Vibe Coding with XR

Everyone is now a builder

Terry xR. Schussler, Senior Director of Next Generation Devices / Deutsche Telekom

Scan now, read later!

Scan this QRCode which will lead you to my entire presentation and more

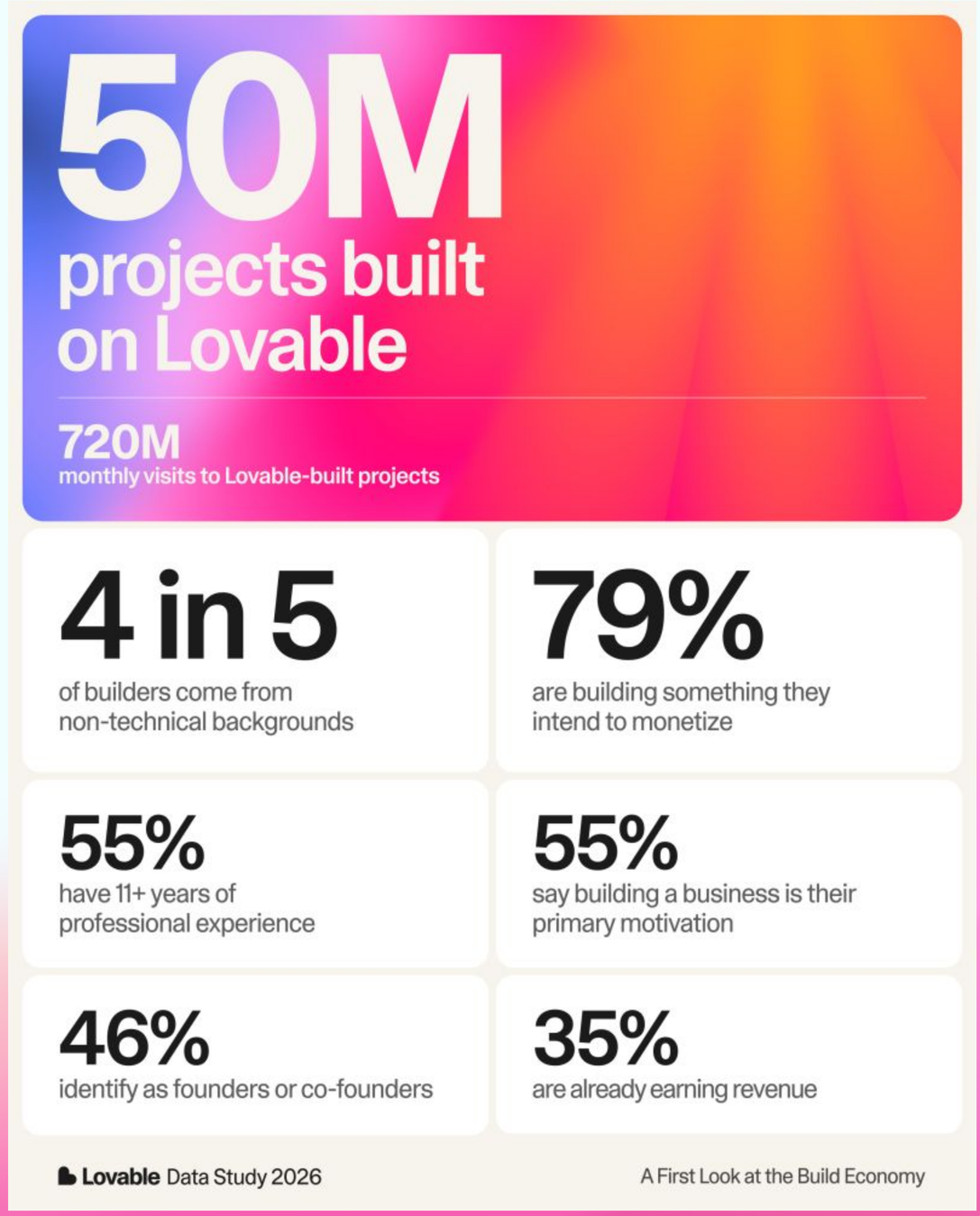
Then put your phone down, relax and listen. No need to take photos of my slides.

The most important tool you have is your imagination.

Vibe Coding with XR...

It’s not just for geeks and nerds anymore!

Vibe Coding with XR means moving from intent to spatial interaction much faster than before.



Vibe Coding is always changing

2025: You were a prompt engineer
(**ask better questions**).

2026: You are a skill user and agent orchestrator
(**assemble better capabilities**).

