GH-600 · GITHUB CERTIFIED: AGENTIC AI DEVELOPER

OPERATION CERTIFIED

The Field Manual & Battle-Tested Study Plan

A complete, gap-free preparation system for the GitHub Agentic AI Developer exam

> *The exam tests whether you can run an AI agent through its operating loop. So this manual makes you **study for the agent exam by running yourself as the agent** — the six exam domains are the agentic loop.*

120	700	~40–60	6	$165	Scenario
MINUTES	PASS / 1000	QUESTIONS	DOMAINS	BASE (REGION VARIES)	+ INTERACTIVE LABS

ROLE-BASED · INTERMEDIATE

DELIVERED BY MICROSOFT · MAINTAINED BY GITHUB

PROCTORED (PEARSON VUE)

RETAKE 24H AFTER 1ST ATTEMPT

Tech With Koushik

MISSION BRIEFING

What GH-600 actually is — and how to read this manual

GH-600 is **not** a "how do I prompt Copilot" test. It is a role-based exam about **operating, supervising, integrating and governing autonomous AI agents** inside a production SDLC, with GitHub as the control plane. Almost every question is a **scenario**: "Given this situation, what should you do / what went wrong / what is the safest configuration?" There may also be short **interactive/lab** tasks.

The mindset shift from your existing certs

- **Foundations/Actions/Copilot** asked "*how does this feature work?*"
- **GH-600** asks "*you're the human-in-the-loop — how do you keep an autonomous agent fast, safe, reliable and auditable?*"
- The winning answer is almost always the one that **preserves human oversight and least privilege while not needlessly slowing delivery**. Hold that compass through every question.

THE BIG IDEA (YOUR UNFAIR ADVANTAGE)

The six domains map **1:1 onto the loop an agent actually runs**. Learn the loop once and you can reconstruct any domain's content from memory — and you'll *recognise* which phase a scenario is really testing.

EXAM VITAL	FACT
Credential	GitHub Certified: Agentic AI Developer
Length / Pass	120 min · 700 of 1000
Questions	~40–60 · scenario + possible interactive
Format	MCQ, multi-select, drag/drop, ordering
Feature scope	Mostly GA features; common Preview features may appear
Retake	24h after attempt 1; longer cooldowns after
Renewal	Annual free online assessment
Sandbox	Try the exam UI at `aka.ms/examdemo`

How to read the callouts in this manual

⚠ TRAP RADAR
The exact spot people fail. Shows the tempting decoy answer vs. what the exam wants.

🎯 EXAM SIGNAL
"When you see keyword X, they're testing concept Y." Pattern-match faster.

◈ PRO MOVE
The senior-engineer instinct that turns a 50/50 into a lock.

◑ CONFIDENCE DIAL
Fill the dots after each domain. All-solid = exam-ready.

The Agentic Loop = your study map (six mission phases)

PHASE 1 · 15–20%
① Plan
Architecture & SDLC — separate planning from action.

PHASE 2 · 20–25% ★ BOSS
② Act with Tools
Tool use, MCP, environments, safe execution.

PHASE 3 · 10–15%
③ Remember
Memory, state, context drift, continuity.

PHASE 4 · 15–20%
④ Evaluate
Success signals, root-cause analysis, tuning.

PHASE 5 · 15–20%
⑤ Orchestrate
Multi-agent coordination, isolation, lifecycle.

PHASE 6 · 10–15%
⑥ Guardrails
Autonomy levels, HITL, least privilege, accountability.

Read the weights like a strategist: **Phase 2 is the single heaviest block** and pairs with Phase 6 to make ≈35% of the exam about tools & safety. Front-load them.

THE STUDY PLAN

The 4-Week Flight Plan (≈10–12 focused hrs/week)

Tuned for your profile — strong GitHub flow + Foundations/Actions/Copilot already earned — so it skips fundamentals and drills the agentic layer. Each week ends by **teaching the concept back out loud** (the single highest-yield study habit). A 10-day sprint variant is at the bottom.

