Building Agentic Workflows with GitHub Copilot

Hector Flores / htek.dev / Developer session

Use this handout as a reading guide. Demo sections are proposed storyboards, not recordings or claims of executed runs.

Contents

Building Agentic Workflows with GitHub Copilot
The artifact is no longer the only thing you design
From suggestions to governed processes
A goal, a loop, and a stop condition
Agency is not unlimited authority
One issue. Three surfaces.
A good goal describes observable success
Constraints are part of the solution
Better context beats a bigger prompt
Use the simplest sufficient process
Three surfaces. Different responsibilities.
Put repository events to work
Separate reasoning from write execution
Tool management is architecture
The issue enters the workflow
The useful part was not the label
Define the process in code
Autonomy, delegation, and orchestration
Orchestrate the process, not every thought
Investigate with structured handoffs
Design for the run that does not finish
Knowledge is not capability
Turn a vague action into a narrow contract
Rejection is useful feedback
Human-in-the-loop, not human-as-cleanup
What a polished demo can hide
Scenarios worth trying
Scale the operating model

Pick one measurable workflow
Define the outcome.
Check the recording environment
Inspect the effective configuration
A reusable investigation process
What do you actually need?
Take the material with you

HECTOR FLORES / HTEK.DEV

Building Agentic Workflows with GitHub Copilot

Define the outcome. Bound the action. Verify the result.

GitHub: Dynamic workflows / GitHub Agentic Workflows / Hector: Three pillars of Agentic DevOps

Speaker notes

Purpose: set an engineering, not a magic-demo, expectation. This session explores how developers can adopt agent-driven workflows while retaining accountability. We will follow one synthetic engineering issue across repository automation, code-defined orchestration, and deterministic tools. The three demo sections currently contain proposed storyboards; recordings will be produced only after script approval. Transition: what actually changes about a developer's job?

The artifact is no longer the only thing you design

Before

Specify each implementation step

Now also

Specify the outcome and boundaries

Always

Own the evidence and decision

Intent becomes an engineering artifact.

Hector: Context engineering / Hector: Three pillars of Agentic DevOps

Speaker notes

Purpose: establish the thesis as a proposed way of working, not a universal replacement for programming. Requirements, acceptance criteria, context, and workflow definitions deserve the same review discipline as code. Humans still design systems and inspect results. Hector's feedback-loop framing motivates this shift. Transition: locate it in Copilot's evolution.

From suggestions to governed processes

Assistance

Suggest code in the editor

Agent loop

Plan, act, test, iterate

Workflow

Reuse steps, rules, and handoffs

The shift is **who owns the next step**.

Introducing GitHub Copilot / Copilot general availability / Copilot CLI general availability / Dynamic workflows announcement

Speaker notes

Purpose: explain evolution without pretending older modes disappeared. GitHub introduced Copilot's technical preview on June 29, 2021 and made it generally available on June 21, 2022. The modern terminal-native Copilot CLI became GA on February 25, 2026. Dynamic workflows were announced October 1, 2026. These are selected milestones, not an exhaustive product history. Do not confuse the modern agentic CLI with the older gh copilot shell-command extension. Transition: define agentic work.

A goal, a loop, and a stop condition

Observe	Decide
Relevant context	Analysis and judgment
Act	**Verify**
Permitted tools	Evidence and feedback

Stop when the **acceptance criteria** are met, not when the answer sounds complete.

GitHub: Dynamic workflows / GitHub Agentic Workflows

Speaker notes
Purpose: provide a conceptual model rather than a formal product definition. A workflow combines work that needs judgment with operations that should be mechanically checked. Ask the audience to identify where they already have a repetitive observe-decide-act loop. Transition: automation is useful only when authority is explicit.