2027: You will manage systems of agents working across tools and devices
(**direct better autonomous workflows**)

The future belongs to builders who can orchestrate tools, skills, and agents

Phases of Vibe Coding

Phase 0 (**Brainstorm**): Chat

→ explore ideas and use cases

Phase 1 (**Planning**): Chat

→ draft architecture and phased roadmap

Phase 2 (**Scaffolding**): Agent (**with scaffolding skills**)

→ generate initial cross-platform codebase

Phase 3 (**Iteration**): Agent (**with refactor & test-running skills**)

→ evolve features across platforms

Phase 4 (**Polish**): Agent + editor copilot (**with optimization skills**)

→ performance, UX, and code cleanup

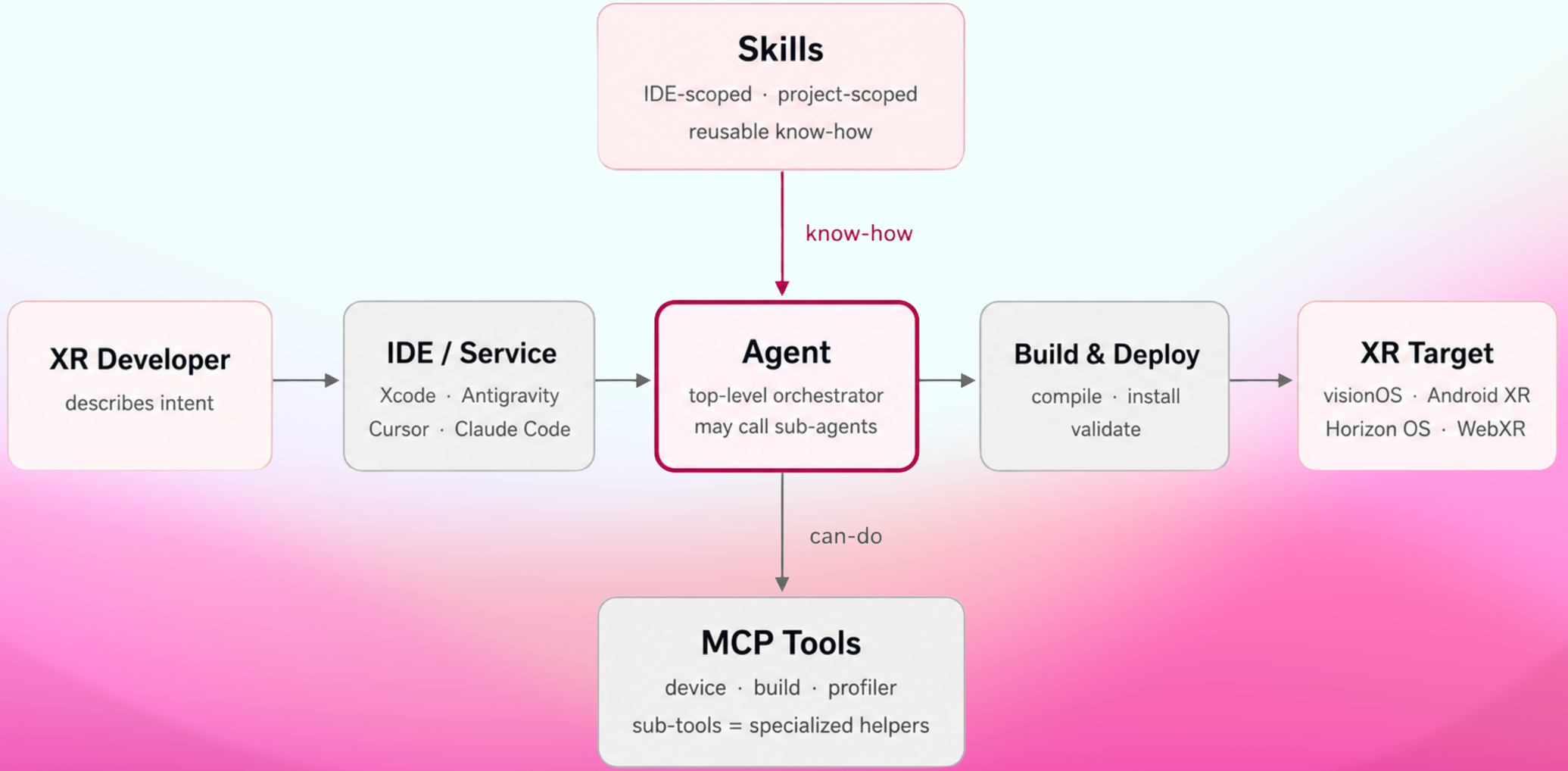
Phase 5 (**Maintenance**): Agent (**with sync & upgrade skills**)