WEEK	FOCUS (MAP TO LOOP PHASE)	DO THIS	DELIVERABLE / PROOF OF SKILL
Week 1 Foundation of autonomy	① PLAN + read the whole study guide once	Complete the 3 Microsoft Learn modules (Foundations / Architecture / Tooling). In a sandbox repo, hand a task to Copilot coding agent and watch the **plan → PR** flow. Write custom instructions + one `*.agent.md`.	A repo with `.github/copilot-instructions.md`, an `AGENTS.md`, and one custom agent that produces a *plan before code*.
Week 2 ★ Boss level	② TOOLS/MCP + ③ MEMORY	Add an **MCP server** to the agent; configure an **allow list** scoped to read-only tools. Configure the **agent firewall** + custom allowlist. Break something on purpose and add error handling/retries. Write an `AGENTS.md` that persists decisions.	MCP + firewall configured; you can explain *on paper* what the firewall does **not** cover.
Week 3 Judgment layer	④ EVALUATE + ⑤ ORCHESTRATE	Wire code scanning as an evaluation signal on agent PRs. Do a real **root-cause triage** of a bad agent PR. Set up a **planner → implementer → reviewer handoff**. Run two agents on isolated branches and resolve a conflict.	A written failure-analysis note classifying a root cause into the 3 buckets; a handoff chain that works.
Week 4 Governance + polish	⑥ GUARDRAILS + full review + mocks	Apply **branch rulesets + CODEOWNERS** to agent config files. Design an autonomy matrix. Take 2 timed mock exams; review **every** wrong answer to a written reason. Reproduce the **60-Second Brain Dump** from memory 3×.	Consistent 85%+ on mocks; brain dump reproduced cold.

◈ PRO MOVE — SPACED REPETITION BEATS RE-READING

Turn every **Trap Radar** and **Exam Signal** in this manual into a flashcard. Review them Day 2, Day 4, Day 8, Day 15, Day 25. The exam rewards *recognition under time pressure*, not recall of prose.

⚠ TRAP RADAR — DO NOT STUDY FROM "BRAIN DUMPS"

Paid "exam dump" sites (Marks4sure, NeoDumps, CertsAdvice, etc.) violate the certification agreement and can void your credential. They're also frequently **wrong** — many mislabel GH-600 as "GitHub Administrator/Azure." Use the official study guide, Microsoft Learn, GitHub Docs, and *reputable* practice-question courses only.

⏱ Accelerated 10-Day Sprint (if the clock is tight)

DAYS	TARGET
1–2	Study guide + Phase ① Plan · hands-on custom instructions/agent
3–5	Phase ② Tools/MCP/firewall (spend the most time here) + Phase ③ Memory
6–7	Phase ④ Evaluate + Phase ⑤ Orchestrate
8	Phase ⑥ Guardrails + the Gotcha Vault (p.10)
9	Timed mock #1 → review every miss
10	Timed mock #2 + reproduce the Brain Dump cold · rest early

Weekly self-check ritual (5 min, huge payoff)

- Cover the page. **Speak** the domain's job in one sentence.
- Name its **2 biggest traps** from memory.
- Fill the **Confidence Dial**. Any dot empty = revisit before moving on.

MISSION PHASE ① · PLAN

Prepare Agent Architecture & SDLC Processes

Integrate agents into the SDLC · separate planning/reasoning from action · build observability & the right amount of autonomy.

15–20%
WEIGHT

Every sub-objective, covered

- **Integrate agents into the SDLC:** identify which discrete steps an agent should own; **define inputs, outputs and explicit success criteria** before it runs; spot & mitigate **anti-patterns**.
- **Plan ≠ Action:** configure planning to be **distinct from execution**; make the agent emit a **structured plan**; **validate the plan**; and **block action until the plan is checked & approved**.
- **Observability & control:** choose the **degree of autonomy** (with guardrails); force the agent to leave **inspectable artifacts inside standard dev tooling** (issues, draft PRs, logs, plan files) — not a bespoke dashboard; insert **human intervention without killing delivery speed**.

