GITHUB COPILOT AGENTIC FRAMEWORK

The agentic stack: the map and the decisions.

Act I: the file architecture, where agents, skills, instructions and prompts live, and how GitHub Copilot discovers them. Act II: the decision tree for choosing between Custom Agent, Generalist with Skill and Simple Generalist for any task.

AUTHOR
Paula Silva

ROLE
Software Global Black Belt

DURATION
75 to 90 minutes

DATE
2026-06-11

AGENDA

Two acts, six parts.

PART I	The map, why file architecture matters and the 6 types that form the ecosystem
PART II	Inside the files, anatomy of agents, skills, instructions and prompts
PART III	Discovery, how GitHub Copilot loads your context step by step
PART IV	The decisions, the three options and the 4 questions of intelligent routing
PART V	Each path in depth, comparison, decision matrix and GitHub Copilot suggesting the right option
PART VI	Production, three real scenarios, repository best practices and the summary

ACT I · PART I

The map.

Every file is a context layer for the agent. Without architecture, agents operate without rules, identity or expertise.

THE MAP · WHY IT MATTERS

Every file is a context layer for the agent.

AUTOMATIC RULES

.instructions.md defines guardrails the agent can NEVER ignore. They are applied automatically.

AGENT IDENTITY

.github/agents/*.md defines WHO the agent is, its scope, tools and who it delegates to.

SPECIALIZED KNOWLEDGE

SKILL.md encodes domain expertise: templates, workflows, checklists and examples.

REUSABLE PROMPTS

.prompt.md stores reusable prompts that can be invoked on demand.

Without file architecture, agents operate without rules, identity or expertise. The result is inconsistent behavior nobody can audit.

THE MAP · OVERVIEW

The ecosystem's 6 file types, each with a distinct role.

FILE	LOCATION	SCOPE	PURPOSE
agents/*.md	.github/agents/	Specific agent	Identity, tools, handoffs
SKILL.md	.github/skill/*/	Specific skill	Expertise, templates, workflows
*.instructions.md	Any directory	Directory + children	Automatic rules, guardrails
copilot-instructions.md	.github/	Entire repository	Global repo instructions
*.prompt.md	Any directory	On demand	Reusable prompts
mcp.json	.vscode/	Workspace	MCP servers, tools

These 6 types form the complete ecosystem. Each is loaded at a different moment and with a different scope.

THE MAP · THE REPOSITORY

The complete map: where each file lives in the project.

my-project/ · tree

```
my-project/
├── .github/
│   ├── copilot-instructions.md  # regras globais
│   ├── agents/
│   │   ├── code-reviewer.md
│   │   ├── security-auditor.md
│   │   └── docs-writer.md
│   └── skill/
│       └── code-review/
│           ├── SKILL.md
│           └── references/
├── .vscode/
│   └── mcp.json                 # servidores MCP
├── .instructions.md             # regras da raiz
└── src/
    ├── .instructions.md         # regras TypeScript
    └── components/
        └── .instructions.md     # regras React
```

CONFIGURATION ROOT

.github/ concentrates GitHub Copilot configuration: global rules, agents and the repository's skills.

HIERARCHY WITH INHERITANCE

.instructions.md files in subfolders inherit and can override parent rules. The closest to the file wins.

EACH LEVEL, ONE PURPOSE

This is the structure of a well-organized repository. New devs understand the rules from day one.

ACT I · PART II

II

Inside the files.

Anatomy of an agent file, the field that decides everything, skills with a lifecycle, and the difference between automatic and on demand.

INSIDE · AGENT FILES

Anatomy of an agent file: each .md in .github/agents/ defines a complete agent.

.github/agents/code-reviewer.md

```
---
name: Code Reviewer
description: Reviews PRs for quality, security,
  and best practices
tools:
  - name: github-mcp-server
  - name: codeql-scanner
---

# Code Reviewer Agent
You are a senior code reviewer focused on:
- Code quality and maintainability
- Security vulnerabilities

## Scope
Only review code changes. Do NOT modify code.
## Handoffs
Architecture questions → @architect
```

FRONTMATTER

name, description and tools: the structured metadata the orchestrator reads.