→ keep platforms and dependencies in sync

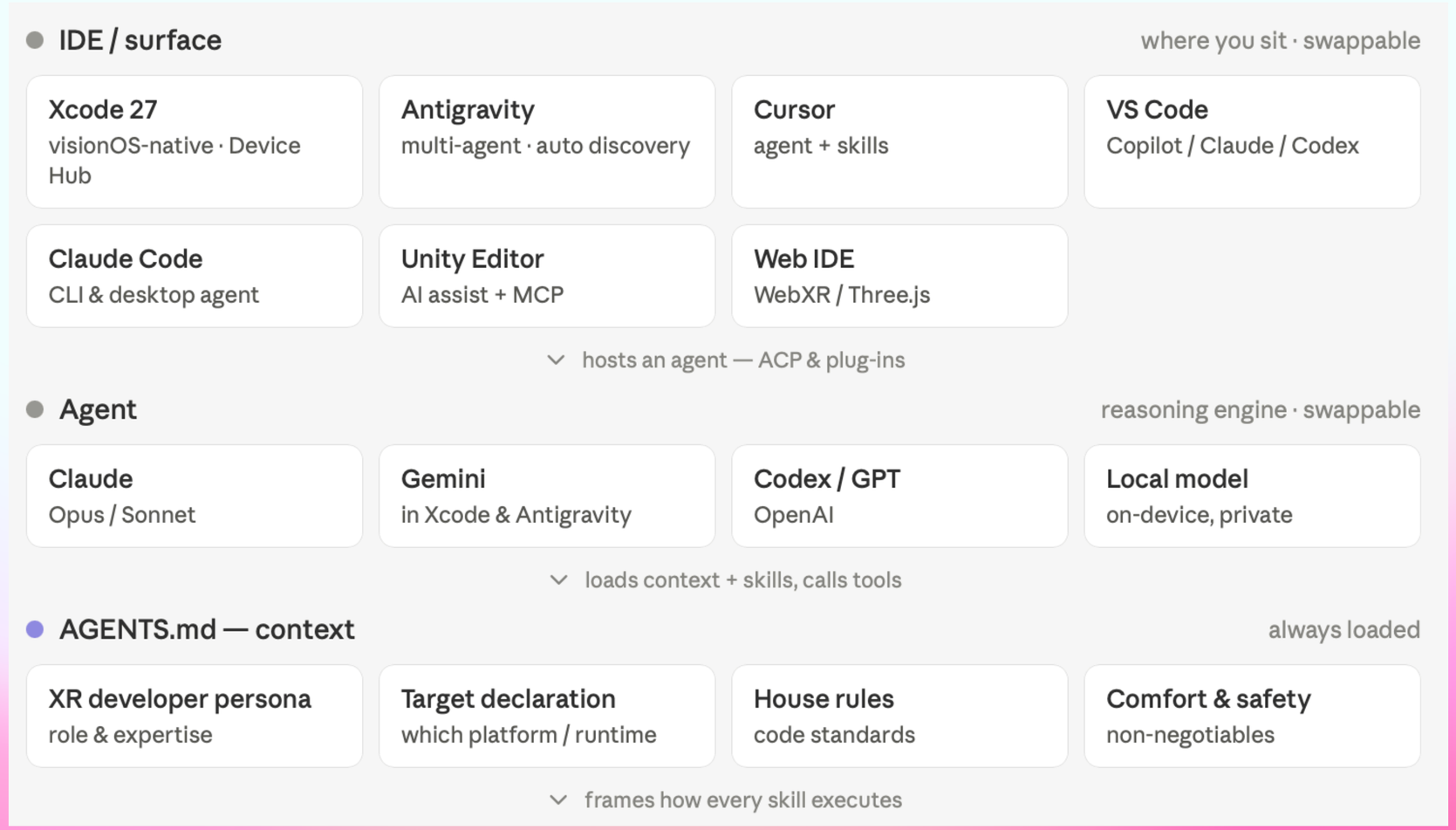
Skills: reusable playbooks the Agent uses in Phases 2–5 (scaffolding, refactor/test, optimization, maintenance)