Agent anti-patterns to recognise & fix

- **Act-before-plan** — no structured plan, so no review gate.
- **No success criteria** — "done" is undefined, so it can't be evaluated later (this bites you again in Phase ④).
- **Unbounded scope / runaway loops** — no stop conditions.
- **Invisible work** — reasoning left in a chat only; nothing durable in the repo.
- **Over-broad task** — one mega-task instead of decomposed steps.

🎯 EXAM SIGNAL

- "…plan should be **distinct from execution**" → separate planning phase + approval gate.
- "**inspectable** / reviewable artifact" → use native GitHub surfaces (draft PR, issue, plan file, run logs).
- "validate before acting" → the correct answer **gates action on plan approval**.
- "structured plan" → machine-checkable output (steps, files, acceptance criteria), not prose.

⚠ TRAP RADAR

Decoy: "Maximise autonomy to move fastest." **Truth:** the exam almost never rewards *maximum* autonomy — it rewards **right-sized** autonomy with a review gate. Also: don't confuse **reasoning** (thinking/planning) with **action** (writing code, pushing). A question that says the agent "started changing files while still figuring out the approach" is describing a **missing plan/act boundary**, not a tooling bug.

DECISION: IS AUTONOMY RIGHT-SIZED?

Action reversible & low-risk? → **yes** →

Let agent proceed, log artifact

↓ **no**

Require plan approval / human gate before action

◑ CONFIDENCE ○ ○ ○ ○ ○

MISSION PHASE ② · ACT WITH TOOLS · ★ HIGHEST-WEIGHTED = STUDY HARDEST

Implement Tool Use & Environment Interaction

Select/permission tools · configure MCP servers, registries & allow lists · integrate into repos & CI · safe execution with retries, rollbacks & escalation.

20–25%
BOSS WEIGHT

Tools & permissions

- Identify the **minimum** tools a task needs, then configure them and their **permissions** (least privilege — this theme runs through Phase ⑥ too).
- In an agent profile (`*.agent.md`), the `tools` list restricts access; **omitting it grants all tools**. Common tool verbs: `read` , `search` , `edit` , `execute` .

MCP (Model Context Protocol) — the money topic

- **Add an MCP server** as a tool to an agent; configure the **GitHub remote MCP server**; configure **MCP registries**; configure **MCP allow lists**.
- **Scope the tools, not just the server:** allowlist specific **read-only** tools; **avoid the `*` wildcard** so the agent can't invoke write/destructive actions on an approved server.
- Coding agent runs **allowlisted MCP tools without prompting** for approval — so a narrow allowlist *is* your control.
- The **built-in GitHub MCP defaults to read-only** on the current repo unless you broaden the token.
- **Remote MCP servers that require OAuth are not yet supported** by the coding agent.
- MCP is governed by the **Copilot MCP policy** (org/enterprise), *not* by editor/extension allowlists.

Environment integration & safe execution

- Evaluate **execution context**; scope agent to a **specific repo** and a **branch-based** scope; invoke it inside a **CI workflow**; let it act autonomously (create branches/PRs); handle **environment-specific constraints**.
- Safe paths: **error handling · retries · rollbacks · escalation paths · traceability/accountability** for every action.
- `copilot-setup-steps.yml` pre-installs deps before the agent runs — the job **must be named** `copilot-setup-steps` or it's ignored; set **least-privilege permissions**; Copilot gets its own token.

⚠ TRAP RADAR — THE MOST-MISSED CONCEPT ON THE EXAM

The agent firewall does NOT protect MCP traffic. The firewall (on by default, with a recommended allowlist) applies **only to processes the agent starts via its Bash tool**, and only inside the Actions appliance. It does **not** apply to **MCP servers** or to processes in **Copilot setup steps**. If a question shows data leaving through an MCP server, "the firewall will block it" is the **decoy**. Control MCP via **tool allow lists + read-only scoping + the MCP policy**.