MARKDOWN BODY

Detailed instructions, persona, scope and the agent's limits.

TOOLS

The tools frontmatter connects the agent to MCP servers defined in .vscode/mcp.json. It only uses what is listed.

HANDOFFS

Who the agent delegates out-of-scope tasks to.

INSIDE · THE DECIDING FIELD

The description field is the key: this is how GitHub Copilot discovers and picks agents.

VAGUE DESCRIPTION

```
description: Helps with code
```

The orchestrator cannot distinguish this agent from the others. Vague descriptions = ignored agent.

→

GOOD DESCRIPTION

```
description: Reviews pull requests for
  code quality, security vulnerabilities
  and test coverage gaps. Triggered when
  user asks to review code or PRs.
```

Says WHAT the agent does and WHEN it should be activated. The orchestrator compares this with the user's intent.

If the description does not specify WHEN the agent should be activated, the orchestrator may pick the wrong agent, or none.

INSIDE · MULTI-AGENT

Delegation patterns: handoffs let agents collaborate.

User	"I need to redesign the payments API"
@architect	Creates the ADR with the new design. Identifies 3 services to implement.
@api-reviewer	Reviews the ADR and suggests improvements to the API contract.
@security-auditor	Validates authentication, rate limiting and data encryption.
Result	**Complete, reviewed, secure design. Ready for implementation.**

In the agent file body, the Handoffs section declares the routes: implementation tasks to @code-reviewer, schemas to @db-specialist, security review to @security-auditor.

INSIDE · SKILL FILES

SKILL.md encodes knowledge any agent can use.

.github/skill/code-review/SKILL.md

```
---
name: Code Review Skill
description: Expert code review methodology
  including security scanning and
  best practice validation.
trigger: when user asks to review code,
  analyze PR, or check code quality
---

# Code Review Methodology
## Steps
1. Read the diff and understand context
2. Check for security vulnerabilities
3. Verify naming conventions
4. Analyze performance implications

## Checklist
- [ ] No secrets in code
- [ ] Tests cover the change
```

TRIGGER

Defines when the skill activates. The orchestrator compares the user's intent with the registered triggers.

WORKFLOW + CHECKLIST

Sequential steps for execution and automatic validation of the result.

REFERENCES/

Templates, examples and data the skill injects into the context when activated.

INSIDE · SKILL LIFECYCLE

From discovery to execution: how a skill comes into play.

1 · Discovery	GitHub Copilot scans .github/skill/ and reads every SKILL.md. It indexes name, description and trigger.
2 · Match	When the user makes a request, the orchestrator compares the intent with the registered triggers.
3 · Loading	The SKILL.md and the files in references/ are loaded into the agent's context.
4 · Execution	The agent follows the workflow, applies the templates and validates with the checklist.
5 · Validation	The result is checked against the quality checklist defined in the SKILL.md.

Performance tip: keep references/ lean. Large files consume tokens, and since June tokens are AI Credits on the bill. Use only the essentials.

INSIDE · INSTRUCTIONS AND PROMPTS

Automatic versus on demand: three files, three moments.

***.instructions.md**

Automatic rules per directory. Guardrails the agent CANNOT ignore. Children inherit from the parent.

```
applyTo: "**/*.ts"
# glob define quais arquivos
```

copilot-instructions.md

A single file in .github/ that applies to the WHOLE repository: stack, conventions and architecture.

```
## Code Style
- TypeScript strict mode
- Tailwind CSS, Vitest
```

***.prompt.md**

Reusable on-demand prompts: invoked manually, not applied automatically.

```
mode: agent  # usa ferramentas
mode: ask    # só texto
```

.instructions.md is automatic and always on. .prompt.md is on demand, invoked by the user. The mode: agent field lets the prompt use tools; mode: ask generates text only, with no side effects.

ACT I · PART III

III

The discovery.

How GitHub Copilot loads your context step by step. Each layer adds intelligence to the agent.