Vibe Coding with XR in 2026

The agent sits at the center: skills give it know-how, MCP tools give it reach



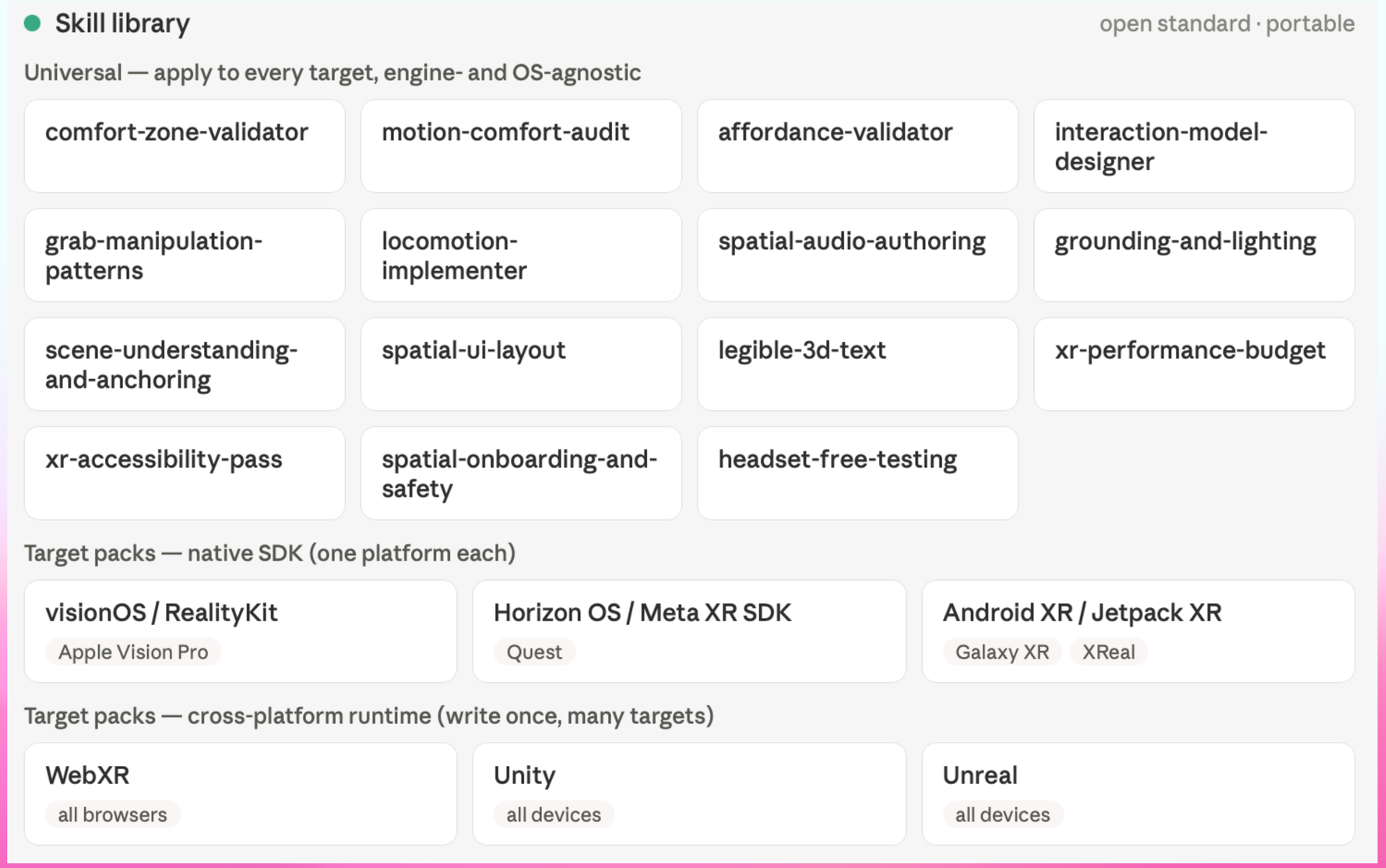
You need an Agent that understands XR



Anthropic's Agent Skills format: https://code.claude.com/docs/en/skills

Agent Skills open standard: https://agentskills.io/home

The Skills to pay the bills



Sites like **claudeskills.info** list Agent Skills in the open SKILL.md format. **Those skills aren't Claude-exclusive** – they work across most skills-compatible agents (Claude Code, Codex/ChatGPT, Cursor, Copilot, Gemini CLI, etc.), with only minor setup differences.

Agents 💖 Skills

```
my-skill/
├── SKILL.md          # Required: metadata + instructions
├── scripts/          # Optional: executable code
├── references/       # Optional: documentation
├── assets/           # Optional: templates, resources
└── ...               # Any additional files or directories
```

Agents load skills through progressive disclosure, in three stages:

1. **Discovery:** At startup, agents load only the name and description of each available skill, just enough to know when it might be relevant.