⚠ TRAP RADAR — ALLOWLIST DIRECTION

Disabling the recommended allowlist does **not** "open up" access — with the firewall still on it **reduces** what the agent can reach (deny-by-default). Allowlist entries are *exemptions*. Also: a **domain** entry covers subdomains; a **URL** entry is limited to that scheme+host+path and its descendants.

🎯 EXAM SIGNAL

- "prevent **data exfiltration**" → agent firewall + allowlist (for Bash-tool egress).
- "agent ran an MCP write it shouldn't" → tighten the **MCP tool allow list** / go read-only.
- "deps fail to install / host blocked" → add to **custom allowlist** or handle in setup steps.
- "run without approval" → an **allowlisted MCP tool** (coding agent).

◈ PRO MOVE

Memorise the firewall's **three blind spots** as one phrase: **"Bash-only, Actions-only, MCP-and-setup-steps excluded."** At least one scenario turns on exactly this.

◑ CONFIDENCE ○ ○ ○ ○ ○

MISSION PHASE ③ · REMEMBER

Manage Memory, State & Execution

Pick memory types · persist state as durable artifacts · detect & correct context drift · keep continuity across tools and environments.

10–15%
WEIGHT

The three memory types (know the boundaries cold)

TYPE	SCOPE	USE IT FOR
Short-term	Within the session / context window	The current task's working state
Long-term	Persists across sessions	Durable project conventions/decisions
External	Outside the model (files, issues, a store)	Large/retrievable knowledge, shared state

- **Scope memory to task-relevant info** — don't dump everything into context.
- Define **expiration, pruning and reset** rules so memory doesn't rot.

Persist state & fight context drift

- Capture **progress & decisions as durable artifacts** — commits, PR descriptions, issue comments, a plan/decision file, or `AGENTS.md` notes.
- **Resume without repeating steps or diverging** from prior decisions.
- **Detect & correct drift** during long-running execution — when the agent slowly wanders from the original intent.

Continuity across tools & environments

- **Share state** between agents/tools.
- **Prevent conflicting context** (two sources disagree).
- **Prevent stale context** (old info still driving decisions).

🎯 EXAM SIGNAL — MEMORY VOCABULARY DECODER

- "resume **without repeating** work" → persist state as **durable artifacts**.
- "over a long run it **forgot / wandered from** the goal" → **context drift** → detect & correct.
- "the same fact is stored two different ways" → **conflicting** context.
- "acting on **outdated** info" → **stale** context → refresh/expire.
- "needs to remember **across sessions**" → **long-term/external**, not short-term.

⚠ TRAP RADAR

Two classics: (1) shoving **everything** into the context window "so it remembers" — the exam wants **scoped** memory + pruning. (2) Confusing **stale** (old but singular) with **conflicting** (two disagreeing sources) — the fixes differ: *expire/refresh* vs *reconcile/dedupe*.

◈ PRO MOVE

In GitHub terms, "durable artifact" almost always means **something visible in the repo/PR/issue** — that's also what makes the work **auditable** (Phase ①) and **evaluable** (Phase ④). One artifact, three domains.

◑ CONFIDENCE ○ ○ ○ ○ ○

MISSION PHASE ④ · EVALUATE

Perform Evaluation, Error Analysis & Tuning

Define success signals · find root causes from logs/traces/artifacts · tune instructions, memory and tool access.

15–20%
WEIGHT

Success criteria & evaluation signals

- Specify **expected outcomes and operational constraints** up front.
- Use **both qualitative and quantitative** signals — not numbers alone.
- **Align evaluation with development intent** (did it solve the real problem, not just pass a test?).
- Generate signals automatically with **scanning tools** — code scanning, secret scanning, tests, linters, Code Quality checks on agent PRs.