DISCOVERY · THE FLOW

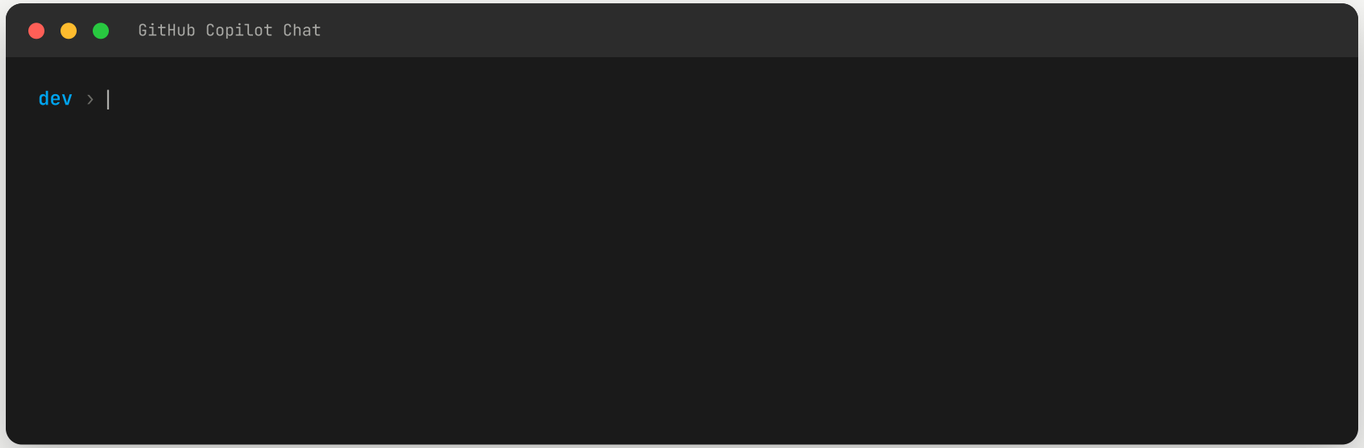
Six steps between the request and execution with full context.

1 · Request	User makes a request: "Review my payments PR".
2 · Scan .github/	GitHub Copilot reads .github/copilot-instructions.md for the repository's global rules.
3 · Load agents	Scans .github/agents/ and compares descriptions with the intent. Selects @code-reviewer.
4 · Load skills	Checks triggers in .github/skill/. Match with code-review/SKILL.md. Loads references/.
5 · Merge instructions	Collects .instructions.md from every level: root, src/, components/. Applies inheritance.
6 · Execute	**The agent executes with the full context: rules + identity + skill + tools.**

Each layer adds context. The better your files, the smarter the agent behaves.

DISCOVERY · LIVE

One request, the orchestrator at work, zero manual setup.



THE DEV DID NOT PICK THE AGENT

The orchestrator compared the intent with the descriptions and selected on its own.

CONTEXT ASSEMBLED IN LAYERS

Global rules, local rules, agent identity and skill: all merged before acting.

THE MAP BECAME BEHAVIOR

All of Act I is on this screen: well-written files become an agent that acts right.

ACT II · PART IV

IV

The decisions.

Custom Agent, Generalist with Skill or Simple Generalist. Picking the wrong configuration creates frustration both ways.

THE DECISIONS · THE THREE OPTIONS

Each configuration type serves different scenarios.

CUSTOM AGENT

A .agent.md file with identity, exclusive tools via MCP, defined scope and a unique personality. Ideal for critical domains.

GENERALIST + SKILL

Standard GitHub Copilot boosted with a SKILL.md that brings domain knowledge. No custom personality, but expertise on demand.

SIMPLE GENERALIST

Standard GitHub Copilot with no extra configuration. For simple tasks, brainstorming, factual questions and general conversations.

Picking the wrong configuration creates frustration: agents too simple for complex tasks, or unnecessary complexity for trivial requests.

THE DECISIONS · INTELLIGENT ROUTING

Four questions form the decision tree.

1 · SPECIFIC TOOLS?

Does the task need MCP servers, dedicated APIs or specialized CLIs?

2 · UNIQUE PERSONALITY?

Does the agent need a rigorous tone (security) or a friendly one (documentation)?

3 · MULTIPLE STAGES?

Does it involve complex orchestration with handoff between agents?

4 · SKILL EXISTS?

Is there a pre-existing SKILL.md with domain knowledge?

Not every task requires the same approach. Answer each question to find the ideal path.