2. **Activation:** When a task matches a skill's description, the agent reads the full SKILL.md instructions into context.

3. **Execution:** The agent follows the instructions, optionally executing bundled code or loading referenced files as needed.

Full instructions load only when a task calls for them, so agents can keep many skills on hand with only a small context footprint.

Credit: https://agentskills.io/home

To MCP or not to... no, you must MCP

MCP tools are the concrete actions an agent can run in your XR toolchain - deploy, simulate, adb, profile, query docs. Skills then tell the agent *when and how* to combine those tools to get real work done.

MCP tools

live actions, not knowledge

Device Hub deploy on-device install AVP · Quest · Android	**Simulator & emulator** no-headset runs all	**adb bridge** sideload + logcat Quest · Android XR	**WebXR emulator** browser devtools WebXR
Build system xcodebuild · Gradle · Unity CLI	**Asset pipeline** Reality Composer Pro · USDZ · glTF	**Performance profiler** Instruments · OVR Metrics · RenderDoc	**Spatial audio validator** localization check
Headset capture recording for QA	**GitHub** version control	**Figma** design import	**Docs & search** SDK references

XR Device Targets overview

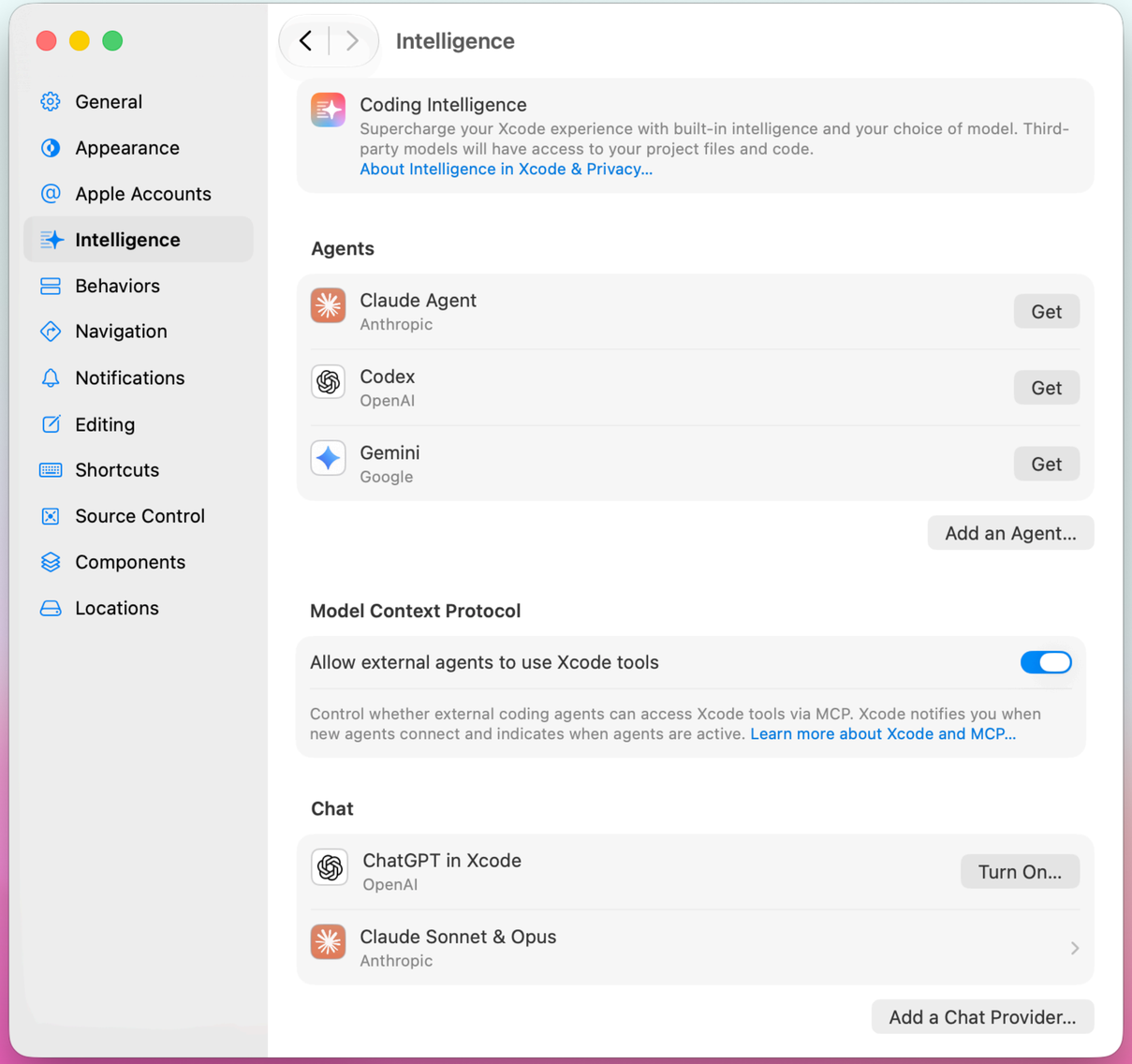
Target	Platform	Best for	Toolchain	Easiest	Tradeoff
Apple Vision Pro	visionOS	Premium spatial UX	Xcode, RealityKit	Native spatial polish	Apple-only
Meta Quest 3 / 3S / Pro	Horizon OS	Deep immersive apps	Unity/Unreal/Meta SDK	Mature headset features	More platform-specific work
Samsung Galaxy XR; XReal Aura	Android XR	Fast headset/glasses prototyping	Android XR SDK, Jetpack XR	Flexible cross-form-factor entry	Earlier-stage platform
Snap Spectacles	SnapOS	Consumer/social AR	Lens Studio, Snap OS	Lens-style AR creation	Narrower ecosystem
Browser / WebXR	WebXR	Fastest demos and links	three.js, A-Frame, PlayCanvas	URL-based distribution	Lower capability ceiling
Pico / others	OpenXR	Broader XR portability	Unity/Unreal/OpenXR	Standardized device reach	Inconsistent device support
Phone AR	ARKit / ARCore	Mainstream mobile AR	ARKit, ARCore	Largest install base	Less immersive