Root-cause analysis — the 3 buckets (memorise)

ROOT CAUSE	LOOKS LIKE	FIX BY
Reasoning error	Bad plan/logic; misunderstood task	Revise instructions / constraints
Tool misuse	Wrong tool, wrong args, unsafe call	Refine tool usage & access
Context / env issue	Missing deps, wrong scope, stale/bad context	Fix environment / memory

Evidence lives in: **logs, plans, traces, outputs, and workflow artifacts.**

Tuning levers (in order of preference)

- Revise **instructions, workflows or constraints**.
- Refine **memory usage**.
- Refine **tool usage & tool access**.

⚠ TRAP RADAR — THE REFLEX THAT FAILS YOU

When an agent misbehaves, the tempting answer is **"switch/retrain the model."** On GH-600 the correct fix is almost always to **tune the controllable layer** — instructions, constraints, memory scope, or tool access. You govern the harness, not the weights.

🎯 EXAM SIGNAL

- "which **signal / evidence** to inspect" → logs, traces, plans, outputs, workflow artifacts.
- "agent picked the **wrong tool**" → tool misuse → refine tool **access**.
- "passed tests but solved the **wrong problem**" → misaligned success criteria / need qualitative signal.
- "generate signals **automatically**" → scanning tools on the PR.

◈ PRO MOVE

Separate **symptom** ("the PR broke the build") from **root cause** ("the agent lacked the dependency because setup steps were misconfigured"). Questions that list a symptom want you to trace it to **one of the three buckets** before choosing a fix.

◑ CONFIDENCE ○ ○ ○ ○ ○

MISSION PHASE ⑤ · ORCHESTRATE

Orchestrate Multi-Agent Coordination

Coordinate multiple agents with the right pattern & isolation · make behaviour observable · recover from failures · manage the agent lifecycle.

15–20%
WEIGHT

Operate & manage multi-agent workflows

- Apply an **orchestration pattern** — sequential **handoffs** (planner→implementer→reviewer), **parallel** workers, or an **orchestrator→subagents** model.
- **Isolate agents for parallel execution** — separate branches / contexts so they can't collide.
- Detect & resolve **conflicts**: overlapping code changes, **duplicated effort**, and **contradictory outputs**.

Observability for multi-agent behaviour

- Produce **artifacts suitable for review & audit**.
- Document **key decisions, handoffs and outcomes** across agents.
- Support **post-hoc analysis** of what happened.

Failures & recovery

- Distinguish **failed vs partial vs stalled** executions — they are *not* the same.
- Respond to **degraded coordination** across agents.
- Recovery patterns: **rollback** and **human-in-the-loop**.

Agent lifecycle

- **Add** agents to an existing workflow.
- **Update / reconfigure / replace** an agent **without disrupting active workflows**.
- **Retire** an agent while **preserving auditability & continuity**.

⚠ TRAP RADAR

(1) Running agents in parallel **without isolation** → the exam expects branch/context isolation *before* parallelism. (2) Treating a **stalled** agent as **failed** — a stalled/partial run may just need a nudge or resume, not a full rollback. (3) Forgetting that **replacing/retiring** an agent must **keep the audit trail intact**.

🎯 EXAM SIGNAL

- "two agents edited the **same files**" → isolation + conflict resolution.
- "agents did the **same work twice**" → duplicated-effort conflict → coordination/handoff.
- "swap an agent **mid-flight**" → lifecycle change without disruption + auditability.
- "who did what, when" → decision/handoff **documentation artifacts**.

◈ PRO MOVE — HANDOFFS ARE A CONCRETE GITHUB FEATURE

Custom agents support **handoffs** in the agent profile (`label` , target `agent` , pre-filled `prompt` , and `send: true/false`). `send:false` = a human reviews before the next step — that's orchestration *and* a guardrail at once.

◑ CONFIDENCE ○ ○ ○ ○ ○

MISSION PHASE ⑥ · GUARDRAILS

Implement Guardrails & Accountability

Right-size autonomy by risk · human-in-the-loop for judgment calls · least-privilege & controlled paths · preserve velocity.

10–15%
WEIGHT

Define autonomy levels

- Classify each agent action by **operational, security and compliance risk** to right-size human intervention.
- Assign autonomy to **maximise delivery speed while staying compliant** with org security + **Responsible AI** standards.

