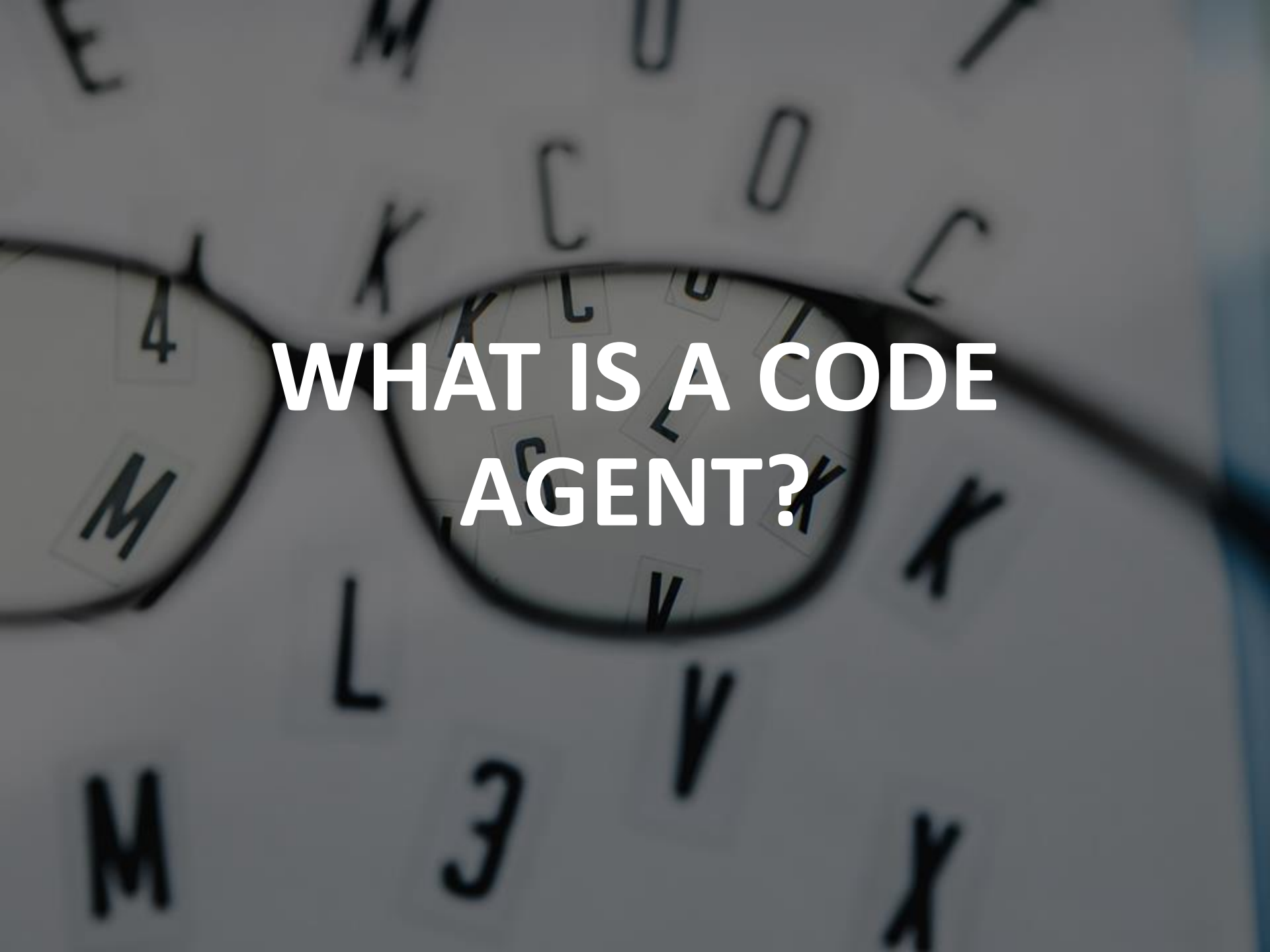

Code Agents — Code generation & debugging

Learning Objectives

1. Define code agents and contrast with completion-only tools
2. Diagram architecture & trust boundaries (ACI, tools, memory, harness)
3. Design retrieval/context plans; budget tokens effectively
4. Execute a closed-loop debugging cycle; capture artifacts
5. Interpret benchmark results; pick meaningful metrics


WHAT IS A CODE AGENT?