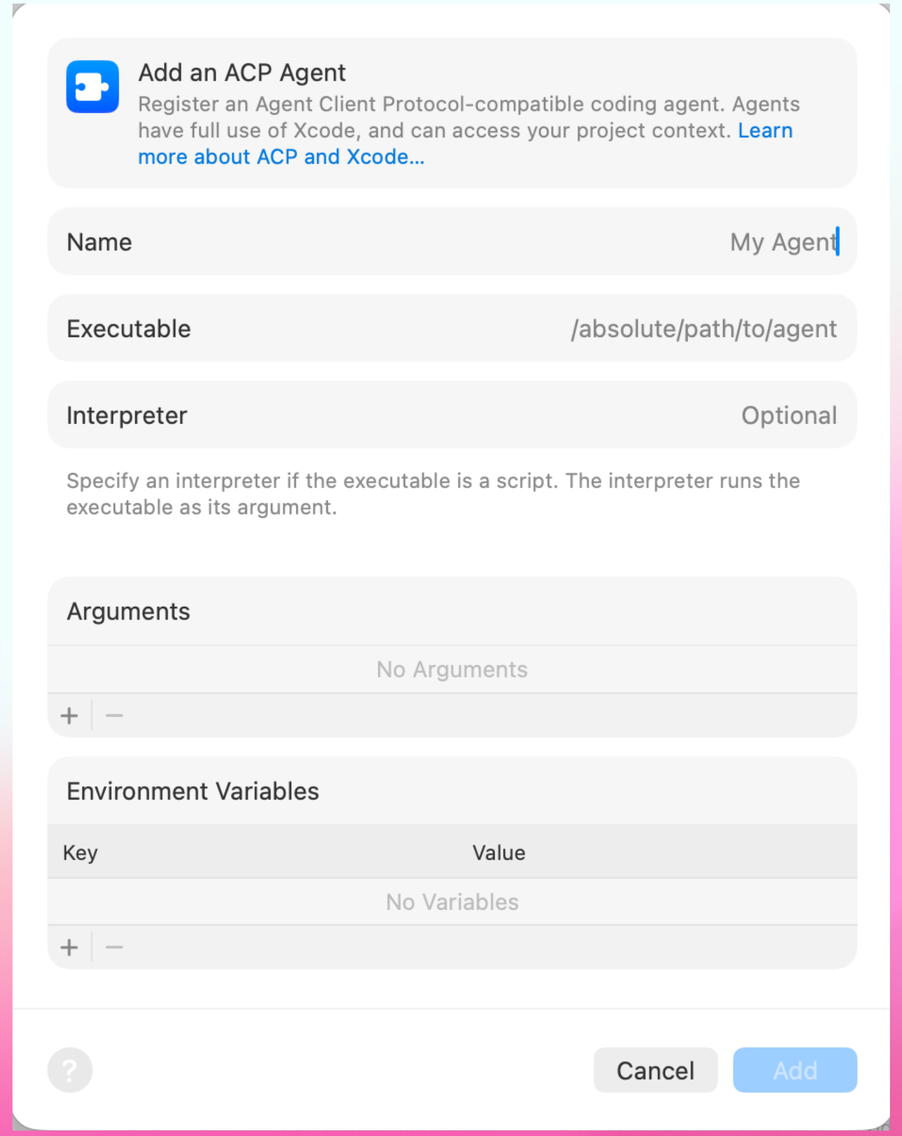
WebXR is the fastest door in; native platforms unlock deeper spatial capability