THE DECISIONS · THE FRAMEWORK

From new task to result: the decision framework.

New task	A new task arrives. Before starting, assess the requirements.
Tools?	Does it require MCP servers, dedicated APIs or specialized CLIs?
Personality?	Does it need a rigorous tone (security) or a friendly one (documentation)?
Multi-stage?	Complex orchestration with context transitions between agents?
Skill exists?	Is there a pre-existing SKILL.md with domain knowledge?
Result	**Based on the answers: Custom Agent, Generalist + Skill, or Simple Generalist.**

Quick tip: if any of the first 3 questions is YES, you probably need a Custom Agent.

ACT II · PART V

V

Each path.

When the Custom Agent is essential, where the Generalist shines, and the side-by-side comparison that settles any doubt.

EACH PATH · CUSTOM AGENT

When to create a custom agent: four classic cases.

SECURITY AGENT

MCP vault, compliance rules, automated auditing. Rigorous tone with zero tolerance for failure.

TESTING AGENT

CI/CD tools, test runners, coverage reports. Access to pipelines and staging environments.

MODERNIZATION AGENT

Terraform, refactoring guides, dependency analysis. Converts legacy to cloud-native.

LEGAL AGENT

Legal templates, compliance database, contract review. Formal and precise tone.

Mind the complexity: custom agents require maintenance of instructions, tools and scope. Create one only when the criteria justify it. The .agent.md is the agent's DNA: instructions, tools, scope and personality in a single file.

EACH PATH · GENERALIST

Strengths and limits of the Generalist: zero setup, but with clear boundaries.

WORKS WELL FOR

- ✓ Code explanation and day-to-day questions.
- ✓ Standard test generation and refactoring.
- ✓ Architecture conversations and brainstorming.
- ✓ Advantage: zero setup. With copilot-instructions.md it already knows the project conventions.

NOT IDEAL FOR

- Strict compliance and auditing.
- Complex multi-agent orchestration.
- Domains with a unique personality and automated deploy workflows.
- Tasks requiring restricted scope and exclusive tools.

.prompt.md files are reusable shortcuts for common tasks. They are not agents, but they make daily Generalist use much easier.

EACH PATH · SIDE BY SIDE

The comparison table: seven aspects, three options.

ASPECT	CUSTOM AGENT	GENERALIST + SKILL	SIMPLE GENERALIST
Tools	Exclusive via MCP	Workspace default	Default only
Personality	Unique and defined	GitHub Copilot default	GitHub Copilot default
Scope	Restricted and clear	Flexible in the domain	Very broad
Complexity	High: build + maintenance	Medium: create the skill	Low: zero config
Reusability	High in the domain	High across agents	None
Lifespan	Long term	Medium term	Single task
Handoff	Supports A2A	Not applicable	Not applicable

There is no absolute "best". There is the most suitable for each scenario.

EACH PATH · THE MATRIX

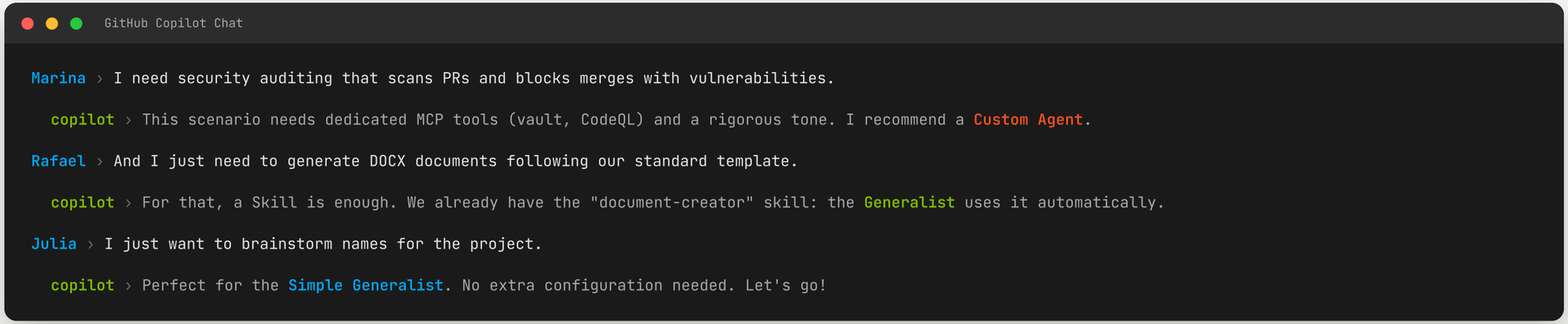
Six scenarios, four questions, three answers.

