Code review workflows

Diffs, test results, and human judgment

Access and cost preflight

Follow the course safety baseline.

1. Check the current GitHub docs and your organization's rules.
2. Check the repository, data, and review permissions.
3. Use a small pull request with clear acceptance criteria.
4. Set a usage limit for metered work.

Review is a human responsibility

Automation may raise questions. A reviewer decides whether a change is correct, safe, maintainable, and within scope.

```
Issue and acceptance criteria
        ↓
Proposed change and checks
        ↓
Human review
        ↓
Accept, request revision, or implement manually
```

Review rubric

- Correctness against acceptance criteria
- Tests and check results
- Security, dependency, and permission implications
- Scope and maintainability
- Repository conventions and human approval

Conflict-resolution exercise

1. Read both branch intentions.
2. Write the merged behavior and tests before editing.
3. Resolve the conflict manually in the repository sandbox.
4. Run checks and request review.
5. Demonstrate an approved assisted workflow only after checking current documentation and policy.

Required access

Check CLI `/review` and `/security-review`, Copilot PR review, Git, Node.js, and npm. Use a private training remote with the baseline branch already present.

Stop if a command or review access is missing. A worksheet does not replace the required runs.

Review the same diff through separate passes

1. Record the human verdict before assisted review.
2. Leave the patch uncommitted and run `/review`.
3. Run `/security-review` against that same local patch.
4. Triage both outputs, then commit and request Copilot PR review.
5. Repeat the local commands on the correction diff before committing fixes.

Keep the original verdict. Record no-findings results honestly.

A finding needs support

Claim	Check
Exposed password field	Inspect the actual response and add a regression test
Database compromise	The mock route never executes SQL; reject that claim
Leaked credential	The strings are fake and unused; do not claim a live leak
Pagination defect	Check page-one behavior separately from security review

Record: review source, file/line, impact, check, and triage decision.

Security review has a narrow scope

`/security-review` examines current local changes.
It is read-only and does not replace repository security scanning.

A clean result does not prove that a change is secure.
The human reviewer still decides.

Agenda and time plan

Time	Topic	Trainer move
0:00	Review workflow and roles	Define accountability
0:08	Summaries and descriptions	Check the draft against the diff
0:18	Local commands and PR review	Keep the patch uncommitted
0:28	Defect and triage demo	Check behavior and reject unsupported claims
0:43	Suggestions and conflicts	Preserve author intent
0:51	Limits and adoption	Design human gates
0:58	Lab handoff and Q&A	Assign reviewer roles

Transition: "Copilot can increase review coverage; it cannot own the decision."

Copilot across a PR lifecycle

```
Issue and acceptance criteria
          ↓
Branch and bounded commits
          ↓
Draft summary and PR description
          ↓
Automated checks + Copilot first pass
          ↓
Human review and author response
          ↓
Approval, revision, or close
```

At each stage, save the output the reviewer needs to check the change.

Commit messages and PR descriptions

Copilot can draft text from the visible change, but authors must check:

- the user-visible behavior is accurate;
- tests and results are not invented;
- breaking changes and migrations are named;
- security and dependency implications are named;
- issue links and acceptance criteria match the work;
- no sensitive details appear in generated prose.

A polished summary can still be wrong.

A reviewable PR template

```
## Problem
What user-visible problem does this solve?

## Change
List the bounded implementation choices.

## Validation
- [ ] Focused tests
- [ ] Existing checks
- [ ] Manual edge case

## Risk and rollback
Name data, dependency, security, and recovery considerations.
```

Requesting Copilot review

For a manual request:

1. Check that the PR is ready for a first pass.
2. Check current availability, permissions, and repository policy.
3. Request the Copilot review through the supported PR control.
4. Wait for comments and review them as hypotheses.
5. Resolve them, reject them with a reason, or implement a checked fix.
6. Request human review with the diff, findings, and check results.

Do not use Copilot review to bypass required reviewers.