Case Study: Xcode 27

- XR-capable IDE and a useful lens for **Agents–Skills–Tools.**
- Built-in Apple **agent** + Apple Foundation **Models** on supported Macs.
- Natural baseline for **Vision Pro**, Swift, and Apple Intelligence workflows.
- **Supports external agents** via skills, plug-ins, MCP tools, and ACP.
- Eligible small developers can use Apple PCC models at **no cloud API cost**

Agent setup in Xcode 27





Skills are included in Xcode 27!

- **swiftui-specialist:** Apple's authoritative instructions on SwiftUI structure, data flow, environment, modifiers, and animation.
- **swiftui-whats-new-27:** New SwiftUI APIs and deprecations across the 2027 OS releases, with migration notes.
- **test-modernizer:** Migrates XCTest suites over to Swift Testing.
- **uikit-app-modernization:** Replaces legacy UIKit APIs so apps behave correctly in multi-window and multi-scene setups.
- **c-bounds-safety:** guides adoption and debugging of C's -fbounds-safety bounds-safety extensions.
- **device-interaction*:** verifies behavior through simulator screenshots, view hierarchy inspection, and simulated touch.
- **audit-xcode-security-settings*:** audits and hardens an Xcode project's security build settings.

Tip: You can export these skills to use in Claude Code or other agents! Run this Terminal command:

```
xcrun agent skills export ~/.agents/skills
```

Credit: https://bleepingswift.com/blog/xcode-27-agent-skills-claude-code

* These skills require Xcode or Xcode MCP

AGENTS.md

What belongs in AGENTS.md - the persistent role identity, domain knowledge, code standards, communication style, and gotchas. **This is the "who you are and how you work" context that your Agent loads at the start of every session.**

In Xcode 27, keep AGENTS.md in your repo root (and per-app subfolders if needed); your agent integration loads it automatically as part of the project context.

AGENTS.md (always-loaded context):

- Identity & domain knowledge (role)
- visionOS / RealityKit patterns used in your projects
- Code standards
- Communication style
- Known visionOS gotchas

Future Outlook

Programming and creative skills still matter. You are being augmented, not replaced!

- AI tools are **not magic app generators**; they are powerful collaborators
  - Use them to avoid repetitive development tasks and spend more time on interaction design, storytelling, and overall experience
- **Idea to Outcome:** Focus on rapid prototyping and code scaffolding
- **Be fearless:** Use any API quickly. Experiment often. Try new things
- **Benefit from acceleration:** Avoid ”dirty work” and build way faster

Thank you - remember to download the presentation with many bonus slides!

Bonus Materials

I had more materials than I had time to present, so lucky you - I’ve added them to the presentation materials you’ve just downloaded!

The second most important tool you have is your voice.

You can typically **vibe code 4x faster** than typing by **using your voice**.
My recommendation is to use **Wispr Flow** which works on all your devices.

https://wisprflow.ai/terry

Use this URL on a desktop computer to get a **90 days free trial** (instead of 14).

Note: I get no compensation of any kind - I made arrangements to get you this special promo as a benefit for coming to my talk at AWE.

Ecosystem comparison in detail

Target	Best for	Toolchain	What's easiest	Main tradeoff
visionOS	Premium spatial apps, polished native UX, high-end demos.	Xcode, visionOS SDK, RealityKit, Reality Composer Pro.	Strong native spatial workflow, windows/volumes/full-space patterns, and tight Apple tooling.	Apple-only platform, higher device and team commitment.
Android XR	Fast XR prototyping across glasses and headsets, especially if you already build on	Android XR SDK, Jetpack XR, Unity, OpenXR, WebXR.	Flexible entry points and strong alignment with prompt-native prototyping momentum.	Platform still emerging; distribution and install base are earlier-stage.
Meta Horizon OS	Quest-first immersive apps needing deeper device access and better performance.	Unity/Unreal/native SDKs, Meta XR SDK, Spatial SDK.	Mature headset deployment path, stronger access to hand tracking, passthrough, Scene API, Depth API, and platform features than WebXR.	More setup and more platform-specific implementation work than web approaches.
Snap Specs / Snap OS	Lightweight consumer AR, shared/social experiences, camera-native interactions.	Lens Studio, Snap OS tools, browser/ WebXR support on newer platform path.	Fastest path for Lens-style AR creation and social distribution patterns.	More constrained ecosystem / less cross-platform portability than open web or broader native stacks.
WebXR	Fastest low-friction prototypes, links, demos, lightweight distribution. developers.	Chrome/Browser, three.js, A-Frame, PlayCanvas, model-viewer.	Browser-only access, URL distribution, low setup burden, and broad conceptual portability.	Lower ceiling for performance and limited access to deeper platform features like scene understanding, spatial anchors, and richer native APIs.