Agency is not unlimited authority

The agent may	A human still owns
Investigate within scope	The goal and acceptable risk
Propose a change	The delivery decision
Request a permitted effect	Policy and approval boundaries

More autonomy is a choice, not a maturity score.

gh-aw: Safe outputs / GitHub: Dynamic workflows

> **Speaker notes**
> Purpose: separate capability from authority. Safe-output jobs can execute permitted writes automatically; they do not imply that someone reviewed each action. Similarly a workflow checkpoint creates a pause, not necessarily an authenticated approval record. We must design the actual decision and its evidence. Transition: use a small engineering scenario to make these concepts tangible.

One issue. Three surfaces.

A Task Tracker endpoint handles an invalid status inconsistently.

GitHub

Triage the issue

Copilot workflow

Investigate the behavior

Extension tool

Validate the proposal

Human

Review the evidence

Synthetic fixture, not a claim about a current production defect.

Public Task Tracker workflow examples / Hector: Agentic Workflows hands-on guide

Speaker notes
Purpose: introduce the common thread. Hector's public repository demonstrates triage, reviews, docs updates, and weekly digests around a Task Tracker API. We will use the same domain but an isolated fixture. The invalid-status scenario is controlled demonstration data. There is no assumed automatic bridge from a GitHub event into a local CLI session. Transition: write the goal before adding agents.

A good goal describes observable success

Weak: "Fix our API."

Better: "Explain why invalid task statuses are accepted. Produce a failing acceptance test and a minimal proposal. Do not change unrelated endpoints."

Outcome + scope + evidence + stop condition

Hector: Context engineering / GitHub: Dynamic workflows

Speaker notes
Purpose: show a reusable requirements pattern. The better instruction is our proposed example, not product syntax. Add who owns approval and what must happen if

evidence is missing. Avoid asking agents to infer unstated product behavior. A developer should first agree whether invalid input should be rejected and what response contract is intended. Transition: constraints are not an afterthought.

Constraints are part of the solution

Scope: this issue, these files, this repository
Authority: read, propose, or execute?
Resources: agreed budget and concurrency
Escalation: missing evidence, ambiguity, or rejected validation

gh-aw: GitHub tools and allowlists / gh-aw: Safe outputs / gh-aw: Network policy

Speaker notes
Purpose: translate a broad goal into an operating envelope. These are independent dimensions: a tool can be exposed but denied by credentials, or have credentials but be inappropriate for the task. Agree on resource limits from user requirements and measured cost; do not present guessed budgets as guaranteed safety. Transition: bounded authority still needs relevant context.

Better context beats a bigger prompt

Intent

Requirements and non-goals

Grounding

Relevant code and conventions

Feedback

Failing tests and real results

Give each stage **what it needs**, not everything you have.

Hector: Context engineering / Hector: Three pillars of Agentic DevOps

Speaker notes
Purpose: adapt Hector's context-engineering principle. A workflow should pass evidence and bounded questions instead of dumping the entire conversation into every agent. Keep trusted instructions distinct from untrusted issue text. Update durable conventions through review rather than automatically canonizing every model answer. Transition: decide how much orchestration the task actually needs.

Use the simplest sufficient process

Task shape	Start with
Small local question	One agent
Dependent stages	A sequence
Independent investigations	Parallel workers
Consequential action	Verification and a human handoff

An agent count is not an outcome metric.

GitHub: Dynamic workflows / Hector: The maturity curve

Speaker notes

Purpose: counter the temptation to build an agent swarm for every request. Hector's maturity-curve essay argues for earned simplicity; present that as his experience, not a universal law. Parallelism helps independent work but adds coordination and resource costs. If workers must edit the same files, first solve isolation and integration. Transition: map the GitHub surfaces before showing them.

Three surfaces. Different responsibilities.

Surface	Primary role
GitHub Agentic Workflows	Event/schedule-driven repository automation in Actions
Copilot dynamic workflows	A reusable process defined in code
CLI extensions	Session-aware capabilities implemented in JavaScript

An extension can **host** a dynamic workflow.