Automatic review configuration

Automatic review can be configured at repository or organization scope when supported.

Before enabling:

- identify eligible repositories and excluded data;
- define when reviews run and who owns triage;
- preserve branch protection and required human approvals;
- monitor noise, missed findings, and review delay;
- document an off switch and escalation owner.

Trainer cue: Demonstrate navigation, not fixed UI labels that may change.

Quality gates

Gate	Checks
Ready for first pass	Clear description, focused diff, tests available
Ready for human review	Checks pass; AI comments triaged
Ready for approval	Acceptance criteria met; risks reviewed
Ready for merge	Required approvals and branch protections satisfied

Copilot review comes before human approval. It never replaces it.

Demo PR: intentional defect

```
export function canViewInvoice(user, invoice) {
  return user.role === "admin" || user.id === invoice.ownerId;
}
```

The requirement says suspended users must never view invoices.

```
if (user.status === "suspended") return false;
return user.role === "admin" || user.id === invoice.ownerId;
```

Ask which tests and policy requirements should accompany this suggestion.

Demo narrative: complete lifecycle

1. Open the issue and read acceptance criteria aloud.
2. Inspect the focused branch diff before generated prose.
3. Generate or improve the PR description; correct unsupported claims.
4. Run required checks and request Copilot review.
5. Classify each comment: valid, uncertain, noise, or out of scope.
6. Apply one checked fix and add a regression test.
7. Request human review and record the final decision.

Reviewing suggested fixes

Before applying:

- restate the finding in your own words;
- reproduce the defect or identify the violated requirement;
- inspect the complete patch, not only highlighted lines;
- add or update a test that fails before the fix;
- run broader checks appropriate to the risk;
- retain author ownership of the resulting code.

One-click application saves typing. The reviewer still decides.

What review may catch

- localized correctness issues;
- missing error handling or tests;
- suspicious patterns visible in the diff;
- inconsistent naming or repository conventions;
- simple maintainability concerns.

What it may miss

- unstated product intent;
- cross-system behavior absent from context;
- runtime, operational, legal, or organizational risk;
- subtle authorization and data-boundary flaws;
- deliberately misleading but plausible changes.

Merge conflict resolution

```
Base behavior + branch A intent + branch B intent
                          ↓
              write merged acceptance criteria
                          ↓
                resolve conflict in sandbox
                          ↓
                 run tests and human review
```

If an approved cloud-agent workflow is available, compare its proposal with the manual resolution. Do not blindly choose "ours" or "theirs."

Adoption patterns

Start: Optional first-pass review on a small set of repositories.

Measure: Actionable findings, false positives, review time, rework, and defects that reach users.

Adapt: Tune scope, guidance, ownership, and escalation.

Expand: Only when quality, security, and developer experience meet team criteria.

Document how to pause or disable reviews when a feature is unavailable or a repository needs an exception.

Facilitation prompts

- Which comment needs domain expertise to evaluate?
- Which code reference or test would resolve an uncertain comment?
- Who resolves review noise?
- Which repositories should not use automatic review?
- How will the team detect over-reliance?

Transition: "Now apply the workflow as both author and reviewer."

Lab handoff

You will:

1. follow one pull request from a flawed change brief;
2. classify human and Copilot findings;
3. apply only fixes supported by code references or test results;
4. resolve the prepared conflict without losing either branch goal;
5. record the final human decision.

Final deliverable: Review notes, the corrected project, conflict results, and the final decision. Include initial and correction outputs from `/review` and `/security-review`, with triage and a private training PR reference.

Remember

1. Check generated summaries against the diff.
2. AI review is a first pass; a human reviewer makes the decision.
3. Suggested fixes require reproduction, inspection, and tests.
4. Conflict resolution starts from intended merged behavior.
5. Adoption should be measured and reversible.

Questions and lab readiness

Which review decision needs a human, and what should they check?

- Who is the final reviewer?
- Which check is required before requesting review?
- Is the conflict scenario ready?