What types of issues can it find
  - Limited by available information
    - e.g., may not have visibility into performance issues without understanding of perf implications of a design decision
- How well does it know your codebase
  - Like all agentic LLMs, still relies on having right context to surface relevant idioms & practices
  - If these are well documented, may work better
  - May try to infer from past changes or pull requests comments, but may be harder

10 min break

In-Class Activity

- In groups of 2, try out Baz on pull requests
    - Clone a repo --- can be one of yours (e.g., your city simulator) or an existing project (ideally one you are familiar with)
    - Prepare small code changes (e.g., a few lines of code)
    - Goal: build changes that have issues the tool can't find (e.g., violate coding style conventions for project, put code in the wrong place, have unhelpful identifiers, etc.)
        - Build pull requests for your changes, ask Baz to review, see what types of issues it is able to catch
- Deliverables
    - Screen recording through Kaltura
        - Upload to OneDrive, turn on link sharing, share link in Lecture 10 activity submission on Canvas
        - Submit answers to questions on your experiences on Canvas (next slide)
- Aim to finish by 7:10pm today; Due tomorrow at 4:30pm

Questions to answer

- What types of issues did you explore?
  - How did you design the issue, and why did you think it would be hard to find?
- Which of the issues was the LLM able to find in code review? Which was it not able to find?
- How helpful were the suggested fixes and explanations?
- What did you learn about using LLMs for code review?

- **Deliverable**: Submit through Canvas, at least a page