GitHub Agentic Workflows / GitHub: Dynamic workflows / GitHub: Copilot extensions

Speaker notes

Purpose: prevent terminology collisions. Dynamic workflows are a feature across Copilot CLI, the Copilot app, and SDK; the CLI is our demonstration surface. Hookflow is a separate community governance system, not a synonym for gh-aw. CLI extensions are also distinct from the retired GitHub App-based Copilot Extensions integration and ordinary IDE extensions. Transition: start where repository events happen.

Put repository events to work

Trigger

Issue, PR, schedule

Definition

Markdown + frontmatter

Actions

Agent execution

Result

Permitted outputs

GitHub Agentic Workflows complements CI/CD.

GitHub Agentic Workflows / Hector: Agentic Workflows hands-on guide

Speaker notes
Purpose: introduce gh-aw without suggesting that all CI should become probabilistic. Deterministic builds and tests still belong in normal automation. Markdown instructions and configuration are compiled into Actions workflows. Useful tasks include repository triage and maintenance. Revalidate exact schema and authentication before using historical tutorial snippets. Transition: where do effects occur?

Separate reasoning from write execution

Agent step

Analyze with scoped reads

Request

Structured permitted output

Handler job

Validate and execute

An upper bound is not a promise of exactly one action.

gh-aw: Safe outputs

Speaker notes
Purpose: explain safe outputs as separation of permissions, not perfect model safety. The agent requests actions; separate permission-controlled jobs process them. A configured maximum limits actions rather than requiring that count. Current docs describe implicit default outputs when none are configured, so specify and inspect the

effective configuration. Some outputs publish automatically. Transition: layer the controls rather than relying on one.

Tool management is architecture

Question	Control to inspect
What can it call?	Toolsets and individual allowlists
What can it access?	Credential and repository scope
Where can it connect?	Network and hosted-retrieval policy
What can it change?	Outputs, permissions, policy gates

Exposure, permission, and approval are different things.

gh-aw: Tools / gh-aw: GitHub tools and allowlists / gh-aw: Network policy

Speaker notes

Purpose: make tool management the central technical lesson. Toolsets load API groups; allowed narrows tools and can bound calls. Do not assume a runner-domain allowlist also governs provider-hosted retrieval. MCP connects external tools but does not remove authentication, data-scope, or trust concerns. Read the effective configuration and test a denied action as well as a successful one. Transition: walk through the first proposed recording.

Demo 1 / proposed storyboard / not recorded

The issue enters the workflow

Open a synthetic invalid-status issue.
Observe the configured event and run.
Inspect the scoped read tools.
Review the bounded triage result.
Evidence: issue, workflow revision, run, and actual output.

Public Task Tracker workflow examples / gh-aw: GitHub tools and allowlists / gh-aw: Safe outputs

Speaker notes
Recording cue: start on the synthetic issue, then show only the configuration that explains authority. End at the posted classification, not a code merge. Before capture, obtain a fresh successful rehearsal and check labels, credentials, schema, and permissions. Existing scheduled runs in the source repository were not recording-ready at planning time. Static fallback: these four steps plus captured screenshots and run evidence. Transition: why did this work?

The useful part was not the label

A clear goal made the decision bounded.
Relevant context made it explainable.
Narrow output permissions limited the effect.
A human still decides what work to accept.
Automate the repetitive decision without hiding its consequences.

GitHub Agentic Workflows / gh-aw: Safe outputs / Hector: Context engineering

Speaker notes
Purpose: return from a demo to a transferable pattern. Triage is valuable when it routes work consistently and preserves evidence, not merely when an AI leaves a comment. Review classification mistakes and unexpected outputs before broadening automation.

Do not equate a friendly explanation with correct judgment. Transition: what if the investigation itself has a repeatable structure?

Copilot dynamic workflows / public preview

Define the process in code

Combine **deterministic operations** with **agent judgment**.

Sequence. Branch. Parallelize. Verify. Pause. Resume.