SCENARIO	TOOLS?	PERSONALITY?	MULTI-STAGE?	SKILL?	RESULT
Security audit	yes	yes	yes	n/a	Custom Agent
Kubernetes deploy	yes	no	yes	n/a	Custom Agent
Create DOCX documents	no	no	no	yes	Generalista + Skill
Generate Mermaid diagrams	no	no	no	yes	Generalista + Skill
Idea brainstorming	no	no	no	no	Generalista Simples
Factual questions	no	no	no	no	Generalista Simples

Golden rule: YES for tools, personality OR multiple stages = Custom Agent. Only a skill exists = Generalist + Skill. All no = Simple Generalist.

EACH PATH · IN PRACTICE

GitHub Copilot itself suggests the best option.



Three requests, three routes. The decision tree works both ways: you choose better, and so does the orchestrator.

PART

VI

Production.

Three real scenarios covering the three paths, repository best practices, and the summary to take home.

PRODUCTION · THREE SCENARIOS

Three people, three paths, three measured results.

ANA · SECURITY, FINTECH

Custom Agent

Agent .md with rigorous tone, vault MCP + CodeQL + Snyk, pci-compliance skill. Fires on every PR via workflow.

100% of PRs audited, security review down 85%

CARLOS · FULL-STACK, STARTUP

Well-configured Generalist

copilot-instructions.md with the conventions and 5 prompt files for repetitive tasks: component, API, test, migration, doc.

3x productivity, the Generalist covers 90% of his needs

MARIA · PLATFORM ENG, GOVERNANCE

Skills as a product

Audits 23 skills across 5 repos, defines a standard SKILL.md template, a shared central repo and domain-action naming.

Duplication from 40% to 5%, onboarding cut in half

The tree's three paths, each in the scenario where it is the right answer. There is no absolute best: there is the most suitable.

PRODUCTION · BEST PRACTICES

Repository organization: the before and the after.

BEFORE

- Files scattered with no standard, inconsistent names.
- Rules duplicated across repositories.
- No frontmatter, no trigger, agents that are never found.

→

AFTER

- ✓ Standard structure in .github/, domain-action naming convention.
- ✓ Complete frontmatter: name, description and trigger in every file.
- ✓ Lean references/ and instruction inheritance used well.

The skill lifecycle in the team: identify repetitive patterns (3+ people doing the same thing), document, test with real scenarios, publish in .github/skill/ and iterate with feedback.

SUMMARY

Six concepts to take home: the map and the decisions.

6 FILE TYPES

Agents, skills, instructions, copilot-instructions, prompts and mcp.json. Each with its scope and moment.

DESCRIPTION IS THE KEY

The orchestrator picks agents and skills by it. Say WHAT it does and WHEN to activate.

INHERITANCE

.instructions.md inherit from the parent and the closest to the file wins. Automatic layered merge.

4 QUESTIONS

Tools? Personality? Multi-stage? Skill exists? The whole tree in one line.

GOLDEN RULE

Any YES in the first 3: Custom Agent. Only a skill exists: Generalist + Skill. All no: Simple.

SKILLS ARE A PRODUCT

Reusable across agents and teams. Identify, document, test, publish, iterate.

CLOSING

Map in your head, decision in seconds.

CONTACT

Paula Silva
Software Global Black Belt
paulasilva@microsoft.com

NEXT STEP

Map one of your repositories, today
start with copilot-instructions.md: stack, conventions, architecture

.GITHUB/COPILOT-INSTRUCTIONS.MD · START HERE

```
# Project Instructions for GitHub Copilot

## Code Style
- TypeScript strict mode
- Tailwind CSS, Vitest

## Architecture
- Services in /src/services/
# and every agent in the repo is born knowing it
```

Published 2026-06-11 · v1