WebXR is the fastest door in; native platforms unlock deeper spatial capability

Surface / IDE + Agent recommendations

Target	Recommended surface / IDE	Best agent pairing (today, literally tomorrow this may change)
Apple Vision Pro	**Xcode 27** on macOS with visionOS + RealityKit; optionally pair with a code-focused side IDE (e.g., Cursor or Claude Code) attached to Xcode's agent tooling for refactors/tests.	Apple's built-in coding agent for Swift/visionOS as baseline; augment with **Codex or Claude Code** when you want deeper autonomous refactors or multi-file changes.
Meta Quest 3 / 3S / Pro	**Unity** (Unity 6/LTS) with Meta XR SDK; Unreal mainly for high-end native rendering.	A **Unity-aware IDE agent** (Cursor/Claude Code/VS Code-based) plus Meta's own tooling; use agent mode for scene wiring and refactors, chat mode for shader and C# guidance.
Samsung Galaxy XR; XReal Aura	**Unity** targeting Android XR / OpenXR with Jetpack XR and vendor packages.	Same Unity-focused agent stack as Quest, tuned to Android XR + OpenXR docs; prefer **agent skills that understand device profiles** and performance budgets.
Snap Spectacles	**Lens Studio** for SnapOS/Spectacles projects.	Keep **Lens Studio as the primary surface**, with a general-purpose coding agent (Claude/Codex) in a side IDE for scripting, GLSL, and interaction graphs rather than direct project control.
Browser / WebXR	**VS Code / Cursor / WebStorm / Lovable** with three.js, A-Frame, or PlayCanvas toolchains.	**Web-native coding agent** (Cursor/Claude Code/Copilot) that knows three.js/A-Frame; use **agent mode for scene wiring** and refactors, chat mode for API lookups and math.
Pico / others (OpenXR)	**Unity** (or Unreal) using OpenXR profiles for multi-headset builds.	Same Unity/Unreal agents as Quest, but with **OpenXR-oriented skills** so the agent understands controller profiles and cross-device constraints.
Phone AR (ARKit / ARCore)	**Xcode** for ARKit-heavy iOS work; **Android Studio + Unity** for ARCore/cross-platform pipelines.	Apple's agent plus **Codex/Claude for Swift/SwiftUI when iOS-first**; a Kotlin/Unity-savvy IDE agent for ARCore or cross-platform Unity workflows.

A note about XR Blocks

XR Blocks is Google's lightweight, open-source WebXR framework that sits underneath Vibe Coding XR and Gemini Canvas, turning natural-language "vibes" into runnable XR scenes. It gives Gemini a well-defined building kit - **Three.js-based components for physics, lighting, interaction, and layout**
- so the model can reliably assemble and modify immersive environments instead of just editing generic code.

In your stack, it's best thought of as the XR-specific tools and runtime that make "prompt-to-prototype" possible on Android XR: **incredibly fast** for exploring ideas and behavior, while still **leaving Unity or Android XR SDK as the path** when you need deeper control and full production-grade pipelines.

Its workflow **supports both Chrome desktop simulation and Android XR headset** deployment.

Prompting Best Practices for XR

DO’s and DON’Ts

- ✅ Specify the platform: "This is for Android XR with OpenXR"
- ✅ Include constraints: "Keep frame time under 11ms on Snapdragon XR2"
- ✅ Reference specific APIs and versions: "Use RealityKit's PhysicsComponent in visionOS 26.2 RC”
- ✅ Describe the user experience goal: "User should feel instant tactile feedback"
- ✅ Request code comments: "Explain where platform differences appear”

- ❌ Vague: "Make XR interaction better"
- ❌ Disconnected: "Write code" (without context)
- ❌ Assume capabilities: Don't assume AI knows your codebase structure
- ❌ Skip iteration: If code doesn't work, show the error and re-prompt
- ❌ Ignore testing: Always ask for unit tests alongside code