Available across CLI, app, and SDK; CLI demonstrations require experimental features.

GitHub: Dynamic workflows / GitHub: Using dynamic workflows / Dynamic workflows announcement

Speaker notes

Purpose: introduce dynamic workflows as their own product feature. They live inside extensions and reuse an author's process rather than leaving every handoff to the model. The same process can produce different agent answers. Check current eligibility and billing caveats in documentation; do not promise universal availability from an older announcement. Transition: compare it to familiar CLI modes.

Autonomy, delegation, and orchestration

Feature	Who chooses the process?
Autopilot	The agent keeps choosing next steps
Fleet	The agent divides and coordinates work
Dynamic workflow	The author defines steps and handoffs in code

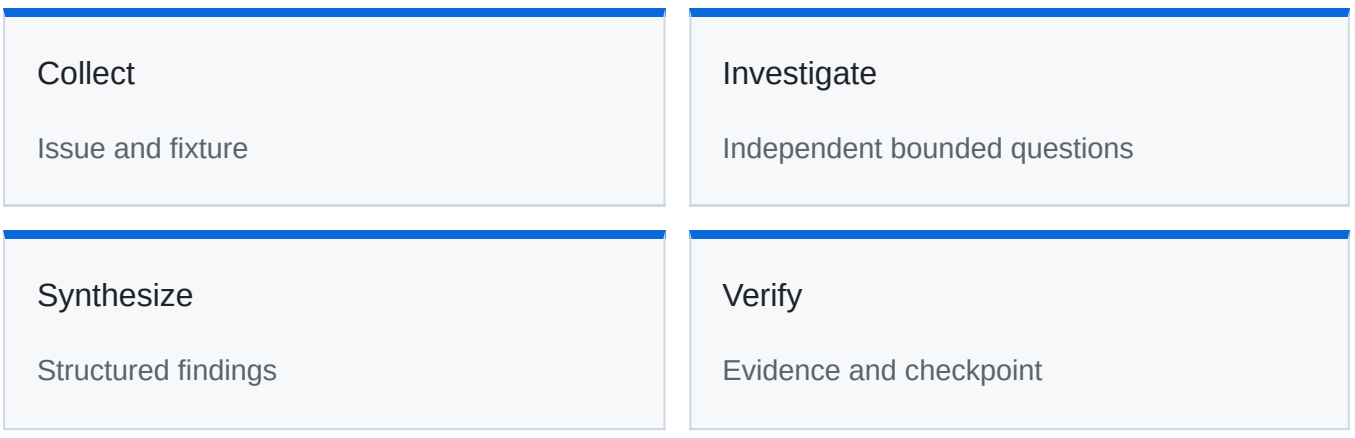
Not three names for "more agents."

GitHub: Dynamic workflows

Speaker notes

Purpose: give the audience a feature-selection model. Autopilot concerns autonomy; fleet encourages agent-selected parallel delegation; a dynamic workflow encodes a process. None inherently proves correctness. A workflow may need only one agent between deterministic steps. Transition: show a process whose handoffs are worth reusing.

Orchestrate the process, not every thought



Write code for **control flow**. Ask agents for **judgment**.

GitHub: Dynamic workflows / GitHub: Using dynamic workflows

Speaker notes
Purpose: explain why deterministic orchestration is useful even though the underlying model is variable. Each worker should have a distinct objective, permitted tools, evidence requirements, and a completion contract. Preserve incomplete results. A structured object helps handoff, but semantic validity still needs checking. Transition: the second storyboard deliberately starts with a human handoff.

Demo 2 / proposed storyboard / not recorded

Investigate with structured handoffs

Explicitly pass the triaged issue into Copilot.
Collect the fixture and acceptance criteria.
Ask API and test investigators distinct questions.
Combine evidence; pause before applying a change.
No assumed Actions-to-local-CLI bridge.