Definition & Scope

Autonomous system that plans, edits, runs, and evaluates code changes

Inputs: issue/ticket, repo files, docs;
Outputs: diffs/PRs & test results

Environment: editor + shell + test harness + **V**ersion **C**ontrol **S**ystem/**C**ontinuous **I**ntegration via **A**gent **C**omputer **I**nterface

Not just code completion; ****closed loop**** with execution feedback

Why “closed loop” is essential

Closed loop: the agent doesn’t just generate code and stop, it

- **runs** the code/tests,
- **observes** results,
- and **edits again** until acceptance criteria are met

Benefits of “closed loop”

1. **Grounding in reality:** Execution results (pass/fail, stack traces) cut through hallucinations and guesswork.

2. **Objective success signal:** Continuous Integration/test outcomes define “done,” not the agent’s confidence.

3. **Smaller, safer diffs:** Iteration encourages minimal patches with clear effects.

4. **Better debugging:** Failures become *data* the agent can use to localize bugs.

5. **Reproducibility & auditability:** Each cycle leaves artifacts (diffs, logs, CI runs).

6. **Safety hooks:** The loop runs inside a sandbox with gates (coverage, SAST, secret scans), so risky edits get blocked early.

Benefits of ‘closed loop

Grounding in reality

Smaller, safer diffs

Safety hooks

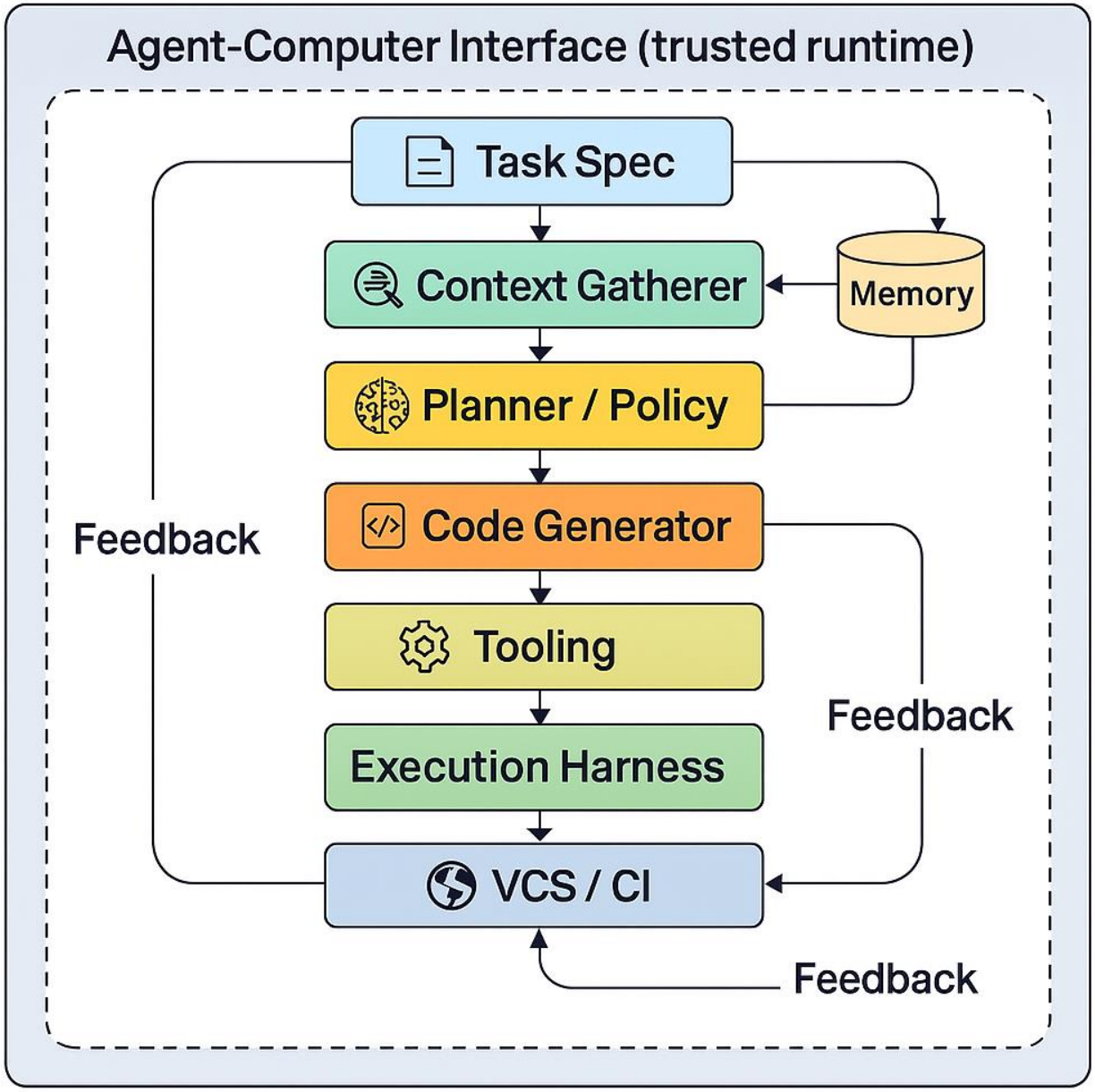
Better debugging

Reproducibility & auditability

Reproducibility & auditability

Mental model (pseudocode)

```
while not acceptance_criteria_met:
    plan = analyze(issue, context)
    patch = propose_minimal_diff(plan)
    apply(patch)
    results = run_harness()   # tests, lint, build, coverage
    if results.green:
        open_or_update_PR(patch, results)
        break
    else:
        diagnose(results)     # read failing tests/tracebacks