Guardrails & human-in-the-loop

- Identify the **subset of actions needing human judgment** — gate only those.
- **Block** actions that violate security/compliance/Responsible-AI policy.
- **Least-privilege** scoping of permissions & execution contexts.
- Require **explicit authorization / controlled paths** for **irreversible or compliance-sensitive** changes.
- **Preserve velocity** by cutting approvals that don't materially reduce risk.

GitHub's built-in coding-agent guardrails (high-yield facts)

- Agent **only opens PRs**; it **cannot push to the default branch**, and **cannot approve, merge, or mark its own PR "ready for review."**
- Work happens on a **dedicated** `copilot/` **branch**; the agent is subject to **branch protections, rulesets & required checks** like any human.
- Its PRs **don't auto-run workflows** — a human with write access must click **"Approve and run workflows."**
- The **PR requester cannot approve their own** agent PR (Required-approvals rule holds).
- **Hidden-character filtering** mitigates prompt injection from issues/comments.
- Protect Copilot/MCP config files with **CODEOWNERS + "Require review from Code Owners."**

⚠ TRAP RADAR — THE VELOCITY/SAFETY TIGHTROPE

The wrong extremes are both traps: **gate everything** (kills delivery — violates "preserve velocity") or **gate nothing** (unsafe). The right answer gates **irreversible / high-risk / compliance-sensitive** actions and lets the low-risk reversible ones flow. Also a hard fact to remember: the coding agent **cannot merge** — any option that has it self-merging is wrong.

🎯 EXAM SIGNAL

- "**irreversible** / production / compliance-sensitive" → require explicit authorization / HITL.
- "who approves the agent's PR?" → a human who is **not the requester**.
- "protect the `.github` /MCP config" → **CODEOWNERS** + required review.
- "reduce approvals but stay safe" → gate by **risk class**, not by volume.
- "least privilege" → minimal token perms in setup steps + read-only MCP tools + scoped repo.

◈ PRO MOVE — THE AUTONOMY MATRIX

Picture a 2×2: **reversible/irreversible** × **low/high risk**. Autonomous only in the reversible+low-risk cell; everything else needs a gate. If you can place a scenario's action in that grid, the answer picks itself.

◑ CONFIDENCE ○ ○ ○ ○ ○

WHERE PEOPLE FAIL

🔒 The Gotcha Vault — every high-risk trap in one place

If you only re-read one page the night before, make it this one. These are the exact distinctions that separate a 690 from a 720.

TOPIC	THE DECOY YOU'LL BE TEMPTED BY	WHAT THE EXAM ACTUALLY WANTS
Agent firewall scope ⚠⚠⚠	"The firewall stops the MCP server from leaking data."	Firewall = **Bash-tool egress inside the Actions appliance only**. It does **NOT** cover MCP servers or Copilot setup steps. Control MCP via allow lists.
Allowlist direction	"Turn off the allowlist to give the agent less access."	With firewall on, allowlist = **exemptions**. Disabling it **reduces** reachable hosts; entries are what's **permitted**.
Domain vs URL entry	They behave the same.	**Domain** entry includes subdomains; **URL** entry is limited to that scheme + host + path (+ descendants).
MCP tool scoping	"Approve the whole MCP server (`*`)."	Allowlist **specific read-only tools**; avoid `*`. Allowlisted tools run **without approval prompts**.
MCP + OAuth	"Just add the OAuth remote MCP server."	Remote MCP servers that **require OAuth are not yet supported** by the coding agent.
Instruction precedence	"Repository instructions always win."	Priority is **Personal > Repository > Organization**, but **all applicable sets are applied**. Conflicts resolve **non-deterministically** — so avoid conflicts.
Setup-steps job name	"Any job name works."	The job **must** be named `copilot-setup-steps` or Copilot ignores it. Use least-privilege `permissions`.
Can the agent merge?	"Let the agent merge once tests pass."	**No.** Agent only **opens** PRs; can't push to default branch, can't approve/merge/mark-ready its own PR. A human merges.
Who approves?	"The person who asked for the PR approves it."	The **requester cannot approve their own** agent PR (Required-approvals rule).
Workflows on agent PRs	"CI runs automatically on the agent's PR."	They **don't** — a write-access human must click **"Approve and run workflows."**
Fixing a bad agent	"Switch or retrain the model."	Tune the **controllable layer**: instructions, constraints, memory scope, tool access.
Stale vs conflicting context	Treat them as the same problem.	**Stale** = old-but-single → expire/refresh. **Conflicting** = two disagreeing sources → reconcile/dedupe.
Failed vs stalled vs partial	All three ⇒ roll everything back.	Diagnose first; a **stalled/partial** run may resume, only a true failure needs rollback + HITL.
Parallel agents	"Run them in parallel for speed."	Only after **isolation** (separate branches/contexts) to avoid overlapping-change conflicts.
Autonomy level	"Max autonomy = max velocity = best."	**Right-size** by risk; gate irreversible/compliance-sensitive actions; keep low-risk reversible work flowing.
Config-file protection	"Branch protection alone protects `AGENTS.md` /MCP config."	Add **CODEOWNERS + Require review from Code Owners** so config edits need the right team's approval.