GitHub: Dynamic workflows / GitHub: Using dynamic workflows / Public Task Tracker workflow examples

Speaker notes
Recording cue: show the deliberate handoff and named workflow, then phases and independent worker results. Finish at evidence-backed findings and a review checkpoint. Capture actual usage rather than promising a fixed cost. Include an incomplete-result example in supporting material. Static fallback: this process diagram and reviewed structured-result excerpts. No recording is represented as completed yet. Transition: a real process must survive failure.

Design for the run that does not finish

Keep partial evidence visibly incomplete.
Make retried side effects idempotent.
Validate meaning, not only JSON shape.
Observe limits and pause at explicit boundaries.
"The workflow stopped" is not "the work is done."

GitHub: Using dynamic workflows / Copilot CLI programmatic reference

Speaker notes
Purpose: introduce operational discipline. The bundled runtime guide documents journaled steps and distinct failure behaviors; code against the tested runtime, not an old snippet. Repeated execution can happen after failure, so journaling is not an exactly-once transaction. Credit ceilings can be soft. Programmatic runs do not show interactive permission prompts; configure permitted capabilities deliberately. Transition: make the low-level operation less ambiguous.

Copilot CLI extensions / experimental

Knowledge is not capability

Instructions / skills

What the agent should know

Extensions / tools

Operations implemented in code

Hooks / policy

Checks at defined boundaries

A CLI extension runs alongside the session as a Node.js process.

GitHub: Copilot extensions / Hector: Knowledge, capability, enforcement / Hector: CLI extensions guide

Speaker notes

Purpose: adapt Hector's layered architecture while grounding product facts in current docs. An extension can expose tools, commands, and session-aware behavior. A hook is a lifecycle integration, not a complete sandbox. MCP is an external tool integration surface; a plugin can distribute capabilities. Use the surface matching the need. Transition: choose a deterministic task worth extracting into code.

Turn a vague action into a narrow contract

Not: "Run whatever command fixes this."

Instead: validate a proposed task-status contract.

Input	Fixed code
Known schema	No shell interpolation

Result
Pass/fail + reasons

GitHub: Copilot extensions / Hector: Vibe testing and false confidence

Speaker notes

Purpose: demonstrate granular tool design. The included validator checks a synthetic status-contract proposal with fixed code and returns structured evidence. It does not write a repository, execute arbitrary commands, or prove an application is correct. The

operation has predictable mechanics even if the agent's choice of proposal varies.
Transition: observe useful rejection rather than only a happy path.

Demo 3 / proposed storyboard / not recorded

Rejection is useful feedback

Submit an invalid status-contract proposal.
Inspect the validator's concrete rejection.
Correct it and validate again.
Present evidence for a human decision.
A passing check is not permission to merge.

GitHub: Copilot extensions / Hector: Vibe testing and false confidence

Speaker notes
Recording cue: show the invalid and corrected proposals and actual validator responses. End with a proposed handoff packet and a deliberate approve/reject decision; rejected approval leaves changes unapplied. No agent should weaken the contract just to get green. Static fallback: inputs, failure reasons, passing result, and review checklist. Script and final media approval are still required. Transition: make review an explicit design element.

Human-in-the-loop, not human-as-cleanup

Boundary	Review question
Before execution	Is this the right scope and authority?
After investigation	Does the evidence support the finding?
Before delivery	Do we accept the change and its risk?

Make rejection a normal, tested path.

GitHub: Dynamic workflows / gh-aw: Safe outputs

> **Speaker notes**
> Purpose: connect to the invitation's human-in-the-loop promise. Decide what can happen automatically and what remains a proposal. Approval must apply to the exact artifact and consequence, not a different result produced after review. Pausing a run is useful but does not replace an approval policy. Transition: show failure modes hidden by polished demonstrations.