        refine_plan()
```


Agent-Computer Interface (trusted runtime)
Task Spec
Context Gatherer
Memory
Planner / Policy
Code Generator
Tooling
Execution Harness
VCS / CI
Feedback
Feedback
Feedback

Context Strategies for Large Repos

- Embed & retrieve: files, symbols, docs, issues (rank by task relevance)
- Static signals: call graph, imports, types, ownership metadata
- Budget: prioritize high-signal snippets; iterative retrieval
- Determinism: cache retrieval results for reproducible runs

Generation Patterns

- Spec-first: restate requirements; define acceptance tests
- Scaffold-first: create stubs/interfaces; fill post-tests
- Diff-first: **unified patches** to minimize churn
- Decompose: small, verifiable steps vs. monolithic changes

Debugging Loop (execution-guided)

- Run tests/build; capture errors, logs, coverage
- Localize fault (binary search, delta debugging)
- Hypothesize cause; **minimal fix**; regenerate
- Selective re-runs; stop criteria: tests pass + coverage ≥ baseline

Evaluation & Benchmarks

- SWE-bench & SWE-bench Verified: real issues/PRs in multi-file repos
- SWE-agent ACI improves agent performance on SWE-bench (reported 12.5% pass@1)
- DebugBench/DebugEval: bug localization/repair tasks
- Metrics: build-green rate, time-to-green, diff churn, coverage Δ, flake rate

In-Class Activity (10–15m)

Artifact	What to produce
Plan	Subgoals + acceptance tests
Retrieval set	Top files/symbols to inspect (with rationale)
Patch outline	1–3 edits with expected test impact
Risks	2 risks + quick mitigations

Reading — Lecture 1

- SWE-bench (arXiv 2310.06770); SWE-bench Verified (OpenAI, 2024/25)
- SWE-agent (arXiv 2405.15793; NeurIPS'24)
- OpenHands (arXiv 2407.16741; OpenReview)
- AutoCodeRover (arXiv 2404.05427)
- Self-Debug (ICLR'24); Reflexion (NeurIPS'23)
- DebugBench/DebugEval (2024–25)