FAST RECOGNITION

Myth-Busting + the Keyword→Concept Decoder

Myth vs Reality

MYTH	REALITY
GH-600 is a Copilot prompt-engineering test.	It's about **operating & governing** agents in the SDLC.
The most autonomous setup scores highest.	**Right-sized** autonomy + oversight wins.
The firewall secures MCP.	It doesn't — MCP is out of scope for the firewall.
Custom instructions are all-or-nothing.	Layered (**personal/repo/org**) + **path-specific** files.
The agent can merge if checks pass.	Never — a **human** merges.
Bad output ⇒ change the model.	Tune **instructions / tools / memory**.
More approvals = safer.	Gate by **risk**; needless gates hurt velocity (a scored value).

Custom instructions & agent files — quick reference

- `.github/copilot-instructions.md` — repo-wide instructions.
- `.github/instructions/*.instructions.md` — **path-specific** (frontmatter `applyTo`).
- `AGENTS.md` — nested allowed; **nearest in the tree wins**; root file = primary.
- `CLAUDE.md` / `GEMINI.md` — root only, alternative agent instructions.
- `*.agent.md` — a **custom agent** (YAML frontmatter: `name, description, tools, model, target, handoffs`; prompt ≤ 30,000 chars).
- `excludeAgent: code-review | cloud-agent` — keep a file from one agent.

🎯 Keyword → Concept Decoder

IF THE QUESTION SAYS…	THEY'RE TESTING…
"distinct from execution", "before acting"	Plan/act boundary + approval gate (①)
"inspectable / auditable artifact"	Native GitHub surfaces (①/⑤)
"data exfiltration", "blocked host"	Agent firewall + allowlist (②)
"external tool", "MCP", "without prompting"	MCP allow list + read-only scoping (②)
"install deps", "prepare environment"	`copilot-setup-steps.yml` (②)
"resume", "forgot", "drifted"	State persistence / context drift (③)
"across sessions"	Long-term / external memory (③)
"root cause", "wrong tool", "logs/traces"	Error analysis → 3 buckets (④)
"same files", "duplicated", "contradictory"	Multi-agent conflict + isolation (⑤)
"replace/retire agent", "who did what"	Lifecycle + audit artifacts (⑤)
"irreversible", "compliance", "Responsible AI"	HITL + autonomy right-sizing (⑥)
"least privilege", "scope permissions"	Guardrails / token & tool scoping (⑥)
"who approves", "merge"	Agent can't merge; requester can't approve (⑥)

● ◈ PRO MOVE — READ THE LAST SENTENCE FIRST

Scenario questions front-load a lot of noise. Read the **final question line** first ("What should you do to *minimise risk without slowing delivery*?"), then hunt the stem for only the facts that constraint touches. The italic qualifier is usually the tie-breaker between two "correct-looking" options.