What a polished demo can hide

Untrusted issue text becomes an instruction.
Context drifts; workers duplicate or conflict.
Tests turn green by weakening assertions.
Credentials, rate limits, retries, or costs surprise you.
Evaluate failures as deliberately as successes.

gh-aw: GitHub tools and allowlists / gh-aw: Network policy / Hector: Vibe testing and false confidence / Hector: Context engineering

Speaker notes
Purpose: balance inspiration with concrete gotchas. Treat fetched content as data, enforce authority outside prompts, isolate edits, and inspect meaningful assertions. Coverage is a useful gate but not proof of quality. Separate observations from hypotheses and preserve missing evidence. These are engineering risks to evaluate, not claims that one tool solves every risk. Transition: choose realistic first applications.

Scenarios worth trying

Need	First bounded artifact
Issue intake	Classification with evidence
Documentation drift	A reviewed update proposal
Failing CI	A diagnosis and reproducer
Release readiness	A checklist with explicit blockers
Migration	An isolated, tested change set

GitHub Agentic Workflows / GitHub: Dynamic workflows / Hector: Agentic Workflows hands-on guide

Speaker notes
Purpose: offer choices for different companies without prescribing one architecture. Match each candidate to data access, available validation, review capacity, and acceptable blast radius. Start with a useful artifact that someone can reject. Do not begin with unattended production remediation just because the platform can call an API. Transition: adoption changes the operating model too.

Scale the operating model

Ownership / Who maintains the workflow?

Isolation / What data, tools, and workspaces are shared?

Evidence / How do we know it helped?

Economics / What does a useful result actually cost?

gh-aw: GitHub tools and allowlists / gh-aw: Network policy / Hector: Three pillars of Agentic DevOps / Hector: The maturity curve

Speaker notes

Purpose: address the invitation's scaling promise. Version prompts and tool contracts, assign owners, evaluate regressions, and route feedback into improvements. Measure accepted useful results, escaped defects, reviewer effort, and observed resource use rather than agent count or generated lines. Enterprise policy, data residency, and approvals must be checked in the actual organization. Transition: give the audience a small adoption action.

Pick one measurable workflow

Read-only

Report with evidence

Propose

Human-reviewed artifact

Automate narrowly

Known effects and rollback

Keep the simplest level that delivers value.

GitHub Agentic Workflows / GitHub: Dynamic workflows / Hector: The maturity curve

Speaker notes
Purpose: end with a practical adoption ladder. Choose one repetitive task, define its acceptance criteria, expose only necessary capabilities, and test the failure paths. Expand authority based on evidence, not novelty. Some organizations should stay at read-only reports, and that is a valid outcome. Transition: summarize with one durable idea.

Define the outcome.

Bound the action.

Verify the result.

Your next engineering artifact might be the workflow.

Explore the handout, notes, and sources

Hector: Three pillars of Agentic DevOps / GitHub: Dynamic workflows / GitHub Agentic Workflows

Speaker notes
Purpose: close the conceptual talk and open discussion. Ask the audience to think of one task where a bounded proposal would remove repetitive effort without removing ownership. The companion handout contains the public sources and qualifications. Do not imply the three proposed recordings have been produced. Transition: Q&A; use appendix slides only when useful.

APPENDIX / AVAILABILITY

Check the recording environment

Dynamic workflows: **public preview**
CLI dynamic workflows: experimental features enabled
CLI extensions: **experimental**
Check current eligibility, policy, permissions, and billing
Documentation reviewed for this presentation on November 12, 2026.

GitHub: Dynamic workflows / GitHub: Using dynamic workflows / GitHub: Copilot extensions / Copilot CLI command reference

Speaker notes
Purpose: retain visible feature-status caveats without burdening every conceptual slide. Announcements describe launch availability; current documentation can add exceptions, including legacy annual-plan billing. Record the runtime and tested behavior with each clip. Availability and controls may change after this review, so revalidate before presenting.