THE SECRET WEAPON

🧠 The 60-Second Brain Dump + Test-Day Tactics

Practice writing this from memory until you can do it cold. In the first minute of the exam (on scratch paper / the whiteboard tool) dump it — now every trap is in front of you instead of in your nerves.

◆ THE DUMP — 8 lines that cover the whole exam

1	**LOOP:** Plan → Tools → Memory → Evaluate → Orchestrate → Guardrails (= the 6 domains). Weights: 20-25 is Tools (boss), 10-15 are Memory & Guardrails.
2	**FIREWALL:** Bash-only, Actions-only. **NOT** MCP, **NOT** setup steps. Allowlist = exemptions (off ⇒ less access). Domain = +subdomains; URL = exact path.
3	**MCP:** scope *tools* not servers; read-only; no `*`; allowlisted runs w/o prompt; OAuth remote = unsupported; GitHub MCP defaults read-only.
4	**MEMORY:** short(session) / long(cross-session) / external. Durable artifacts to resume. Drift=wander; Stale=old(refresh); Conflict=disagree(reconcile).
5	**ROOT CAUSE:** reasoning · tool-misuse · context/env. Evidence: logs/plans/traces/outputs/artifacts. Fix = tune instructions/tools/memory (NOT the model).
6	**MULTI-AGENT:** isolate before parallel; conflicts = overlap/duplicate/contradict; failed≠stalled≠partial; lifecycle keeps audit trail.
7	**GUARDRAILS:** gate irreversible/compliance only; least privilege; agent opens PR only — can't merge/approve-own/push-default; requester≠approver; CI needs "Approve and run".
8	**INSTRUCTIONS:** Personal > Repo > Org (all applied). Files: copilot-instructions / *.instructions.md (applyTo) / AGENTS.md (nearest wins) / *.agent.md. Setup job = `copilot-setup-steps`.

⏱ Time & tactics (120 min ≈ 2 min/question)

- **Flag & move.** Never spend >3 min on one item — mark it, bank the easy points, return later.
- **Two "correct" options?** Pick the one that best serves the **italic qualifier** (safety *without* slowing delivery / least privilege / least disruption).
- **Eliminate absolutes** in guardrail Qs — "always/never fully autonomous" answers are usually wrong.
- On interactive/lab tasks: do the **smallest correct action**; don't over-configure.
- **Read multi-selects twice** — note exactly how many to choose.

✅ Go/No-Go pre-flight checklist

☐ I can recite the 8-line Brain Dump cold.
☐ I can name the firewall's 3 blind spots.
☐ I've configured a real MCP allow list + firewall.
☐ I've built a custom agent + handoff chain.
☐ I can classify any failure into 3 root causes.
☐ I know exactly what the agent *cannot* do to a PR.
☐ Mocks ≥ 85%, every miss reviewed to a reason.
☐ All six Confidence Dials filled solid.
☐ Tested the exam UI at aka.ms/examdemo.
☐ ID ready, room/space checked, well rested.

● THE COMPASS, ONE MORE TIME

When stuck, choose the option that keeps a **human in the loop**, enforces **least privilege**, leaves an **auditable artifact**, and **doesn't needlessly slow delivery**. That single sentence resolves a surprising share of the exam.

📚 Trusted resources only

- **Official study guide** — Microsoft Learn, Exam GH-600.
- **Microsoft Learn modules:** Foundations of Agentic AI in GitHub · Designing Agent Architecture & SDLC Integration · Tooling, MCP & Agent Execution Environments.
- **GitHub Docs:** Copilot coding agent — custom agents, custom instructions, MCP, agent firewall, risks & mitigations, build guardrails.
- **Hands-on** a real repo > any amount of reading.
- **Avoid** braindump/"real questions" sites — against policy & often wrong.

You've operated the loop end-to-end. Now go get your PR merged. 🚀

Operation Certified · GH-600 Field Manual — built from the official objectives, verified against GitHub Docs.