APPENDIX / CONTROL CHECKLIST

Inspect the effective configuration

Layer	Ask
GitHub tools	Which groups and individual calls?
Authentication	Which repositories and permission scopes?
Network	Runner egress and hosted retrieval?
Safe outputs	Which writes and maximum counts?
Policy	What requires a separate human decision?

gh-aw: GitHub tools and allowlists / gh-aw: Safe outputs / gh-aw: Network policy

Speaker notes
Purpose: provide a durable checklist instead of a potentially stale configuration snippet. The old public demo uses broad defaults and historical settings; do not copy it unchanged. Compile and validate the exact current frontmatter and inspect generated jobs. Explicitly account for implicit output defaults. Keep credentials out of configuration committed to source.

APPENDIX / CONCEPTUAL PSEUDOCODE

A reusable investigation process

```
validate input and collect bounded evidence
ask independent investigators distinct questions
preserve incomplete results explicitly
combine and verify findings
pause for review of this exact proposal
return evidence; do not deploy
```

Illustration of control flow, not copy-paste API syntax.

GitHub: Dynamic workflows / GitHub: Using dynamic workflows

Speaker notes

Purpose: explain orchestration without claiming a snippet is executable. Production code should handle ordinary failed results separately from cancellation, validate semantics, version durable keys, and make retries safe. A workflow's structured result is an interface contract, not evidence that the finding is true. Use the bundled guide for the installed runtime.

APPENDIX / CHOOSE A SURFACE

What do you actually need?

Requirement	Start with
Reusable instructions and resources	Skill
External service tools	MCP
A lifecycle command	Hook
Session-aware JavaScript capability	CLI extension
Repeatable code-defined process	Dynamic workflow
Distributable capability bundle	Plugin

GitHub: Copilot extensions / GitHub: Dynamic workflows

Speaker notes

Purpose: give the audience a practical product vocabulary. These surfaces compose; they are not competing maturity levels. A dynamic workflow is hosted in an extension, and a plugin may distribute it. Keep authority, authentication, and validation independent

of packaging. Avoid conflating local extension processes with the retired GitHub App integration.

APPENDIX / PRESENTATION CONTROLS

Take the material with you

Arrow keys navigate; Esc opens overview.
S opens speaker view; F toggles fullscreen.
The handout includes public notes and source links.
Download slide and handout PDFs from the toolbar.
All notes are public. Demo sections are currently **storyboards**.

reveal.js speaker view / reveal.js PDF export

Speaker notes
Purpose: explain the companion site honestly. Speaker view keeps notes out of the projected display, but static source and the handout expose those notes publicly. PDFs replace video with readable storyboard content until approved recordings exist. For local delivery, serve the built site from a local static web server; opening index.html directly is not the supported route.

Research sources

GitHub: Dynamic workflows (official documentation)
GitHub: Using dynamic workflows (official documentation)
Dynamic workflows announcement (official announcement, October 1, 2026)
GitHub: Copilot extensions (official documentation)
Copilot CLI command reference (official documentation)
Copilot CLI programmatic reference (official documentation)
GitHub Agentic Workflows (official project documentation)
gh-aw: Tools (official project documentation)

gh-aw: GitHub tools and allowlists (official project documentation)
gh-aw: Safe outputs (official project documentation)
gh-aw: Network policy (official project documentation)
Introducing GitHub Copilot (official announcement, June 29, 2021)
Copilot general availability (official announcement, June 21, 2022)
Copilot CLI general availability (official announcement, February 25, 2026)
Hector: Agentic Workflows hands-on guide (author experience; historical examples)
Hector: Context engineering (author perspective)
Hector: Three pillars of Agentic DevOps (author perspective)
Hector: Knowledge, capability, enforcement (author architecture experience)
Hector: CLI extensions guide (author experience; recheck APIs against installed runtime)
Hector: The maturity curve (author perspective, not a universal maturity standard)
Hector: Vibe testing and false confidence (author case study)
Public Task Tracker workflow examples (public example repository)
reveal.js speaker view (official presentation documentation)
reveal.js PDF export (official presentation documentation)